Τεχνολογικό Εκπαιδευτικό Ίδρυμα Κρήτης

Σχολή Τεχνολογικών Εφαρμογών
Τμήμα Εφαρμοσμένης Πληροφορικής & Πολυμέσων

Πτυχιακή Εργασία

Τίτλος:

Σχεδίαση- Δημιουργία Ιστοσελίδων και η ανάδειξη της διαφοράς μεταξύ Symfony και Drupal

ΝΕΡΤΙΛΑ ΟΝΙΕΑ (ΑΜ: 1789)

Επιβλέπων Καθηγητής: ΠΑΠΑΔΟΥΡΑΚΗΣ ΓΕΩΡΓΙΟΣ
Επιτροπή Παρουσίασης:
Ημερομηνία Παρουσίασης: 00/00/0000

Ευχαριστίες
Αισθάνομαι υπόχρεη και νιώθω ότι πρέπει να ευχαριστήσω ιδιαιτέρως τον καθηγητή μου τον Δρ. Παπαδουράκη Γεώργιο ο όποιος μου ανάθεσε το θέμα αυτό και με βοήθησε στην συνεχεία για την ολοκλήρωση του. Επίσης ευχαριστώ τον Γιάννη Στεφανή ο όποιος με την σειρά του έγινε πολύτιμη πηγή πληροφοριών και βασικών αρχών του Drupal και Symfony ανάλογα .

Σύνοψη

Αυτή η πτυχιακή εργασία αναφέρεται στην εκμάθηση του Symfony (open source framework) και του Drupal (CMS) οι οποίες είναι γλώσσες που χρησιμοποιούνται για την δημιουργία ιστοσελίδων, και εστιάζεται σε κάποια σημεία όπως ακολουθεί παρακάτω : Στην πλήρη παρουσίαση των γλωσσών Symfony & Drupal και των λειτουργιών της.Επίσης στην ανάλυση του περιβάλλοντος τους, και στην αξιοποίηση ενός ηλεκτρονικού ιστού μικρής κλίμακας, γραμμένου και στης δυο γλώσσες αυτές ξεχωριστά (Symfony & Drupal). Το όλο εγχείρημα αποτελεί ένα εύκολο οδηγό ο όποιος θα βοηθήσει για μια καλή αρχή αφομοίωσης για όποιον επιθυμεί να μάθει Symfony & Drupal. Αυτός ο οδηγός θα διευκολύνει την κρίση του χρήστη ώστε εκείνος να διαλέξει ποια γλωσσά θα χρησιμοποιήσει ως εργαλείο για την αποπεράτωση της εργασίας του.

Abstract

This study's purpose is for learning Symfony (open source framework) and Drupal, which are languages that are used for building webpages, and focuses on some issues as follows below: The full presentation and functionality of Symfony and Drupal languages. The analysis of their environment, and the development of a small web site written in both languages (Symfony and Drupal) The project is an easy guide that will help anyone who wishes to learn Symfony and Drupal. This guide will help the user to choose which language can be used as a tool for his project.

Περιεχόμενα

Πίνακας εικόνων:

1. Εισαγωγή

Στο κεφάλαιο αυτό αρθρώνονται γενικές πληροφορίες για αυτή την πτυχιακή εργασία.

Πιο συγκεκριμένα ακολουθεί μία ανάλυση της πτυχιακής εργασίας, των αίτιων τα οποία μας παρακίνησαν στην επιλογή του θέματος καθώς και μία περιγραφή και περίληψη των διάφορων στοιχείων της και της εφαρμογής τους .

1.1 Περίληψη

Ο βασικός στόχος αυτής της πτυχιακής είναι η διερεύνηση πρωτοποριακών μεθόδων ανάπτυξης διαδικτυακών εφαρμογών. Επίσης η αναφορά στην ανάπτυξη τακτικών εφαρμογής για τη χρήση του Symfony και του Drupal και η υπόδειξη των διαφορών μεταξύ τους.

Το Symfony είναι ένα open source framework ανάπτυξης εφαρμογών βασισμένο στην γλώσσα προγραμματισμού PHP σε περιβάλλον LAMP. Υλοποιεί ολοκληρωμένες μεθόδους ανάπτυξης, ελέγχου και deployment μιας εφαρμογής. Είναι κατάλληλο για υλοποίηση τεχνικών ανάπτυξης βασισμένο σε scum/XP Programing τεχνικών. Στην βάση του χρησιμοποιεί MVC αρχιτεκτονική πάνω σε ένα αντικειμενοστραφές μοντέλο της βάσης δεδομένων μέσω ενός στρώματος ORM. Το Symfony είναι επηρεασμένο από το Ruby On Rails και μεταφέρει την εμπειρία από το RoR στην γλώσσα PHP 5.

Το Drupal είναι ένα πρόγραμμα σύγχρονου συστήματος διαχείρισης περιεχομένου (CMS) ανοικτού κώδικα. Ποιό συγκεκριμένα χρησιμοποιήθηκαν οι γλώσσες προγραμματισμού ιστοσελίδων PHP,Javascript και CSS. Για την αποθήκευση και εμφάνιση πληροφοριών από την βάση δεδομένων χρησιμοποιήθηκε η MySql και διάφορα queries της. Επίσης θα αναλύσουμε μέσα από την πρακτική εφαρμογή, βήμα προς βήμα όλες της λεπτομέρειες που θα μας βοηθήσουν να κατανοήσουμε τον τρόπο λειτουργίας του Symfony και του Drupal και πως αναπτύσσονται οι διάφορες λειτουργίες του. Αυτό θα μας βοηθήσει και στην κατανόηση της γλώσσας προγραμματισμού PHP που χρησιμοποιείται ευρέως στο διαδίκτυο σήμερα από χιλιάδες προγραμματιστές στον κόσμο.

1.2 Κίνητρο για διεξαγωγή της εργασίας

Στις μέρες μας, το Internet και η τεχνολογία, αναπτύσσονται με αλματώδη βήματα σε ένα κοινωνικό και επαγγελματικοί περιβάλλον που όλο και περισσότερο εξαρτάται και έχει ανάγκη από αυτά. Αυτή η ανάγκη υποδείχνει τους λόγους ύπαρξης νέων και σύγχρονων δομών, με όλο και ποιο ανεπτυγμένες υπηρεσίες για την ανεύρεση άμεσων τρόπων που θα οδηγήσουν στο καλύτερο και γρηγορότερο αποτέλεσμα .

Έτσι, συγκεκριμένα είναι σκόπιμο να βρεθεί ένας τρόπος όποιος θα κάνει δυνατόν της δημιουργία μιας ιστοσελίδας ανάλογα με τις ιδιαίτερες συνθήκες της κάθε περιπτώσεις .

Τα συμπεράσματα βγαλμένα μετά από την ανάλυση των εφαρμογών του Symfony και Drupal αναδεικνύουν τις διαφορές μεταξύ αυτόν των δυο και συγκεκριμένα αποδεικνύουν ότι το Symfony αναφέρεται σε χρήστες με ιδιαίτερες γνώσεις ενώ το Drupal με τη σειρά του, σε χρήστες χωρίς ιδιαίτερες γνώσεις. Στο υλικό αυτής της εργασίας, πιο κάτω, θα συναντήσουμε διάφορες παραλλαγές αυτού.

1.3 Σκοπός και Στόχοι εργασίας

Σκοπός της εργασίας αυτής είναι η εκμάθηση του σχεδιασμού και προγραμματισμού μιας ιστοσελίδας χρησιμοποιώντας το framwoek symfony και το εργαλείο ανοικτού κώδικα Drupal και η κατανόηση του τρόπου λειτουργίας τους.

Επίσης η αφομοίωση κάποιων βασικών γλωσσών προγραμματισμού (πχ. HTML, η PHP,η SQL, CSS και η Javascript) που απαιτούνται για την δημιουργία ιστοσελίδων. Μέσα από την ανάλυση θα

γίνει κατανοητή η δομή τους, η ιεραρχία και το σύστημα αρχείων τους, προκειμένου να χειριστούν αποδοτικά και να βοηθήσουν στην αποπεράτωση του κάθε έργου ανάλογα με την περίπτωση.

1.4 Δομή εργασίας

- **Η δομή της εργασίας μας είναι οι εξής στα πιο κάτω κεφάλαια που ακολουθούν:**

1) Στο 2ο κεφάλαιο αναφερόμαστε στην μεθοδολογία που χρησιμοποιήσαμε για την υλοποίηση της πτυχιακής και αναφερόμαστε περιληπτικά σε θεωρίες των γλωσσών προγραμματισμού που χρησιμοποιήσαμε και στο Symfony και στο Drupal .

- **Για το Symfony**

2) Στο 3^{ο} κεφαλαιο ακολουθει η εγκατασταση του Symfony και τα αρχεια που θα χρειαστουν να τρεξει το Symfony
3) Στο 4^{ο} και στο 5^{ο} κεφάλαιο ακολουθεί η λεπτότερης σχεδίαση του συστήματος,το μοντέλο δεδομένων και η. Αναλύεται η αρχιτεκτονική Model – View – Controller στα συστατικά της, γίνεται αναφορά και αξιολόγηση στο πλαίσιο ανάπτυξης (framework) που πρόκειται να χρησιμοποιηθεί κατά την ανάπτυξη (PHP framework) και περιγράφονται οι κλάσεις από τις οποίες αποτελείται το σύστημα. Επίσης, παρουσιάζεται και το σχεσιακό σχήμα της βάσης δεδομένων που αναπτύχθηκε.
4) Στο 6^{ο} και 7^{ο} ,8^{ο} ακολουθη το ιστορικο του project και η δρομολογηση (routing) ακολουθη το Doctrine Query object, κατηγοριες router (δηλ. Ο συνδεσιμος των κατηγοριων)
5) Και στο 9^{ο} ,10^{ο} και 11^{ο} γινεται ο λεχος του Symfony και λειτουργιες ελεγχου και τελος γινετε το from framework

- **Και για το Drupal**

Εισαγωγή στο σύστημα διαχείρισης περιεχομένου
(6) Το 12ο κεφάλαιο περιγράφει την Εισαγωγή στο σύστημα διαχείρισης περιεχομένου,γενικα του Drupla, (Δηλ. Της Τεχνολογική υποδομή που εχει του Drupal)
7) Στο 13^{ο} και 15^{ο} κεφάλαιο αναφερόμαστε στην γενηκη περιγραφή του Drupal ,και στην υλοποίηση .

- **Στο τελος του κεφαλεου κανουμε την συγκριση του Symfony και του Drupal σε εναν πινακα .**

2.Μεθοδολογία Υλοποίησης

Περιγραφή των παραδοχών και της μεθοδολογίες υλοποίησης της εργασίας.

2.1 Μέθοδος ανάλυσης και ανάπτυξης πτυχιακής

Στην υποκατηγορία αυτή θα ασχοληθούμε με την παρατήρηση και πρακτική ανάλυση όλων των βημάτων που θα ακολουθήσουμε για την ολοκλήρωση των ιστοσελίδων με την βοήθεια του Symfony και Drupal ανάλογα. Η αφομοίωση των ξεχωριστών στοιχείων τους, θα μας βοηθήσει να κινηθούμε σε επίπεδα βατά και προσιτά σε μας.

ΜΕΘΟΔΟΛΟΓΙΑ	ΑΝΑΦΟΡΑ ΣΤΗΝ ΒΙΒΛΙΟΓΡΑΦΙΑ
Symfony	www.symfony-project.org/
MVC	www.wikipedia.org/wiki/MVC
ORM	**en.wikipedia.org/wiki/Object-relational_mapping**
Framework	**en.wikipedia.org/wiki/Framework**
Drupal	http://drupal.org/
Modules(Ενότητες)	http://drupal.org/project/Modules
Themes	http://drupal.org/project/Themes
PHP	http://en.wikipedia.org/wiki/PHP, http://www.php.net/
SQL	http://www.sqlcourse.com, http://en.wikipedia.org/wiki/SQL
Javascript	http://en.wikipedia.org/wiki/Javascript
Css	http://en.wikipedia.org/wiki/CSS

Πίνακας 1 - Σχετικές Μεθοδολογίες Ανάπτυξης

2.2 Τεχνολογίες και εργαλεία για την ανάπτυξη των ιστοσελιδων (ΘΕΩΡΙΕΣ):

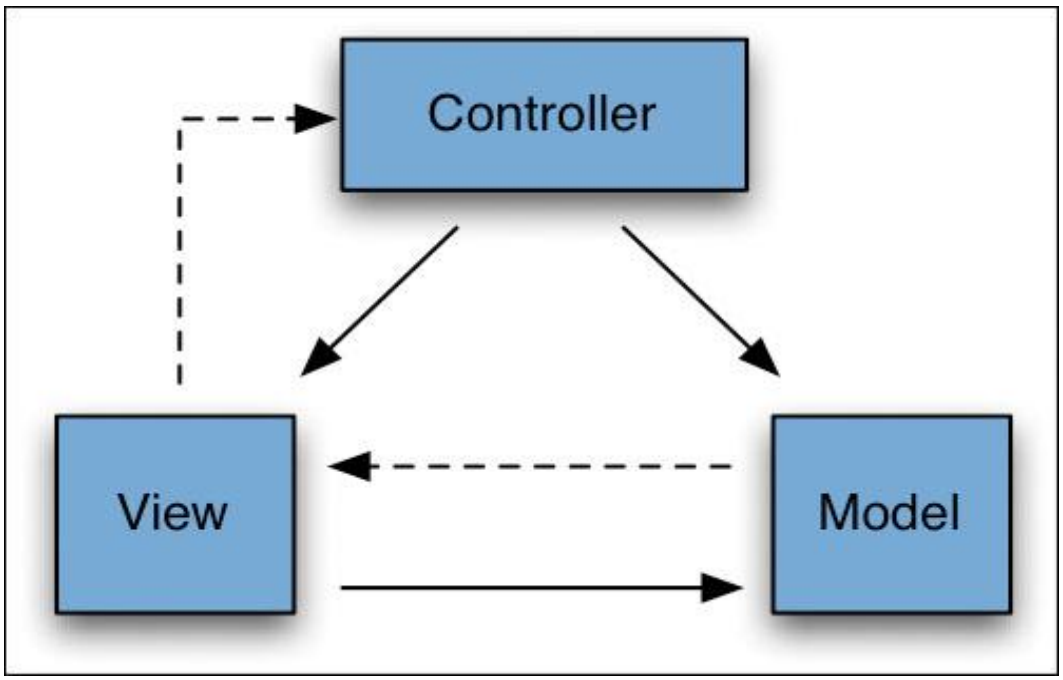
2.2.1 Symfony:

Το Symfony είναι ένα open source framework ανάπτυξης εφαρμογών βασισμένο στην γλώσσα προγραμματισμού PHP σε περιβάλλον LAMP. Υλοποιεί ολοκληρωμένες μεθόδους ανάπτυξης, ελέγχου και deployment μιας εφαρμογής. Είναι κατάλληλο για υλοποίηση τεχνικών ανάπτυξης βασισμένο σε scum/XP Programing τεχνικών. Στην βάση του χρησιμοποιεί MVC αρχιτεκτονική πάνω σε ένα αντικειμενοστραφές μοντέλο της βάσης δεδομένων μέσω ενός στρώματος ORM. Το Symfony είναι επηρεασμένο από το Ruby On Rails και μεταφέρει την εμπειρία από το RoR στην γλώσσα PHP 5.

2.2.2 Η Αρχιτεκτονική του Symfony

Το Symfony είναι βασισμενο στα κλασικα προτυπα σχεδιαζμου ιδιου γνωστα σαν αρχιτεκτονικη Model– View–Controller(MVC)που αποτελουνται από τρεια επίπεδα:
Η αρχιτεκτονική του συστήματος ακολουθεί το αρχιτεκτονικό πρότυπο Model – View – Controller.

Το συγκεκριμένο αρχιτεκτονικό πρότυπο επιτρέπει το διαχωρισμό της λογικής από την παρουσίαση στο υπό ανάπτυξη σύστημα με αποτέλεσμα να προσφέρει μεγαλύτερη ευελιξία στην ανάπτυξη και στη συντήρηση του συστήματος και ευκολότερο έλεγχο των επιμέρους υποσυστημάτων. Μία εφαρμογή που αναπτύσσεται ακολουθώντας τη συγκεκριμένη τεχνοτροπία απαρτίζεται από ξεχωριστές δομικές μονάδες που υλοποιούν τους τρεις διακριτούς ρόλους που ορίζονται από τη συγκεκριμένη αρχιτεκτονική. Τα μοντέλα είναι κομμάτια κώδικα που αναπαριστούν τα δεδομένα της εφαρμογής. Οι όψεις (views) είναι κομμάτια κώδικα που αναλαμβάνουν να παρουσιάσουν το αποτέλεσμα του συστήματος στο χρήστη. Οι ελεγκτές είναι τμήματα κώδικα που δέχονται και διεκπεραιώνουν τις εντολές του χρήστη. Στη σύγχρονη ανάπτυξη δικτυακών εφαρμογών το αρχιτεκτονικό πρότυπο Model – View – Controller βρίσκει ιδιαίτερη εφαρμογή και αποδεικνύεται σταδιακά ως ιδανική λύση για την ανάπτυξη πολύπλοκων εφαρμογών μιας και το μοντέλο ανάπτυξης δικτυακών συστημάτων επιβάλλει εγγενώς το διαχωρισμό της παρουσίασης, η οποία γίνεται μέσω HTML, και της λογικής, η οποία πραγματοποιείται στο διακομιστή της εφαρμογής. Σε αυτό το πλαίσιο ταιριάζει απόλυτα η λογική της λειτουργίας του συγκεκριμένου αρχιτεκτονικού προτύπου. Στο σχήμα που ακολουθεί παρουσιάζονται διαγραμματικά τα επιμέρους στοιχεία της αρχιτεκτονικής και αναπαρίστανται οι συσχετίσεις μεταξύ αυτών. Επιγραμματικά, αξίζει να αναφερθεί ότι ο ελεγκτής έχει πρόσβαση τόσο στις όψεις, τις οποίες δίνει προς απεικόνιση, όσο και στα μοντέλα, από τα οποία αντλεί πληροφορίες ενώ και οι όψεις, με τη σειρά τους, έχουν τη δυνατότητα άντλησης πληροφοριών από τα μοντέλα. Στη συνέχεια ακολουθεί η ενδελεχής ανάλυση του αρχιτεκτονικού προτύπου Model – View – Controller στα επιμέρους τμήματα που το αποτελούν.



Εικόνα 1 : Σχηματικό διάγραμμα του αρχιτεκτονικού μοντέλου Model – View – Controller και των σχέσεων μεταξύ των τμημάτων του.

Με μία πρώτη μάτια της εικόνας μπορεί κάποιος να καταλάβει ότι υπάρχει επικοινωνία μεταξύ του Ελικτή της Προβολής και του Μοντέλου. Όταν ο χρήστης (browser) αιτηθεί μια ενέργεια, τότε αυτή πηγαίνει στον Ελεγκτή. Αυτός με τη σειρά του, αναλόγως το αίτημα, ανοίγει μια επικοινωνία με το Μοντέλο από το οποίο αιτείται κάποια δεδομένα εφόσον τα δεδομένα που ζητήθηκαν είναι

αποθηκευμένα σε βάση. Εάν βρίσκονται στη βάση, το Μοντέλο απαντάει, στέλνοντας τα αποτελέσματα πίσω στον Ελεγκτή. Αφού τα λάβει, καλεί την αντίστοιχη προβολή που θα αναλάβει να εμφανιστεί τα αποτελέσματα. Όταν βρεθεί η απαραίτητη προβολή, τα αποτελέσματα στέλνονται πάλι πίσω στον ελεγκτή ο όποιος και σέσω του διακομιστεί παρουσιάζει τα στοιχειά στον χρηστή. Στην συνέχεια της ενότητας, ακολουθεί η εκτενή ανάλυση των τριών βιβλιοθηκών βάση των οποίων υλοποιείται το MVC σε μια Symfony εφαρμογή.

2.2.3 Model

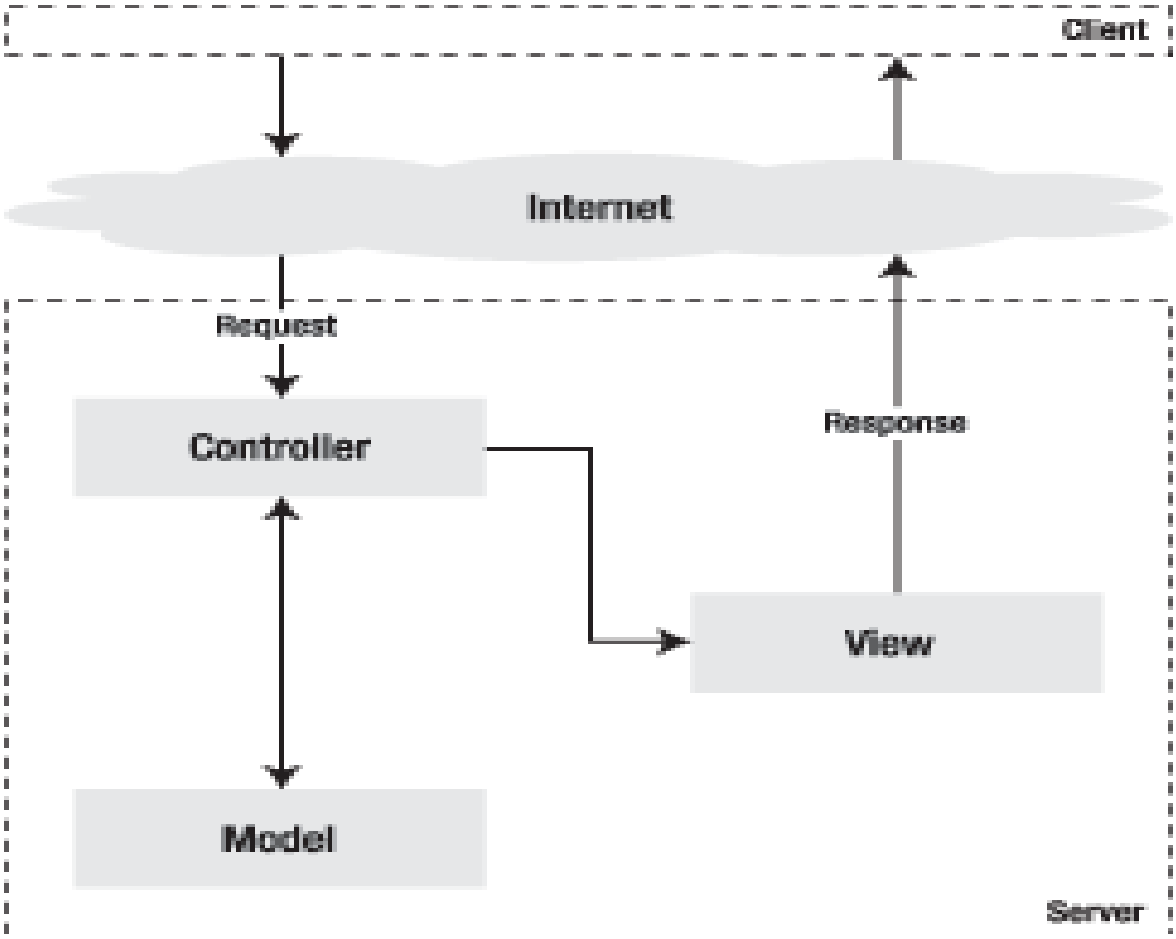
Ένα Μοντέλο αντιπροσωπεύει τις πληροφοριες (δεδομενα) μιας εφραμογης και τους κανονες που χρειαζεται για το χειρισμό αυτων των δεδομένων. Στην περίπτωση του Symfony, τα μοντελα χρησιμοποιούνται κυριως για τη διαχειριση των κανόνων αλληλεπίδρασης με την αντιστοιχη βάση δεδομένων. Στις περισσοτερες περιπτωσεις, ενας πινακας της βασης δεδομενων σας θα αντιστοιχει σε ενα μοντελο δηλ. Αποτελεί την αναπαράσταση των δεδομένων τα οποία χρησιμοποιεί και επεξεργάζεται η εφαρμογή. Τα μοντέλα προσθέτουν λογική στην πληροφορία των εκάστοτε δεδομένων, π.χ. υπολογίζοντας αν σήμερα είναι η μέρα γενεθλίων ενός χρήστη του συστήματος. Όταν ένα μοντέλο μεταβάλλει την κατάστασή του ενημερώνει τα views που σχετίζονται με αυτό ώστε να ανανεωθούν με τη νέα κατάσταση. Τα περισσότερα συστήματα χρησιμοποιούν ένα μηχανισμό μόνιμης αποθήκευσης, π.χ. μία σχεσιακή βάση δεδομένων, για την αποθήκευση των δεδομένων. Τα μοντέλα δεν είναι απαραίτητο να γνωρίζουν για το μέσο αποθήκευσης της πληροφορίας. Παρόλα αυτά είναι πολύ βολικό αν το εκάστοτε μοντέλο γνωρίζει πώς να αναγνώσει και να αποθηκεύσει τον εαυτό του στο μέσο μόνιμης αποθήκευσης της εφαρμογής. Μία υλοποίηση της συγκεκριμένης ιδέας επιτυγχάνεται με το πρότυπο σχεδίασης Active Record.

2.2.4 View

Προβάλει ένα μοντέλο σε μορφή κατανοητή από το χρήστη και προσαρμοσμένη στο μέσο πεικόνισης. Συνήθως, παρέχονται διαφορετικά views για κάθε μοντέλο τα οποία χρησιμοποιούνται ανάλογα την εκάστοτε ανάγκη παρουσίασης της πληροφορίας του μοντέλου. Συνηθισμένο παράδειγμα είναι η ύπαρξη ενός view με συνοπτικές πληροφορίες για το μοντέλο και ενός δεύτερου view, το οποίο απεικονίζει αναλυτικά όλα τα στοιχεία του μοντέλου. Εκτός από τη διάκριση στον τρόπο με τον οποίο απεικονίζεται το μοντέλο, τα views διακρίνονται και με βάση το μέσο στο οποίο είναι σχεδιασμένα να προβάλλονται. Πολλές φορές ένα μεγάλο σύστημα παρέχει στους χρήστες του πολλαπλές δυνατότητες πρόσβασης και απεικόνισης της πληροφορίας, π.χ. μέσω του web browser, μέσω μίας ανεξάρτητης γραφικής εφαρμογής, μέσω μίας ανεξάρτητης εφαρμογής κειμένου ή μέσω εκτυπωμένης πληροφορίας. Σε κάθε διαφορετικό μέσο απεικόνισης αντιστοιχεί και ένα διαφορετικό view, το οποίο λαμβάνει υπόψη του τις ιδιαιτερότητες του μέσου ώστε να εμφανίσει με το βέλτιστο δυνατό τρόπο τα στοιχεία του μοντέλου, π.χ. η προβολή ενός γραφήματος στην κονσόλα του συστήματος ή προβολή μίας έγχρωμης εικόνας σε μία ασπρόμαυρη εκτύπωση είναι περιπτώσεις που απαιτούν ιδιαίτερο χειρισμό από το αντίστοιχο view.

2.2.5 Controller

Λαμβάνει τα μηνύματα και τις εντολές εισόδου από το χρήστη και αποκρίνεται ενεργώντας πάνω στα μοντέλα ή / και εμφανίζοντας κάποιο συγκεκριμένο view. Ο ελεγκτής αποτελεί το συνδετικό κρίκο του συστήματος και είναι υπεύθυνος για τη διεκπεραίωση των εντολών του χρήστη της εφαρμογής. Η συνήθης λειτουργία του είναι η αναγνώριση των εντολών του χρήστη, η άντληση και επεξεργασία της πληροφορίας από τα μοντέλα βάσει των εντολών του χρήστη και η παρουσίαση της ζητούμενης πληροφορίας μέσω του κατάλληλου view.



Εικόνα 2 : Η επικοινωνία μεταξύ των τμημάτων Model – View – Controller.

Η αρχή της αρχιτεκτονικής του MVC είναι να διαχωρίσει τον κώδικα σε τρία layers, σύμφωνα με φύση του. Ο κώδικας λογικής δεδομένων είναι τοποθετημένος μέσα στο model, ο κώδικας της παρουσίασης μέσα στο view, και η λογική της εφαρμογής μέσα στον controller. Άλλα επιπρόσθε σχεδιαστικά πρότυπα μπορούν να κάνουν την εμπειρία του προγραμματισμού ακόμα ευκολότερ Τα επίπεδα model, view και controller μπορούν να υποδιαιρεθούν περαιτέρω.

- Model : Τα δεδομένα μας ή αλλιώς η επιχειρησιακή λογική της εφαρμογής μας
- View : Το επίπεδο παρουσίασης, δηλαδή το πώς εμφανίζονται τα αποτελέσματα / δεδομένα
- Controller: Η διεπαφή με τον χρήστη και ουσιαστικά η λειτουργικότητα της εφαρμογής

Ένα κάλος κανόνας που καθορίζει εάν το View είναι αρκετό ξεκάθαρο είναι ότι πρέπει να περιέχει μόνο το ελάχιστο PHP κώδικα. Για να μπορεί να κατανοηθεί από έναν σχεδιαστή HTML που δεν έχει γνώση PHP. Η όποια γνώστες εντολές στα View είναι αυτές εδώ echo, if/endif, foreach/endforeach, και αυτά είναι όλο.Επίσης δεν πρέπει να υπάρχει κώδικας PHP που να μοιάζει με HTML tags. Όλοι οι λογική είναι μεταφερόμενη στο controller script, και περιέχει μόνο καθαρό κώδικα PHP χωρίς HTML μέσα. Φανταστείτε ότι ο ίδιος Controller μπορεί να ξαναχρησιμοποιηθεί σε μια τελείως ξεχωριστή παρουσίαση ίσως σε ένα PDF η σε δομή XML.

2.3 ORM:

2.3.1 Object-Relational Mapping (ORM) εργαλεία:

Είναι ένα interface που μεταφράζει την object-oriented λογική του Symfony και PHP 5 σε relational λογική των Βάσεων Δεδομένων. Σε αυτήν την κατηγορία βρίσκεται η τεχνική στην οποία θα επικεντρωθούμε. Τα ORM εργαλεία επιτρέπουν την εύκολη και αυτοματοποιημένη αποθήκευση ολόκληρων γράφων από συνδεδεμένα μεταξύ τους αντικείμενα σε μια σχεσιακή βάση δεδομένων. Την αποθήκευση και το φόρτωμα των αντικειμένων τα αναλαμβάνει το ORM εργαλείο, εκτελώντας αυτόματα τα κατάλληλα SQL ερωτήματα. Εμείς απλώς πρέπει να το τροφοδοτήσουμε με την κατάλληλη μετά-πληροφορία για την απεικόνιση των αντικειμένων μας στη σχεσιακή βάση δεδομένων

Εικόνα 3: Object-Relational Mapping (ORM).

2.3.2 Τι είναι τα ORM συστήματα

Ως αντικειμενο-σχεσιακή απεικόνιση (Object-Relational Mapping ή ORM) ορίζεται ένας αυτοματοποιημένος τρόπος διασύνδεσης του μοντέλου αντικειμένων (object model) μίας αντικειμενοστραφούς εφαρμογής με τη σχεσιακή βάση δεδομένων της εφαρμογής, χρησιμοποιώντας μετά-δεδομένα (metadata) για την περιγραφή του τρόπου της διασύνδεσης.

Με τα ORM συστήματα, μπορούμε να φτιάξουμε στην ουσία μια εικονική βάση δεδομένων από αντικείμενα, η οποία μπορεί να χρησιμοποιηθεί από την αντικειμενοστραφή γλώσσα προγραμματισμού που υποστηρίζει το ORM σύστημα.
Ουσιαστικά, τα ORM συστήματα αναλαμβάνουν το επίπεδο DAL της εφαρμογής. Επιτρέπουν την εύκολη αποθήκευση ολόκληρων γράφων από συνδεδεμένα αντικείμενα σε μια σχεσιακή βάση δεδομένων. Το ORM διαχειρίζεται μόνο του αυτή τη διαδικασία, διαμορφώνοντας αυτόματα τα κατάλληλα SQL ερωτήματα προς τη βάση.

2.3.3 Ως αντικειμενο-σχεσιακή απεικόνιση (Object-Relational Mapping ή ORM)

Ορίζεται ένας αυτοματοποιημένος τρόπος διασύνδεσης του μοντέλου αντικειμένων (object model) μίας αντικειμενοστραφούς εφαρμογής με τη σχεσιακή βάση δεδομένων της εφαρμογής, χρησιμοποιώντας μετά-δεδομένα (metadata) για την περιγραφή του τρόπου της διασύνδεσης.
Με τα ORM συστήματα, μπορούμε να φτιάξουμε στην ουσία μια εικονική βάση δεδομένων από αντικείμενα, η οποία μπορεί να χρησιμοποιηθεί από την αντικειμενοστραφή γλώσσα προγραμματισμού που υποστηρίζει το ORM σύστημα.
Ουσιαστικά, τα ORM συστήματα αναλαμβάνουν το επίπεδο DAL της εφαρμογής. Επιτρέπουν την εύκολη αποθήκευση ολόκληρων γράφων από συνδεδεμένα αντικείμενα σε μια σχεσιακή βάση δεδομένων. Το ORM διαχειρίζεται μόνο του αυτή τη διαδικασία, διαμορφώνοντας αυτόματα τα κατάλληλα SQL ερωτήματα προς τη βάση.

- Το πρόβλημα της κληρονομικότητας και του πολυμορφισμού. Ενώ οι αντικειμενοστραφείς γλώσσες προγραμματισμού υποστηρίζουν την έννοια της κληρονομικότητας, οι σχεσιακές βάσεις δεδομένων δεν την υποστηρίζουν. Έτσι, δεν υπάρχει ευθύς τρόπος της αποθήκευσης μιας ιεραρχίας κλάσεων και υποκλάσεων σε μια βάση δεδομένων.
- Με την ίδια αναλογία, υπάρχει το πρόβλημα της πρόσβασης στα αντικείμενα. Ενώ στο αντικειμενοστραφές μοντέλο προγραμματισμού, η πρόσβαση σε ένα αντικείμενο μπορεί να διαφοροποιηθεί ανάλογα με τη διεπαφή (interface) που χρησιμοποιείται προς αυτό, στις βάσεις δεδομένων πρέπει να χρησιμοποιηθούν όψεις (views) για την διαφοροποίηση των προοπτικών με τις οποίες βλέπουμε τα δεδομένα.
- Το πρόβλημα των σχέσεων. Στα μοντέλα αντικειμένων, ένα αντικείμενο μπορεί να σχετιστεί με ένα άλλο, χρησιμοποιώντας μια απλή αναφορά (object reference). Όμως στις σχεσιακές βάσεις δεδομένων, μια σχέση αναπαριστάται μέσω ενός ξένου κλειδιού (foreign key). Οι διαφορές μεταξύ των δύο είναι λεπτές και μικρές, όπως η κατευθυντικότητα.
- Το πρόβλημα των τύπων δεδομένων. Οι βαθμωτοί τύποι δεδομένων σε μια αντικειμενοστραφή γλώσσα προγραμματισμού διαφέρουν συνήθως από τους βαθμωτούς τύπους που μπορεί να αποθηκεύσει μια σχεσιακή βάση δεδομένων. Ένα παράδειγμα είναι η διαφορετική αντιμετώπιση ενός String. Ενώ στις βάσεις δεδομένων τα strings πρέπει να έχουν collation και να έχουν κάποιο μέγιστο μέγεθος, στις αντικειμενοστραφείς γλώσσες δεν υπάρχουν αυτοί οι περιορισμοί γιατί το collation χρειάζεται μόνο σε ορισμένες λειτουργίες (όπως η ταξινόμηση) ενώ το μέγεθος του String αυξομειώνεται στη μνήμη.

- Το πρόβλημα της ταυτότητας. Ενώ στις σχεσιακές βάσεις δεδομένων η ταυτότητα μιας γραμμής εκφράζεται από την τιμή του πρωτεύοντος κλειδιού (primary key), στα αντικείμενα η ταυτότητα συνήθως βασίζεται στην τοποθεσία του αντικειμένου στη μνήμη ή σε μια προσαρμοσμένη μέθοδο ισότητας αντικειμένων.
- Το πρόβλημα της ενθυλάκωσης. Τα αντικειμενοστραφή προγράμματα σχεδιάζονται με μεθόδους που καταλήγουν σε ενθυλακωμένα αντικείμενα των οποίων οι λεπτομέρειες είναι κρυμμένες. Αντίθετα, στις σχεσιακές βάσεις δεδομένων δεν υπάρχουν περιορισμοί για τη σχεδίαση ενθυλακωμένων αντικειμένων που έχουν κρυμμένες λεπτομέρειες, αλλά τα δεδομένα είναι ελεύθερα προς χρήση. Κατ' αυτήν την έννοια, η απεικόνιση των κρυμμένων λεπτομερειών σε μια βάση δεδομένων, καθιστά τις συγκεκριμένες βάσεις δεδομένων ευαίσθητες σύμφωνα με τις αρχές του αντικειμενοστραφούς προγραμματισμού.
- Το πρόβλημα των μηχανισμών προστασίας των ενθυλακωμένων αντικειμένων. Τα RDBMS τείνουν να χρησιμοποιούν μηχανισμούς προστασίας που στηρίζονται σε κανόνες και ρόλους. Κατά την αρχικοποίηση των δεδομένων, τους δίνεται ένα προκαθορισμένο σύνολο από κανόνες και λειτουργίες. Για την τροποποίηση των κανόνων, πρέπει να «αφαιρεθούν» κάποιοι κανόνες από το προκαθορισμένο σετ, γι' αυτό λέμε ότι τα RDBMS στηρίζονται σε «αφαιρετικούς» μηχανισμούς προστασίας. Αντίθετα, στο αντικειμενοστραφές μοντέλο, ένα ενθυλακωμένο αντικείμενο δεν είναι προσβάσιμο από τον έξω κόσμο μέχρι να οριστεί μια διεπαφή (interface) που να παρέχει αυτήν την πρόσβαση.
- Το πρόβλημα των συναλλαγών. Ενώ οι σχεσιακές βάσεις δεδομένων υποστηρίζουν τις μεγάλες συναλλαγές, στον OOP οι συναλλαγές είναι πολύ μικρές και περιορίζονται συνήθως στην ανάθεση μιας τιμής. Στον OOP δεν υπάρχουν οι ανάλογες έννοιες της απομόνωσης και της μονιμότητας.

2.3.4 Χαρακτηριστικά των ORM συστημάτων

Η αυτόματη απεικόνιση των αντικειμένων μιας αντικειμενοστραφούς εφαρμογής σε μια σχεσιακή βάση δεδομένων προσφέρει πολλά πλεονεκτήματα έναντι άλλων τεχνικών πρόσβασης δεδομένων. Μερικά από τα χαρακτηριστικά και τα πλεονεκτήματα των ORM συστημάτων είναι τα εξής:

- Δραματική μείωση του χρόνου ανάπτυξης λογισμικού, του κόστους ανάπτυξης και του κόστους συντήρησης, γιατί το ORM αυτοματοποιεί τη διαδικασία του persistency των αντικειμένων. Ουσιαστικά αναλαμβάνει όλο το DAL που παρουσιάσαμε παραπάνω. Ο προγραμματιστής δεν χρειάζεται να ασχοληθεί με το γράψιμο πολύπλοκων SQL ερωτημάτων και την υλοποίηση του DAL επιπέδου.
- Δίνει τη δυνατότητα στον προγραμματιστή να μοντελοποιήσει τις οντότητες της εφαρμογής βασιζόμενος στις πραγματικές business έννοιες και ανάγκες παρά στην δομή της βάσης δεδομένων.
- Λιγότερο κώδικα σε σύγκριση με την συγγραφή SQL ερωτημάτων μέσα στον κώδικα. Βοηθάει τον προγραμματιστή να επικεντρωθεί στη business λογική της εφαρμογής παρά στην δημιουργία των CRUD SQL ερωτημάτων.
- Μεγαλύτερη ευελιξία στην δημιουργία ερωτημάτων. Τα ORM εργαλεία προσφέρουν συνήθως μια αντικειμενοστραφή γλώσσα ερωτημάτων. Οι γλώσσες αυτές επικεντρώνονται στο μοντέλο αντικειμένων παρά στη μορφή της δομής μιας βάσης δεδομένων. Το ORM εργαλείο θα μεταφράσει αυτόματα την γλώσσα αυτή σε κατάλληλα SQL ερωτήματα.

- Διαφανής πλοήγηση μεταξύ των σχετιζόμενων αντικειμένων (lazy loading). Τα συνδεδεμένα αντικείμενα φορτώνονται δυναμικά από το ORM (ανακτώνται δυναμικά από τη βάση δεδομένων) καθώς ο χρήστης πλοηγείται σε αυτά. Έτσι βελτιστοποιείται η χρήση της μνήμης του εξυπηρετητή.
- Συγχρονισμός. Υποστηρίζονται πολλαπλοί χρήστες που τροποποιούν τα ίδια δεδομένα ταυτόχρονα.
- Caching. Τα αντικείμενα γίνονται cached στην μνήμη για την καλύτερη απόδοση της εφαρμογής και για την μείωση του φόρτου της βάσης δεδομένων.
- Υποστήριξη για συναλλαγές (transactions): Όλες οι αλλαγές σε αντικείμενα γίνονται μέσα σε μια συναλλαγή. Όλη η συναλλαγή είτε θα γίνει committed είτε rolled back.
  Πολλές συναλλαγές μπορούν να πραγματοποιούνται ταυτόχρονα και οι αλλαγές της μίας είναι απομονωμένες από τις αλλαγές της άλλης.
- Διαφανής διατήρηση (transparent persistence): Το ORM εργαλείο παρακολουθεί τις αλλαγές που γίνονται στα αντικείμενα. Όταν κληθεί έπειτα να τα αποθηκεύσει, ξέρει τι αλλαγές πρέπει να κάνει στη βάση δεδομένων και δημιουργεί αυτόματα τα CRUD ερωτήματα που πρέπει να εκτελεστούν.
- Μη παρεμβατικό persistence: Ένα καλό ORM εργαλείο δεν θέτει περιορισμούς όσον αφορά τα αντικείμενα του business model της εφαρμογής. Δηλαδή τα αντικείμενα δεν πρέπει να χρειάζεται να κληρονομήσουν κάποια συγκεκριμένη κλάση ή interface.
- Υποστήριξη πολλών RDBMS όπως Oracle, DB2, Microsoft SQL Server, PostgreSQL, MySQL κ.α.

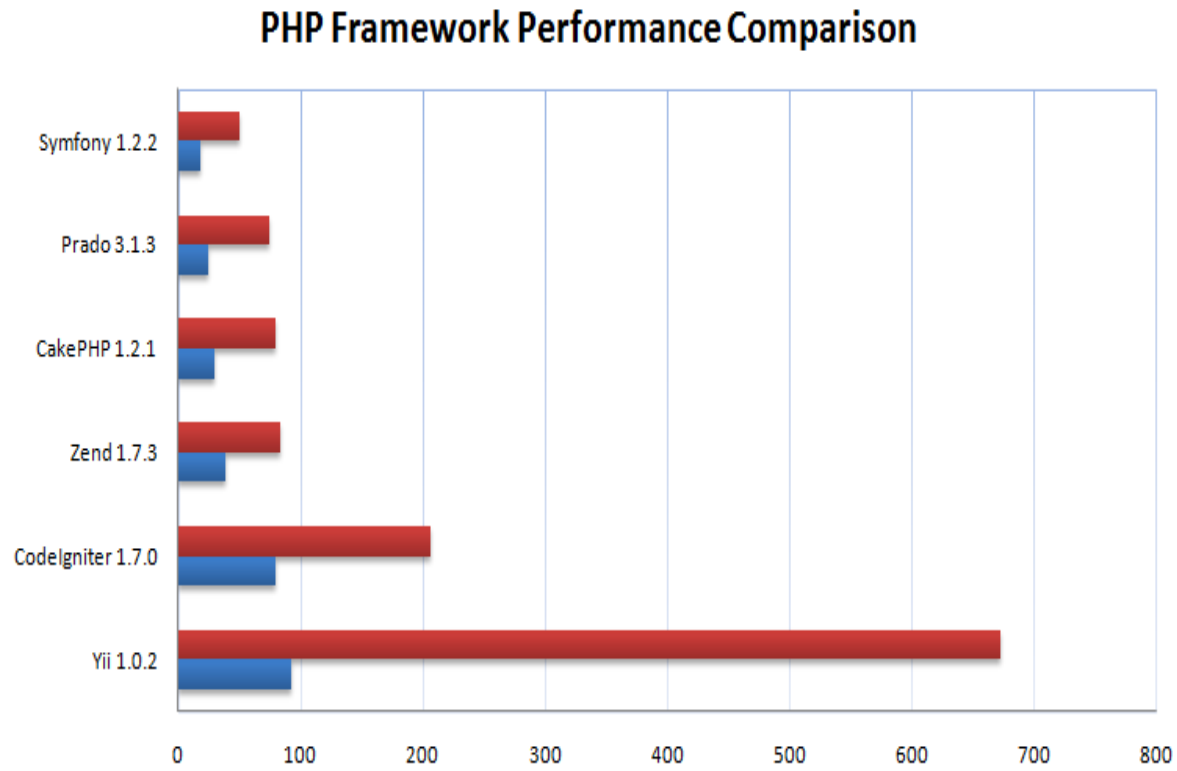
Βέβαια πρέπει να τονίσουμε ότι τα ORM εργαλεία δεν είναι πάντα η καλύτερη λύση για την υλοποίηση του DAL επιπέδου μια εφαρμογής. Για παράδειγμα, για μικρές εφαρμογές, η διαδικασία της ρύθμισης της σωστής απεικόνισης των αντικειμένων σε πίνακες καθώς και η διαδικασία της βελτίωσης της αποδοτικότητας των SQL ερωτημάτων που παράγονται αυτόματα, ίσως είναι περισσότερο χρονοβόρες από την χρησιμοποίηση απλούστερων τεχνικών όπως SQL ερωτημάτων. Από την άλλη, για μεγάλες business εφαρμογές, πρέπει να ζυγιστούν κατάλληλα τα πλεονεκτήματα των ORM εργαλείων που αναφέραμε παραπάνω.

2.4 PHP framework

Για τη σχεδίαση της εφαρμογής με το αρχιτεκτονικό πρότυπο Model – View – Controller χρησιμοποιήθηκε το PHP framework, το οποίο διευκολύνει την ανάπτυξη εφαρμογών σε PHP ακολουθώντας το συγκεκριμένο αρχιτεκτονικό πρότυπο. Η επιλογή του συγκεκριμένου framework βασίστηκε στην πληθώρα δυνατοτήτων, οι οποίες παρέχονται από αυτό. Οι ενότητες που ακολουθούν περιγράφουν τις δυνατότητες και τα χαρακτηριστικά του PHP framework. Τα διαγράμματα που παρατίθενται στη συνέχεια παρουσιάζουν τα στρώματα στα οποία οργανώνεται ο κώδικας του PHP framework. Στο πρώτο σχήμα φαίνεται η βάση, η οποία περιλαμβάνει όλες τις απαραίτητες λειτουργίες για τον εύρυθμο συντονισμό των υπολοίπων τμημάτων του framework. Πάνω από τη βάση βρίσκονται μία σειρά από υποσυστήματα, εκ των οποίων το υποσύστημα της διαχείρισης βάσεων δεδομένων και το υποσύστημα λειτουργιών σχετικών με το Web αναλύονται περαιτέρω σε ξεχωριστά διαγράμματα που ακολουθούν.

2.4.1 Ταχύτητα

Το PHP framework παρέχει αξιόλογη ταχύτητα στην εκτέλεση της εφαρμογής μέσω της τεχνικής της οκνηρής φόρτωσης των συστατικών του. Αυτό σημαίνει ότι φορτώνει τα απαραίτητα συστατικά για την εκτέλεση της εφαρμογής καθώς αυτά απαιτούνται σε αντίθεση με τα υπόλοιπα PHP frameworks, τα οποία για λόγους ευκολίας φορτώνουν στη μνήμη όλα τα συστατικά πριν την εξυπηρέτηση του κάθε HTTP αιτήματος προσθέτοντας με αυτόν τον τρόπο μία αναπόφευκτη χρονική καθυστέρηση στην HTTP απάντηση.Στο σχήμα που ακολουθεί παρουσιάζεται μία συγκριτική αναπαράσταση της ταχύτητας ορισμένων δημοφιλών PHP frameworks. Η μονάδα μέτρησης είναι ο αριθμός αιτημάτων ανά δευτερόλεπτο (RPS) που μπορεί να εξυπηρετήσει το κάθε framework. Επιπλέον, λαμβάνεται υπόψη και η συνεισφορά του Alternative PHP Cache (APC), το οποίο είναι ένα σύστημα βελτιστοποίησης του ενδιάμεσου κώδικα της PHP, το οποίο αποθηκεύει στη μνήμη του διακομιστή το μεταγλωττισμένο κώδικα και τα απαραίτητα δεδομένα έτσι ώστε να επιταχύνεται η εκτέλεση των εφαρμογών PHP. Για κάθε PHP framework η άνω ράβδος δηλώνει τον αριθμό αιτημάτων ανά δευτερόλεπτο με χρήση του APC και η κάτω ράβδος δηλώνει το RPS δίχως τη χρήση APC.



	Yii 1.0.2	CodeIgniter 1.7.0	Zend 1.7.3	CakePHP 1.2.1	Prado 3.1.3	Symfony 1.2.2
RPS with APC	673	206	83	79	75	50
RPS without APC	93	79	38	29	24	18

Εικόνα 4: Γραφική παράσταση των επιδόσεων σε αιτήματα ανά δευτερόλεπτο (RPS) έξι διαδεδομένων PHP frameworks.

2.4.2 Ποιότητα κώδικα

Το PHP framework είναι γραμμένο εξολοκλήρου αξιοποιώντας τις αντικειμενοστρεφείς δυνατότητες της PHP με αποτέλεσμα να αποτελείται από ευανάγνωστο κώδικα, ο οποίος είναι εύκολος στη συντήρηση. Με αυτό τον τρόπο υπάρχει η εγγύηση ότι τυχόν προβλήματα που ανακαλύπτονται στο framework είναι εύκολο να επιλυθούν σε σύντομο χρονικό διάστημα. Η χρήση των αντικειμενοστραφών χαρακτηριστικών της PHP συνεπάγεται την ανάγκη χρήσης της έκδοσης 5 ή μεταγενέστερης αυτής, από την οποία και έπειτα υποστηρίζονται πλήρως τα συγκεκριμένα χαρακτηριστικά.

Τι είναι web framework. Συνήθως οι προγραμματιστές γνωρίζουν κάποιες γλώσσες προγραμματισμού με τις οποίες μπορεί να δημιουργήσουν διάφορες εφαρμογές είτε από το μηδέν είτε χρησιμοποιώντας διάφορες βιβλιοθήκες που τους προσφέρουν έτοιμες κάποιες λειτουργίες. Ένα framework προσφέρει και αυτό κάποιες έτοιμες λειτουργίες όπως caching, templating, object mapping αλλά και μπορεί να επεκταθεί με
plugins αλλά κατά κύριο λόγο προτυποποιεί κάποιες διαδικασίες όπως η αρχιτεκτονική της εφαρμογής, η σύνδεση στη βάση δεδομένων αλλά μπορεί και να δίνει δυνατότητα για γρήγορη και αυτόματη δημιουργία κώδικα όπως το scaffolding.

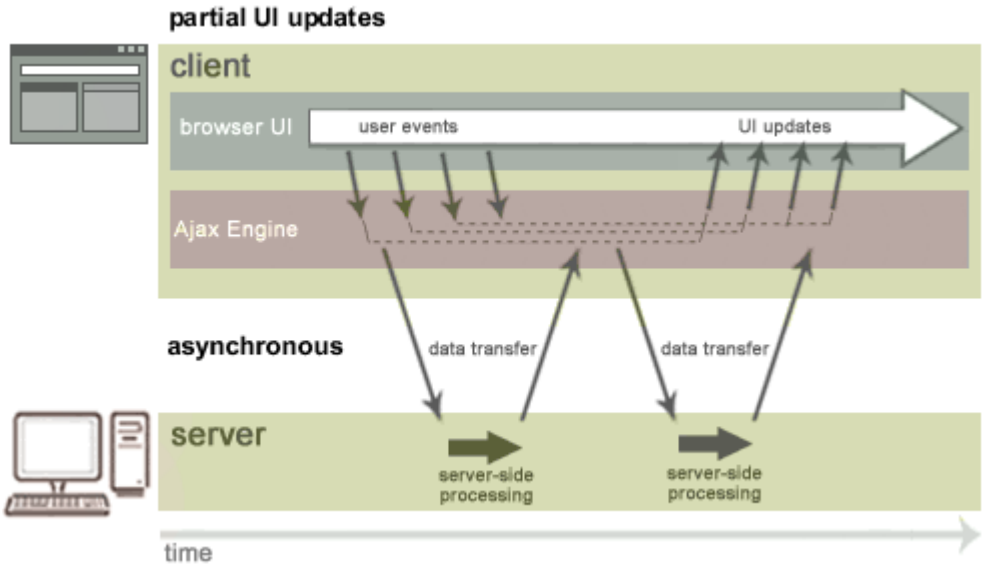
Η PHP είναι μια γλώσσα προγραμματισμού που σχεδιάστηκε για τη δημιουργία δυναμικών σελίδων στο δυαδίκτυο και είναι επισήμως γνωστή ως: HyperText preprocessor. Είναι μια server-side (εκτελείτε στον διακομιστή) scripting γλώσσα που γράφεται συνήθως 13 πλαισιωμένη από HTML, για μορφοποίηση των αποτελεσμάτων. Αντίθετα από μια συνηθισμένη HTML σελίδα η σελίδα PHP δεν στέλνεται άμεσα σε έναν πελάτη (client), αντ' αυτού πρώτα αναλύεται και μετά αποστέλλεται το παραγόμενο αποτέλεσμα. Τα στοιχεία HTML στον πηγαίο κώδικα μένουν ως έχουν, αλλά ο PHP κώδικας ερμηνεύεται και εκτελείται. Ο κώδικας PHP μπορεί να θέσει ερωτήματα σε βάσεις δεδομένων, να δημιουργήσει εικόνες, να διαβάσει και να γράψει αρχεία, να συνδεθεί με απομακρυσμένους υπολογιστές , κ.ο.κ. Σε γενικές γραμμές οι δυνατότητες που μας δίνει είναι απεριόριστες.

Η JavaScript είναι γλώσσα προγραμματισμού η οποίαέχει σαν σκοπό την παραγωγή δυναμικού περιεχομένου και την εκτέλεση κώδικα στην πλευρά του πελάτη (client-side) σε ιστοσελίδες. Το πρότυπο της γλώσσας κατά τον οργανισμό τυποποίησης ECMA ονομάζεται ECMAscript.

2.5 Η τεχνολογία AJAX:

2.5.1 Γενικά

Παρά την τρομερή τεχνολογική ανάπτυξη του διαδικτύου την περασμένη δεκαετία η ευχρηστία των εφαρμογών του παγκόσμιου ιστού υστερεί σε σύγκριση με αυτή των εφαρμογών που εκτελούνται στο περιβάλλον του κάθε υπολογιστή, τις λεγόμενες και ως εφαρμογές επιφάνειας εργασίας. Κάθε σημαντική αλληλεπίδραση σε μια εφαρμογή παγκόσμιου ιστού συνεπάγεται ένα διάστημα αναμονής για την απόκρισή της, μέχρι αυτή να επικοινωνήσει μέσω διαδικτύου με τον διακομιστή που την εξυπηρετεί. Οι Διαδικτυακές Εφαρμογές Εμπλουτισμένων μέσων είναι εφαρμογές του παγκόσμιου ιστού οι οποίες προσεγγίζουν την όψη και αίσθηση των εφαρμογών επιφάνειας εργασίας. Οι εμπλουτισμένες διαδικτυακές εφαρμογές έχουν δύο βασικά χαρακτηριστικά, την αποδοτικότητα και τη διαλογική γραφική διεπαφή χρήστη.



Εικόνα 5 :Κλασική εφαρμογή παγκόσμιου ιστού με επαναφόρτιση της σελίδας για κάθε αλληλεπίδραση του χρήση .

Η αποδοτικότητα των εμπλουτισμένων διαδικτυακών εφαρμογών προέρχεται από την τεχνολογία AJAX η οποία χρησιμοποιεί προγράμματα σεναρίων στην πλευρά του πελάτη για να δώσει στις εφαρμογές παγκόσμιου ιστού μικρότερους χρόνους απόκρισης. Οι AJAX εφαρμογές διαχωρίζουν την αλληλεπίδραση της πλευράς του πελάτη από την επικοινωνία με τις εκτελούν παράλληλα, μειώνοντας τις καθυστερήσεις της επεξεργασίας στην πλευρά του διακομιστή που υπομένει ο χρήστης. Η λειτουργία των παραδοσιακών και των ασύγχρονων εφαρμογών παγκόσμιου ιστού παρουσιάζεται σχηματικά στην Εικόνα? και την Εικόνα αντίστοιχα. Υπάρχουν δύο κατευθύνσεις για την εφαρμογή λειτουργικότητας AJAX. Η πρώτη είναι η χρήση "ακατέργαστου" AJAX. Χρησιμοποιείται η γλώσσα σεναρίων JavaScript για την αποστολή ασύγχρονων αιτημάτων προς το διακομιστή μιας εφαρμογής με απευθείας χρήση του αντικειμένου της XMLHttpRequest (XHR για συντομία). Κατόπιν μέσω της ιεραρχίας αντικειμένων του DOM, ενημερώνεται το περιεχόμενο. Η δεύτερη αφορά τη χρήση έτοιμων βιβλιοθηκών, ονομαζόμενων και ως πλαίσια ανάπτυξης εφαρμογών, όπως τα Prototype, Script.aculo.us, Sarissa, κ.α., πληροφορίες για τα οποία δίνονται στη μεθεπόμενη παράγραφο.

2.5.2 Ταυτότητα

Ο όρος AJAX επινοήθηκε το 2005 από τον Jesse James Garrett ο οποίος στο άρθρο του "Ajax: A New Approach to Web Applications" αναφέρει ότι η AJAX δεν είναι μια τεχνολογία άλλα διάφορες, κάθε μια με τη δική της λάμψη. Σκοπός της είναι η ασύγχρονη φόρτωση περιεχομένου των ιστοσελίδων στο φυλλομετρητή. Η λέξη AJAX αποτελεί αρκτικόλεξο των όρων Asynchronous JavaScript And XML (Ασύγχρονη JavaScript και XML). Η τεχνολογία AJAX χρησιμοποιεί τις ακόλουθες για να πετύχει το σκοπό της:

- Τις γλώσσες, επισήμανσης ιστοσελίδων XHTML και μορφοποίησης CSS, για την παρουσίαση του περιεχομένου.
- Το μοντέλο οντοτήτων DOM για τη δυναμική προβολή και την αλληλεπίδραση στις ιστοσελίδες.
- Τις γλώσσες, επισήμανσης XML και μετασχηματισμών XSLT, για την ανταλλαγή και τη διαχείριση δεδομένων αντίστοιχα.
- Το αντικείμενο XMLHttpRequest της JavaScript, για την ασύγχρονη ανάκτηση δεδομένων.
- Τη γλώσσα σεναρίων JavaScript για τη διασύνδεση και διαλειτουργικότητα των προηγούμενων τεχνολογιών.
- Βέβαια από τότε που πρωτοεμφανίστηκε ο όρος AJAX έχουν προοδεύσει οι διάφορες τεχνολογίες του παγκόσμιου ιστού με αποτέλεσμα:
- Η JavaScript να μην είναι πλέον η μόνη γλώσσα υλοποίησης AJAX λύσεων. Μπορούν να χρησιμοποιηθούν και άλλες γλώσσες σεναρίων όπως η VBScript.
- Το αντικείμενο XMLHttpRequest δεν χρησιμοποιείται αποκλειστικά για ασύγχρονη επικοινωνία. Ο μηχανισμός των ενσωματωμένων πλαισίων iframe που πρωτοεμφανίστηκε το 1996 στον Microsoft Internet Explorer μπορεί να χρησιμοποιηθεί για το ίδιο αποτέλεσμα.
- Οι γλώσσες XML και XSLT δεν είναι οι μόνες που μπορούν να χρησιμοποιηθούν για την ανταλλαγή και διαχείριση δεδομένων. Μπορούν εναλλακτικά να χρησιμοποιηθούν και άλλες μορφές ανταλλαγής δεδομένων, όπως η JSON, η γλώσσα επισήμανσης HTML, ακόμη και το απλό κείμενο.

2.5.3 Βιβλιοθήκες

Η σύγχρονη τάση στην ανάπτυξη αλληλεπιδραστικών εφαρμογών AJAX για τον παγκόσμιο ιστό είναι η χρήση έτοιμων βιβλιοθηκών JavaScript - ή πλαισίων ανάπτυξης κώδικα JavaScript (framework) όπως συχνά

καλούνται. Με τον τρόπο αυτό επιτυγχάνεται η ευκολία υλοποίησης δύσκολων τεχνικών και η σταθερότητα και συμβατότητα με τις διαφορετικές πλατφόρμες λογισμικού των φυλλομετρητών. Οι βιβλιοθήκες αυτές είναι εκατοντάδες.

2.5.4 Το AJAX στο symfony

Σε γενικές γραμμές η χρήση της τεχνολογίας AJAX στο Moodle είναι έως σήμερα mia apo ths texnologies poy dieykolinei ton xrhsth gia na anpatyx8y I eformogh toy gry. Μόνο δύο είναι τα υποέργα του που χρησιμοποιούν AJAX. Το πρώτο αφορά τη χρήση AJAX στις σελίδες που αφορούν μαθήματα και το δεύτερο είναι ένα δυναμικό θέμα εμφάνισης του περιβάλλοντος εργασίας του Moodle με όνομα chameleon. Το chameleon είναι ένα αλληλεπιδραστικό θέμα στο οποίο μπορεί κανείς να επιλέξει διαφορετικά χρώματα και ρυθμίσεις γραμματοσειράς, περιγραμμάτων, κ.ο.κ. Οι διαχειριστές του Moodle έχουν επιλέξει από τον Ιούλιο του 2006 τη βιβλιοθήκη Yahoo! User Interface ως το θεμέλιο ανάπτυξης κώδικα JavaScript και υλοποίησης AJAX λύσεων. Omow simera exei aneptyx8i se poli grugorous ri8moiys .

2.6 Τι είναι το Drupal:

Το Drupal είναι ένα πρόγραμμα σύγχρονου συστήματος διαχείρισης περιεχομένου (Content Management System, CMS) ανοικτού/ελεύθερου λογισμικού. Ποιό συγκεκριμένα χρησιμοποιήθηκαν οι γλώσσες προγραμματισμού ιστοσελίδων PHP,Javascript και CSS. Το Drupal, όπως πολλά σύγχρονα CMS, επιτρέπει στο διαχειριστή συστήματος να οργανώνει το περιεχόμενο, να προσαρμόζει την παρουσίαση, να αυτοματοποιεί διαχειριστικές εργασίες και να διαχειρίζεται τους επισκέπτες του ιστοτόπου και αυτούς που συνεισφέρουν. Παρόλο που υπάρχει μια πολύπλοκη προγραμματιστική διεπαφή, οι περισσότερες εργασίες μπορούν να γίνουν με λίγο ή και καθόλου προγραμματισμό

2.6.1 Modules (Ενότητες):

Η λειτουργίες του Drupal περιέχονται σε ‘’modules’’, τα οποία μπορούν να ενεργοποιηθούν ή να απενεργοποιηθούν. Τα Modules είναι διάφορα κομμάτια κώδικα τα οποία εκτελούν μια συγκεκριμένη εργασία και μπορούμε να έχουμε όσες θέλουμε σε μια ιστοσελίδα. Τα modules είναι ελεύθερα να τα κατεβάσει κάποιος και απαιτούν κάποια διαμόρφωση, αλλά είναι άριστα εργαλεία σχεδιασμένα για να μας βοηθήσουν να επιτύχουμε τα αποτελέσματα που θέλουμε από το δικτυακό μας τόπο με χρήση του Drupal.

2.6.2 Themes (Θεματικές παραλλαγές):

Το Drupal, ως ένα ορθολογισμένο cms που είναι, διαχωρίζει απόλυτα τα δεδομένα από την εμφάνισή τους. Για τον τρόπο εμφάνισης της σελίδας, των εικόνων, των χρωμάτων, των γραμματοσειρών κλπ, είναι υπεύθυνες οι θεματικές παραλλαγές. Ο πυρήνας του drupal εγκαθιστά και 6 βασικές θεματικές παραλλαγές οι οποίες βρίσκονται στον φάκελο "themes" της εγκατάστασης μας. Εκτός από τις 6 αυτές παραλλαγές, μπορούμε να κατεβάσουμε και να χρησιμοποιήσουμε δεκάδες άλλες που υπάρχουν στο κύριο αποθευτήριο του Drupal.

2.6.3 CMS:

Το Συστήματα Διαχείρισης Περιεχομένου (Content Management Systems, CMS) είναι το σύστημα των μεθόδων και τεχνικών που αυτοματοποιουν τις διαδικασίες της συλλογής, διαχειρισης και δημοσιευσης περιεχομενου με χρηση τεχνολογιών πληροφορικής.

2.6.4 XAAMP

Το όνομα του Xampp είναι ένα ακρωνύμιο των:
X(σημαίνει cross-platform=που λειτουργεί σε πολλές πλατφόρμες)
Apache HTTP Server
MySQL
PHP
Perl

Το XAMPP δρα σαν ένας ελεύθερος web server ικανός να εξυπηρετήσει δυναμικές ιστοσελίδες. Είναι διαθέσιμο για Microsoft Windows, Linux, Solaris και Mac OS X,και χρησιμοποιείται κυρίως για web development projects.Το λογισμικό αυτό είναι χρήσιμο όταν δημιουργούμε δυναμικές ιστοσελίδες με jsp και servlet. Μπορούμε να το βρούμε σε μορφή zip tar ή exe για να το κατεβάσουμε και να το εγκαταστήσουμε. Αναβαθμίζεται συχνά για να υποστηρίξει τις τελευταίες εκδόσεις των Apache/MySQL/PHP και Perl.
Η εγκατάσταση του XAMPP παίρνει λιγότερο χρόνο από ότι θα έπαιρνε αν εγκαταστήσουμε κάθε ένα από τα πακέτα ξεχωριστά. Είναι ανεξάρτητο και μπορεί να λειτουργήσουν πολλές εγκαταστάσεις του στον ίδιο υπολογιστή. Αρχικά οι δημιουργοί του XAMPP το σχεδίασαν σαν εργαλείο ανάπτυξης για web designers και προγραμματιστές να ελέγξουν τη δουλειά τους στο δικό τους PC χωρίς να χρειάζεται να έχουν πρόσβαση στο Internet.

2.6.5 Αρχιτεκτονική

Η εφαρμογή που αναπτύχθηκε έχει σχεδιαστεί έχοντας ως βάση την μέγιστη δυνατή λειτουργικότητα. Έτσι έχει προσεχθεί ώστε να είναι:

- Ανεξάρτητη λειτουργικού συστήματος (δηλαδή μπορεί να εγκατασταθεί σε οποιοδήποτε λειτουργικό σύστημα),
- Προσπελάσιμη μέσω οποιουδήποτε φυλλομετρητή (browser),
- πλήρως σπονδυλωτή στη δομή της, δηλαδή να χρησιμοποιεί αρθρώματα (modules) για τις διάφορες λειτουργίες που εκτελεί
- Προσιτή στη διαχείριση από τον διδάσκοντα.

Οι παραπάνω ιδιότητες-στόχοι οδηγούν σε ένα σύστημα υλικού (H/W) και λογισμικού (S/W) το οποίο αποτελείται από:

- Τη βάση δεδομένων που περιέχει όλες τις απαραίτητες πληροφορίες για τη λειτουργία του συστήματος,
- Τα αποθηκευτικά μέσα, δηλαδή τους υπολογιστές που φιλοξενούν το υλικό του μαθήματος,
- Το περιβάλλον εργασίας, δηλαδή το λογισμικό διεπαφής που επεξεργάζεται τις πληροφορίες και κάνει δυνατή την αλληλεπίδραση των χρηστών με το εκπαιδευτικό υλικό. Η εφαρμογή είναι βασισμένη στο πρότυπο τύπου «πελάτη-εξυπηρετητή» (client-server). Εγκαθίσταται σε οποιοδήποτε λειτουργικό σύστημα, υποστηρίζει web server τύπου Apache ή Microsoft IIS, ενώ στηρίζεται εξ' ολοκλήρου σε περιβάλλοντα «ανοιχτού κώδικα» (open source) για την λειτουργία της. Για την ανάπτυξη του ιστογενούς περιβάλλοντος της εφαρμογής και των αλγορίθμων της χρησιμοποιήθηκε η γλώσσα PHP (Pre Hypertext Processor) .

2.6.6 Απαιτήσεις και χαρακτηριστικά

Το XAMPP απαιτεί μόνο ένα zip, tar, ή exe αρχείο για να κατέβει και να τρέξει, και απαιτείται μία μικρή διάρθρωση των επιμέρους στοιχείων που συνθέτουν τον web server. Το XAMPP ενημερώνεται τακτικά και συμπεριλαμβάνει τις τελευταίες εκδόσεις του Apache / MySQL / PHP και Perl. Επίσης, έρχεται με μια σειρά από άλλες μονάδες, συμπεριλαμβανομένων των OpenSSL και phpMyAdmin.

2.7 Apache

Ο Apache HTTP server, συχνά αναφερόμενος απλά σαν Apache, είναι ένας web server ο οποίος διαδραμάτισε καίριο ρόλο στην αρχική ανάπτυξη του παγκόσμιου ιστού. Το 2009 έγινε ο web server που ξεπέρασε το όριο των εκατό εκατομμυρίων σελίδων στο διαδίκτυο. Ο Apache ήταν η πρώτη βιώσιμη εναλλακτική λύση απέναντι στον Netscape Corporation web server (γνωστό σήμερα ως Sun Java System web server), και από τότε εξελίχθηκε σε υπολογίσιμο αντίπαλο άλλων web server που βασίζονται σε Unix όσον αφορά την λειτουργικότητα και τις επιδόσεις.

Ο Apache αναπτύσσεται και συντηρείται από μια ανοικτή κοινότητα προγραμματιστών υπό την αιγίδα του Apache Software Foundation. Η εφαρμογή είναι διαθέσιμη για μια μεγάλη ποικιλία λειτουργικών συστημάτων στα οποία περιλαμβάνονται τα Unix, GNU, FreeBSD, Linux, Solaris, Novell NetWare, Mac OS X, Microsoft Windows, OS/2, TPF και eComStation. Ο Apache χαρακτηρίζεται ως ένα λογισμικό ανοικτού κώδικα. Από τον Απρίλιο του 1996 και μετά, ο Apache είναι ο πιο δημοφιλής HTTP server του διαδικτύου. Επίσης μετά από μέτρηση που πραγματοποιήθηκε τον Αύγουστο του 2009, ο Apache εξυπηρετεί το 54,32% όλων των σελίδων του διαδικτύου και το 66% από τις 1.000.000 πιο δημοφιλείς.

Η πρώτη έκδοση του Apache δημιουργήθηκε από τον Robert McCool, ο οποίος συμμετείχε στην ανάπτυξη του National Center of Supercomputing Applications web server, γνωστό απλά ως NCSA HTTPd. Όταν ο McCool έφυγε από την NCSA στα μέσα του 1994, η ανάπτυξη του HTTPd σταμάτησε, αφήνοντας μια ποικιλία από προσθήκες για βελτιώσεις να κυκλοφορεί μέσω email. Αυτές τις προσθήκες παρείχε ένας αριθμός προγραμματιστών οι οποίοι βοήθησαν να δημιουργηθεί η αρχική ομάδα ανάπτυξης του Apache γνωστή και ως "Apache Group".

Υπάρχουν δύο επεξηγήσεις όσον αφορά το όνομα του project. Σύμφωνα με το Apache Foundation, το όνομα επελέγη από σεβασμό στην φυλή των αυτοχθόνων Αμερικανών Apache οι οποίοι ήταν γνωστοί για την αντοχή και τις ικανότητες τους στην μάχη. Παρ' όλα αυτά, κατά την περίοδο 1996-2001, η επεξήγηση που έδινε η ιστοσελίδα του Apache project ήταν ότι επειδή πρόκειται για έναν server ο οποίος δημιουργήθηκε βασισμένος σε προσθήκες (patches), ονομάστηκε patchy server και με τον καιρό κατέληξε να αποκαλείται Apache.

Ο Apache υποστηρίζει μία πολύ μεγάλη ποικιλία χαρακτηριστικών και δυνατοτήτων. Πολλά από αυτά προσαρτώνται στον πυρήνα με την μορφή modules επεκτείνοντας τις δυνατότητες του. Αυτά περιλαμβάνουν από υποστήριξη server-side γλωσσών προγραμματισμού έως και αλγόριθμους αυθεντικοποίησης. Κάποιες από τις δημοφιλείς γλώσσες που υποστηρίζονται είναι οι Perl, Python, Tcl και PHP. Κάποια από τα δημοφιλή modules αυθεντικοποίησης που υποστηρίζονται είναι τα mod_access, mod_auth, mod_digest και mod_auth_digest. Κάποιες από τις άλλες δυνατότητες περιλαμβάνουν υποστήριξη των πρωτοκόλλων SSL και TLS(mod_ssl), ένα proxy module, ένα URL rewriter (mod_rewrite), παραμετροποιημένες καταγραφές συμβάντων (mod_log_config) καθώς και υποστήριξη φίλτρων (mod_ext_filter).

Μια δημοφιλείς μέθοδος συμπίεσης πού χρησιμοποιείται στον Apache είναι το external extension module (mod_gzip) το οποίο βοήθα στον να μειωθεί το μέγεθος των ιστοσελίδων που εξυπηρετούνται μέσω HTTP. Επίσης δημοφιλές είναι και το ModSecurity το οποίο είναι μια μηχανή ανοιχτού κώδικα που εντοπίζει και εμποδίζει εισβολές σε διαδικτυακές εφαρμογές. Το ιστορικό του Apache μπορεί να διαχειριστεί μέσω ενός web browser χρησιμοποιώντας ελεύθερες εφαρμογές όπως AWStats/W3Perl ή το Visitors. Κάποια επιπλέον χαρακτηριστικά του Apache είναι το Virtual Hosting, που επιτρέπει σε πολλές διαφορετικές ιστοσελίδες να εξυπηρετούνται από μία μόνο εγκατάσταση του server, παραμετροποιήσιμα μηνύματα σφάλματος, Βάσεις δεδομένων βασισμένες σε αυθεντικοποίηση DBMS, διαχείριση περιεχομένου και υποστήριξη διαφόρων

GUIs(Graphical User Interfaces). Ο Apache χρησιμοποιείται κυρίως για την εξυπηρέτηση στατικών και δυναμικών σελίδων στο διαδίκτυο. Πολλές διαδικτυακές εφαρμογές σχεδιάζονται με βάση το περιβάλλον και τα χαρακτηριστικά που προσφέρει ο Apache. Ο συγκεκριμένος server αποτελεί κομμάτι της δημοφιλούς ομάδας εφαρμογών LAMP την οποία αποτελούν ο Apache, το λειτουργικό Linux, το σύστημα διαχείρισης βάσεων δεδομένων MySQL και οι γλώσσες προγραμματισμού PHP/Perl/Python. Ο Apache αποτελεί βασικό κομμάτι πολλών πακέτων εφαρμογών όπως: Oracle Database, IBM WebSphere application server, WebObject application server, Mac OS X, Novell NetWare6.5 καθώς και σε πολλές διανομές του λειτουργικού συστήματος Linux.

2.8 MySQL

Η MySQL είναι ένα σύστημα διαχείρισης σχεσιακών βάσεων δεδομένων (RDBMS) το οποίο μετρά περισσότερες από 10 εκατομμύρια εγκαταστάσεις. Έλαβε το όνομά του από την κόρη του Μόντυ Βιντένιους, την Μάι. Το πρόγραμμα τρέχει έναν εξυπηρετητή (server) παρέχοντας πρόσβαση πολλών χρηστών σε ένα σύνολο βάσεων δεδομένων. Η βάση δεδομένων MySQL έχει γίνει η πιο δημοφιλής

βάση δεδομένων ανοιχτού λογισμικού εξαιτίας της σταθερά υψηλής απόδοσής της, της αξιοπιστίας της και της ευκολίας της χρήσης της. Χρησιμοποιείται παγκοσμίως τόσο από μεμονωμένους δημιουργούς διαδικτυακών χώρων όσο και από πολλούς από τους μεγαλύτερους και τους πιο ραγδαία αναπτυσσόμενους οργανισμούς για την εξοικονόμηση χρόνου και χρήματος. Επίσης, χρησιμοποιείται για τη δημιουργία διαδικτυακών χώρων με μεγάλο όγκο δεδομένων, κρίσιμων συστημάτων για τη λειτουργία εταιρικών εφαρμογών και πακέτων λογισμικού μεγάλων εταιρειών. Η MySQL δεν είναι μόνο η πιο δημοφιλής βάση δεδομένων ανοιχτού λογισμικού, αλλά συγχρόνως έχει γίνει και η επιλεγμένη βάση δεδομένων για τη νέα γενιά εφαρμογών που βασίζεται στο LAMP (Linux, Apache, MySQL, PHP/Perl/Python). Η MySQL τρέχει σε περισσότερες από 20 πλατφόρμες συμπεριλαμβανομένων του Linux, των Windows, του OS/X, του HP-UX, του AIX και του Netware, παρέχοντας στο χρήστη όλη την απαιτούμενη ευελιξία. Η έκδοση της MySQL Enterprise Server 5.0 διαθέτει τα ακόλουθα χαρακτηριστικά:

- Συναλλαγές ACID για τη δημιουργία αξιόπιστων και ασφαλών κρίσιμων εταιρικών εφαρμογών.
- Αποθηκευμένες διαδικασίες για τη βελτίωση της παραγωγικότητας των προγραμματιστών.
- Διαδικασίες πυροδότησης για την επιβολή πολύπλοκων επιχειρηματικών κανόνων στο επίπεδο της βάσης δεδομένων.
- Μεθόδους προστασίας των ευαίσθητων δεδομένων.
- Σχήμα Πληροφοριών για την παροχή εύκολης πρόσβασης σε μεταδεδομένα.
- Κατανεμημένες Συναλλαγές για την υποστήριξη πολύπλοκων συναλλαγών εντός πολλαπλών βάσεων δεδομένων.
- Αρχιτεκτονική Μηχανής Αποθήκευσης για την παροχή της μέγιστης δυνατής ευελιξίας.
- Μηχανή Αποθήκευσης Αρχείων για την καταγραφή του ιστορικού και των δεδομένων λογιστικού ελέγχου.
- Ενοποιημένη Μηχανή Αποθήκευσης για τη δημιουργία μίας λογικής βάσης δεδομένων από πολλούς φυσικούς εξυπηρετητές.
- Εκδόσεις με διορθώσεις των πιο σημαντικών σφαλμάτων. Με τη βοήθεια της PHP μπορούμε να συνδεθούμε σε έναν οποιοδήποτε MySQL Server στον οποίο έχουμε λογαριασμό, να πάρουμε δεδομένα από ήδη υπάρχουσες βάσεις, να εισάγουμε δεδομένα σε πίνακες βάσεων, να ανανεώσουμε κάποια υπάρχοντα δεδομένα, να φτιάξουμε νέες βάσεις και νέους πίνακες και γενικά να κάνουμε οτιδήποτε γίνεται με μια MySQL βάση δεδομένων. Επομένως, μέσα από τις Web σελίδες μας μπορούμε να διαχειριστούμε εύκολα μια MySQL βάση δεδομένων και έτσι οι σελίδες μας να αποκτήσουν πολλές άλλες δυνατότητες που απαιτούν οι σύγχρονες απαιτήσεις των χρηστών δηλαδή

να γίνουν δυναμικές, ελκυστικές και ανταγωνιστικές.
Μια τυπική διαδικτυακή συναλλαγή βάσεων δεδομένων αποτελείται από τις παρακάτω φάσεις:

- Ο web browser ενός χρήστη κάνει μια HTTP αίτηση για μια συγκεκριμένη διαδικτυακή σελίδα.
- Ο διαδικτυακός διακομιστής (Apache Server) λαμβάνει την αίτηση για τη σελίδα, ανακαλεί το αρχείο και το περνά στη μηχανή PHP για επεξεργασία.
- Η μηχανή PHP αρχίζει την ανάλυση του script. Μέσα στο script, υπάρχει μια εντολή που συνδέει τη βάση δεδομένων και εκτελεί ένα ερώτημα. Η PHP ανοίγει μια σύνδεση με το MySQL διακομιστή (server) και στέλνει το κατάλληλο ερώτημα.
- Ο MySQL διακομιστής (server) λαμβάνει το ερώτημα της βάσης δεδομένων, το επεξεργάζεται και στέλνει τα αποτελέσματα ξανά στη μηχανή PHP.
- Η μηχανή PHP σταματά την εκτέλεση του script, που συνήθως περιλαμβάνει τη μορφοποίηση των αποτελεσμάτων του ερωτήματος σε HTML. Επιστρέφει μετά την τελική HTML σελίδα στον web διακομιστή (Apache Server).

2.8.1 Πλεονεκτήματα της Mysql

Μερικοί από τους κύριους ανταγωνιστές της Mysql είναι οι PostgreSQL, Microsoft SQL Server, Oracle. Η Mysql όμως διαφέρει γιατί έχει τα εξής πλεονεκτήματα:

- Υψηλή απόδοση
- Χαμηλό κόστος
- Εύκολη διαμόρφωση και εκμάθηση
- Μεταφερσιμότητα
- Διαθεσιμότητα του κώδικα προέλευσης
- Διαθεσιμότητα υποστήριξης
- Ο διαδικτυακός διακομιστής (Apache Server) περνά την HTML σελίδα ξανά στο browser, όπου ο χρήστης μπορεί να δει τα αποτελέσματα που ζήτησε

2.9 PhpMyAdmin

Το phpMyAdmin είναι ένα σύνολο από php scripts με το οποίο διαχειριζόμαστε τις βάσεις δεδομένων που έχουμε μέσο web. Το phpMyAdmin μπορεί να διαχειριστεί ένα ολόκληρο mysq server ή ακόμα και απλές βάσεις δεδομένων όπου ο κάθε χρήστης έχει ένα λογαριασμό και μπορεί να δημιουργήσει και να διαχειριστεί τις δικές του βάσεις δεδομένων. Υποστηρίζει 47 γλώσσες μεταξύ των οποίων και τα Ελληνικά και είναι λογισμικό ανοιχτού κώδικα. Οι δυνατότητες του PhpMyAdmin είναι οι εξής:

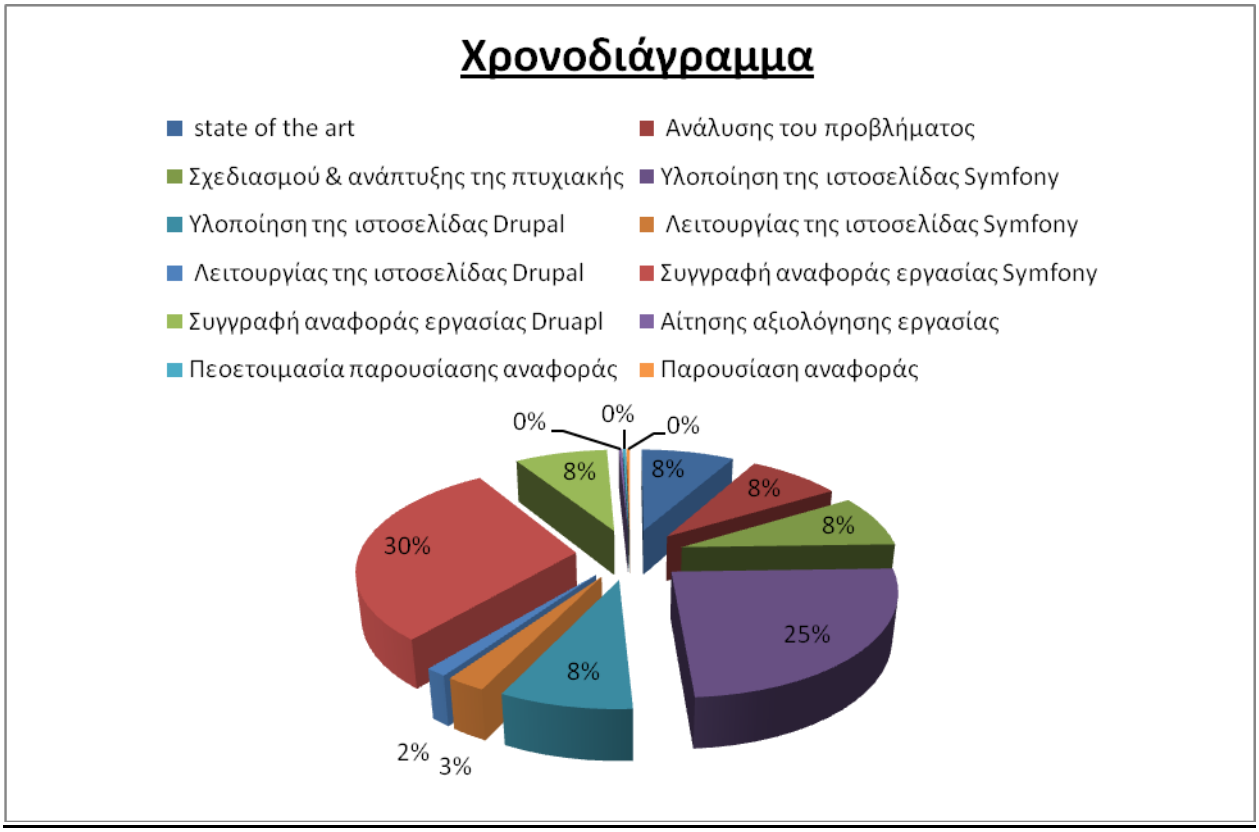
- Δημιουργεί και να διαγράφει βάσεις δεδομένων
- Δημιουργεί, τροποποιεί, διαγράφει, αντιγράφει και μετονομάζει πίνακες
- Κάνει συντήρηση της βάσης
- Προσθέτει, διαγράφει και τροποποιεί πεδία πινάκων
- Εκτελεί Sql ερωτήματα, ακόμα και ομαδικά (batch)
- Διαχειρίζεται κλειδιά σε πεδία
- "Φορτώνει" αρχεία κειμένου σε πίνακες
- Δημιουργεί και διαβάζει πίνακες (που προέρχονται από dump βάσης)
- Εξάγει δεδομένα σε μορφή CVS, Latex, XML
- Διαχειρίζεται πολλούς διακομιστές
- Διαχειρίζεται τους χρήστες MySQL και τα δικαιώματά τους
- Ελέγχει την αναφορική ακεραιότητα των δεδομένων των MyISAM πινάκων

- Δημιουργεί PDF γραφικών του layout της βάσης δεδομένων
- Εκτελεί αναζητήσεις σε όλη τη βάση ή μέρος αυτής
- Υποστηρίζει πίνακες InnoDB και ξένα κλειδιά
- Υποστηρίζει MySQLi, μια βελτιωμένη επέκταση της MySQL

2.10 Σημαντικοί στόχοι για την ολοκλήρωση της πτυχιακής

Χρονοδιάγραμμα	**ΗΜΕΡΕΣ**
state of the art	30
Ανάλυσης του προβλήματος	30
Σχεδιασμού & ανάπτυξης της πτυχιακής	30
Υλοποίηση της ιστοσελίδας Symfony	90
Υλοποίηση της ιστοσελίδας Drupal	30
Λειτουργίας της ιστοσελίδας Symfony	9
Λειτουργίας της ιστοσελίδας Drupal	5
Συγγραφή αναφοράς εργασίας Symfony	110
Συγγραφή αναφοράς εργασίας Druapl	30
Αίτησης αξιολόγησης εργασίας	1
Πεοετοιμασία παρουσίασης αναφοράς	1
Παρουσίαση αναφοράς	1

2.10.1 Χρονοδιάγραμμα



Εικόνα 6 :Το χρονοδιάγραμμα που χρειαστεικε για την υλοποιηση της πτυχιακης

3. Σχέδιο Δράσης για εκπόνηση της πτυχιακής εργασίας

3.1 State of the art

3.1.1 Εισαγωγή στο symfony framework

Το Symfony σε συντομία

Το 2009, σχεδόν την ίδια χρονική περίοδο, κυκλοφόρησαν δύο διαφορετικές εκδόσεις του symfony, η έκδοση 1.3 και η έκδοση 1.4. Και οι δύο εκδόσεις έχουν τα ίδια χαρακτηριστικά μόνο που η έκδοση 1.3 μπορεί να υποστηρίξει projects παλαιοτέρων εκδόσεων ενώ η 1.4 όχι.

Ένα framework βελτιώνει την ανάπτυξη των εφαρμογών αυτοματοποιώντας πολλά από τα πρότυπα που χρησιμοποιούνται για ένα συγκεκριμένο σκοπό. Ένα framework προσθέτει επίσης δομή στον κώδικα, προτρέποντας τον προγραμματιστή να γράφει καλύτερο, πιο ευανάγνωστο, και περισσότερο διατηρήσιμο κώδικα. Τελικά, ένα framework καθιστά τον προγραμματισμό ευκολότερο, καθώς συσκευάζει περίπλοκες λειτουργίες μέσα σε απλές δηλώσεις.

Το symfony είναι ένα πλήρες framework που σχεδιάστηκε για να βελτιστοποιεί την ανάπτυξη των web εφαρμογών με αρκετά από βασικά χαρακτηριστικά. Για πρωτάρηδες, διαχωρίζει τους επιχειρησιακούς κανόνες, τη λογική του εξυπηρετητή, και τις παρουσιάσεις μίας web εφαρμογής. Περιέχει πολλά εργαλεία και κλάσεις που στοχεύουν στο να μειώνουν το χρόνο ανάπτυξης μίας περίπλοκης web εφαρμογής. Επίσης, αυτοματοποιεί κοινές διεργασίες έτσι ώστε ο προγραμματιστής να μπορεί να εστιάσει εξολοκλήρου στις ιδιαιτερότητες μιας εφαρμογής. Το τελικό αποτέλεσμα αυτών των πλεονεκτημάτων σημαίνει ότι δεν χρειάζεται ξανά-ανακαλύψουμε τον τροχό κάθε φορά που δημιουργείται μία καινούρια web εφαρμογή.

Το Symfony είναι γραμμένο εξολοκλήρου σε PHP 5. Έχει δοκιμαστεί πλήρως σε διάφορα projects, και ήδη χρησιμοποιείται σε υψηλής ζήτησης e-business ιστοσελίδες. Είναι συμβατό με τις περισσότερες από τις διαθέσιμες μηχανές βάσης δεδομένων, συμπεριλαμβανομένων των MySQL, PostgreSQL, Oracle και Microsoft SQL Server. Τρέχει σε *nix και Windows πλατφόρμες.

3.1.2 Χαρακτηριστικά του Symfony

Το symfony δημιουργήθηκε με σκοπό να εκπληρώσει τις παρακάτω απαιτήσεις:

- Ευκολία στο να το εγκαταστήσουμε και να το ρυθμίσουμε στις περισσότερες πλατφόρμες
- Ανεξάρτητη μηχανή βάσης δεδομένων
- Εύκολο στη χρήση, στις περισσότερες περιπτώσεις, αλλά ακόμα ευλύγιστο αρκετά για να προσαρμοστεί με περίπλοκα ζητήματα.
- Βασισμένο στην παραδοχή της σύμβασης για τη διαμόρφωση – ο προγραμματιστής χρειάζεται να ρυθμίσει μόνο τα αντισυμβατικά
- Συμβατό με τις περισσότερες καλές πρακτικές web και πρότυπα σχεδίων
- Έτοιμο για επιχειρήσεις – προσαρμόζεται στις υπάρχουσες πολιτικές και αρχιτεκτονικές της τεχνολογίας πληροφοριών, και είναι αρκετά σταθερό για μακροπρόθεσμα projects
- Πολύ ευανάγνωστος κώδικας, με σχόλια phpDocumentor, για εύκολη συντήρηση
- Εύκολο στο να επεκταθεί, επιτρέποντας την ένταξη με άλλες βιβλιοθήκες κατασκευαστών.

Εδώ θα περιγράψουμε τη δημιουργία μιας εφαρμογής για το διαδίκτυο με χρήση του symfony framework, βήμα-βήμα, από τις προδιαγραφές μέχρι και την εκτέλεση. Σκοπός μας είναι να δείξουμε ότι το symfony μπορεί να χρησιμοποιηθεί για να αναπτύξουμε επαγγελματικές εφαρμογές με στυλ και με λίγη προσπάθεια. Με αυτόν τον τρόπο θα μάθουμε να δημιουργούμε πραγματικές ιστοσελίδες από την αρχή μέχρι το τέλος κάνοντας χρήση του symfony.

3.2 Προαπαιτήσεις συστήματος.[1]

Πριν εγκαταστήσουμε το symfony θα πρέπει να ελέγξουμε ότι το λογισμικό που προαπαιτείται για την σωστή λειτουργία του, είναι εγκατεστημένο στον υπολογιστή και έχει ρυθμιστεί ανάλογα. Θα πρέπει να υπάρχει φιλικό εργασιακό περιβάλλον για web development. Επίσης χρειάζεται να είναι εγκατεστημένα, ένας web server (π.χ. Apache), μία μηχανή βάσης δεδομένων, (MySQL, PostgreSQL, SQLite ή οποιαδήποτε συμβατή με PDO[2] μηχανή βάσης δεδομένων), και η PHP 5.2.4 ή μεταγενέστερη έκδοση.

3.3 Ρυθμίσεις παραμέτρων PHP

Καθώς οι ρυθμίσεις των παραμέτρων του PHP μπορούν να διαφέρουν αρκετά από ένα Λειτουργικό Σύστημα σε ένα άλλο, ακόμα και μεταξύ διαφορετικών διανομών Linux, θα πρέπει να ελεγχθεί ότι καλύπτουν τις ελάχιστες απαιτήσεις του symfony.

Πρώτα, πρέπει να σιγουρευτούμε ότι έχουμε τουλάχιστον το PHP 5.2.4 εγκατεστημένο κάνοντας χρήση της ενσωματωμένης συνάρτησης *phpinfo()* ή εκτελώντας την εντολή *php -v* στη γραμμή εντολών. Σε κάποιες ρυθμίσεις παραμέτρων υπάρχει περίπτωση να έχουμε δύο διαφορετικές εκδόσεις PHP εγκατεστημένες: μία για τη γραμμή εντολών, και μία για το web.Έπειτα, κατεβάζουμε το script που ελέγχει τις ρυθμίσεις των παραμέτρων από το παρακάτω URL:

http://sf-to.org/1.4/check.php

Το αποθηκεύουμε στον ριζικό κατάλογο του web, και το εκτελούμε από τη γραμμή εντολών:

$ php check_configuration.php

Αν υπάρξει κάποιο πρόβλημα με τις ρυθμίσεις των παραμέτρων του PHP, η έξοδος της εντολής θα δώσει τρόπους για το τι πρέπει να διορθωθεί και πως. Θα πρέπει πάντως να εκτελεστεί το script του ελέγχου και μέσω browser για να διορθωθούν κι εκεί τυχών σφάλματα.

3.4 Εγκατάσταση του symfony.

Έστω ότι ονομάζουμε το project που θέλουμε να φτιάξουμε Jobeet. Το Jobeet θα είναι ένα site για εύρεση εργασίας. Θα ασχοληθούμε σε βάθος και αναλυτικά για το πως θα το υλοποιήσουμε παρακάτω. Πριν ξεκινήσουμε την εγκατάσταση θα πρέπει να δημιουργήσουμε έναν φάκελο μέσα στον οποίο θα περιέχονται όλα τα αρχεία που συσχετίζονται με το Jobeet:

$ mkdir -p /home/sfprojects/jobeet

1 .Συνιστάται η χρήση Unix-like Λειτουργικού Συστήματος για λόγους ασφαλείας και γρήγορης εκτέλεσης.

$ cd /home/sfprojects/jobeet

Ενώ στα Windows:

c:\> mkdir c:\development\sfprojects\jobeet

c:\> cd c:\development\sfprojects\jobeet

Συνηθίζεται τα αρχεία του symfony framework να εγκαθίστανται στον φάκελο *lib/vendor*. Οπότε πρώτα ας τον δημιουργήσουμε:

$ mkdir -p lib/vendor

Ο πιο εύκολος τρόπος για να εγκαταστήσουμε το symfony είναι να επιλέξουμε το συμπιεσμένο αρχείο (σε μορφή .**tgz** ή .**zip**) με την έκδοση του symfony που επιθυμούμε από το site του symfony, να το τοποθετήσουμε στον φάκελο */lib/vendor* που μόλις δημιουργήσαμε, να το αποσυμπιέσουμε και να μετονομάσουμε τον φάκελο σε symfony:

$ cd lib/vendor

$ tar zxpf symfony-1.4.0.tgz

$ mv symfony-1.4.0 symfony

$ rm symfony-1.4.0.tgz

Στα Windows η αποσυμπίεση του αρχείου μπορεί να επιτευχθεί με τη χρήση του Windows Explorer. Μετά τη μετονομασία του φακέλου σε symfony η δομή του φακέλου θα είναι:

c:\dev\sfprojects\jobeet\lib\vendor\symfony.

3.4.1 Επιβεβαίωση της εγκατάστασης

Για να επαληθεύσουμε ότι μετά την εγκατάσταση όλα λειτουργούν κανονικά χρησιμοποιούμε την γραμμή εντολών του symfony για να εμφανίσουμε την έκδοση του symfony:

$ cd ../..

$ php lib/vendor/symfony/data/bin/symfony -V

και στα Windows:

cd:\> cd ..\..
cd:\> php lib\vendor\symfony\data\bin\symfony -V

3.5 Project Setup

Στο symfony, οι εφαρμογές οι οποίες μοιράζονται το ίδιο μοντέλο δεδομένων ομαδοποιούνται σε projects. Για τα περισσότερα projects, θα υπάρχουν δύο εφαρμογές: μία αρχική και μία τελική.

3.5.1 Δημιουργία Project

Από τον φάκελο *sfprojects/jobeet*, τρέχουμε την διεργασία του symfony *generate:project* για να δημιουργήσουμε το symfony project:

$ php lib/vendor/symfony/data/bin/symfony generate:project jobeet

Στα Windows:

c:\> php lib\vendor\symfony\data\bin\symfony generate:project jobeet

Η εργασία *generate:project* παράγει την προκαθορισμένη δομή των φακέλων και των αρχείων που χρειάζονται για ένα symfony project:

Φάκελος	Περιγραφή
apps/:	Φιλοξενεί όλες τις εφαρμογές του project
cache/:	Τα αρχεία του framework της κρυφής μνήμης
config/:	Τα αρχεία ρύθμισης παραμέτρων του project
lib/:	Τις βιβλιοθήκες και κλάσεις του project
Log/:	Τα αρχεία log του framework
plugins/:	Τα εγκατεστημένα plugins
test/:	Τα αρχεία ελέγχου της μονάδας και της λειτουργίας
web/:	Τον web root φάκελο

Η διεργασία *generate:project* έχει επίσης δημιουργήσει μια συντόμευση του symfony στον ριζικό φάκελο του project έτσι ώστε να μειώσει των αριθμό των χαρακτήρων που πρέπει να γραφτούν όταν τρέχουμε μία διεργασία.

Έτσι, πλέον, αντί να χρησιμοποιούμε το πλήρη μονοπάτι για το πρόγραμμα του symfony θα μπορούμε να χρησιμοποιήσουμε τη συντόμευση.

3.6 Δημιουργία Εφαρμογής

Τώρα, δημιουργούμε την αρχική εφαρμογή τρέχοντας την διεργασία *generate:app*

$ php symfony generate:app frontend

Βασιζόμενη στο όνομα της εφαρμογής που δόθηκε σαν επιχείρημα, η διεργασία *generate:app* δημιουργεί την προκαθορισμένη δομή φακέλων που χρειάζεται για την εφαρμογή στον *apps/frontend/* φάκελο:

Φάκελος	Περιγραφή
config/:	Τα αρχεία ρύθμισης παραμέτρων τις εφαρμογής
lib/:	Οι βιβλιοθήκες και οι κλάσεις της εφαρμογής
modules/:	Ο κώδικας της εφαρμογής (MVC)
templates/:	Τα καθολικά πρότυπα αρχεία

3.6.1 Δικαιώματα στις δομές των φακέλων

Πριν προσπαθήσουμε να έχουμε πρόσβαση στο καινούριο project που μόλις δημιουργήσαμε, θα πρέπει να ορίσουμε τα δικαιώματα εγγραφής στους φακέλους cache/ και log/ έτσι ώστε ο web server να μπορεί να γράψει σε αυτά:

$ chmod 777 cache/ log/

3.7 Ρυθμίσεις παραμέτρων Web Server: Ο ασφαλής τρόπος

Μια καλή πρακτική για το web είναι να μπουν στον web root φάκελο μόνο τα αρχεία που χρειάζεται να έχει πρόσβαση ο web browser, όπως stylesheets, JavaScripts, και εικόνες. Ως προεπιλογή, συνιστάται τα αρχεία αυτά να αποθηκεύονται στον υποφάκελο *web/* ενός project

symfony. Σε αυτόν τον φάκελο υπάρχουν μερικοί υποφάκελοι για web assets (*css/* και *images/*) και δύο αρχεία front controller. Τα front controllers είναι τα μόνα PHP αρχεία που χρειάζεται να βρίσκονται κάτω από τον web root φάκελο.

3.7.1 Ρυθμίσεις παραμέτρων του Web Server

Για να γίνει το νέο project προσβάσιμο στον κόσμο, θα πρέπει να αλλάξουμε τις ρυθμίσεις παραμέτρων του Apache.

Εντοπίζουμε και ανοίγουμε το αρχείο *httpd.conf* και προσθέτουμε τις ακόλουθες ρυθμίσεις στο τέλος του:

```
# Be sure to only have this line once in your configuration

NameVirtualHost 127.0.0.1:8080

# This is the configuration for your project

Listen 127.0.0.1:8080

<VirtualHost 127.0.0.1:8080>

 DocumentRoot "/home/sfprojects/jobeet/web"

 DirectoryIndex index.php

 <Directory "/home/sfprojects/jobeet/web">

AllowOverride All

Allow from All

 </Directory>

Alias /sf /home/sfprojects/jobeet/lib/vendor/symfony/data/web/sf

 <Directory "/home/sfprojects/jobeet/lib/vendor/symfony/data/web/sf">

  AllowOverride All

  Allow from All

 </Directory>

</VirtualHost>
```

Αυτές οι ρυθμίσεις κάνουν τον Apache να “ακούει” στην port 8080 στον υπολογιστή μας, έτσι, μετά από την επανεκκίνηση του Apache, η ιστοσελίδα θα είναι προσβάσιμη από το ακόλουθο URL:

http://~localhost~:8080/

Η port 8080 μπορεί να αλλαχθεί σε οποιονδήποτε αριθμό μεγαλύτερο του 1024 καθώς οι αριθμοί αυτοί δεν χρειάζονται δικαιώματα διαχειριστή.

3.7.2 Ελέγχουμε τις καινούριες ρυθμίσεις

Επανεκκινούμε τον Apache κι ελέγχουμε εάν μπορούμε να έχουμε πρόσβαση στην καινούρια εφαρμογή ανοίγοντας έναν browser και πληκτρολογώντας *http://localhost:8080/index.php*, ή *http://www.jobeet.com.localhost/index.php/* ανάλογα με τις ρυθμίσεις που επιλέξαμε.

Εικόνα 7: Πρώτη σελίδα εγκατάστασης του Symfony

Μπορούμε επίσης να δοκιμάσουμε να αποκτήσουμε πρόσβαση στην εφαρμογή στο περιβάλλον της ανάπτυξης:

http://www.jobeet.com.localhost/frontend_dev.php/

Στη γραμμή εργαλείων ελέγχου σφαλμάτων του web στην πάνω δεξιά γωνία θα πρέπει τα συμπεριλαμβανόμενα μικρά εικονίδια να αποδεικνύουν ότι η ρύθμιση του ψευδωνύμου *sf/* είναι σωστή.

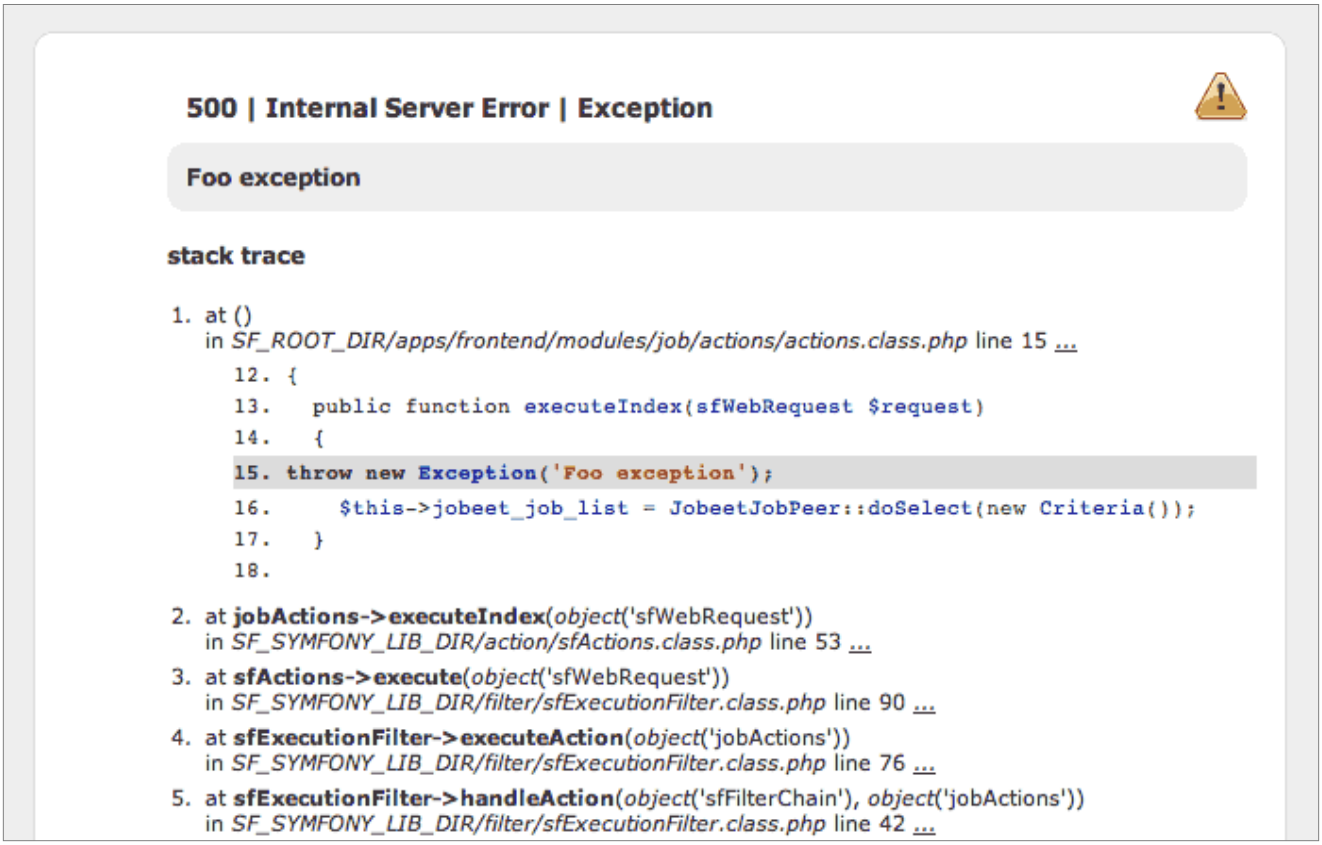
3.8 Τα περιβάλλοντα

Στον φάκελο *web/* υπάρχουν δύο PHP αρχεία: το *index.php* και το *frontend_dev.php* τα οποία ονομάζονται front controllers. Όλα τα αιτήματα προς την εφαρμογή γίνονται μέσω αυτών.

Και τα δύο τα αρχεία δείχνουν στην ίδια εφαρμογή αλλά σε διαφορετικά περιβάλλοντα. Όταν αναπτύσσεται μία εφαρμογή, εκτός από την περίπτωση που αναπτύσσεται απ' ευθείας στον production server, χρειάζονται διάφορα περιβάλλοντα.

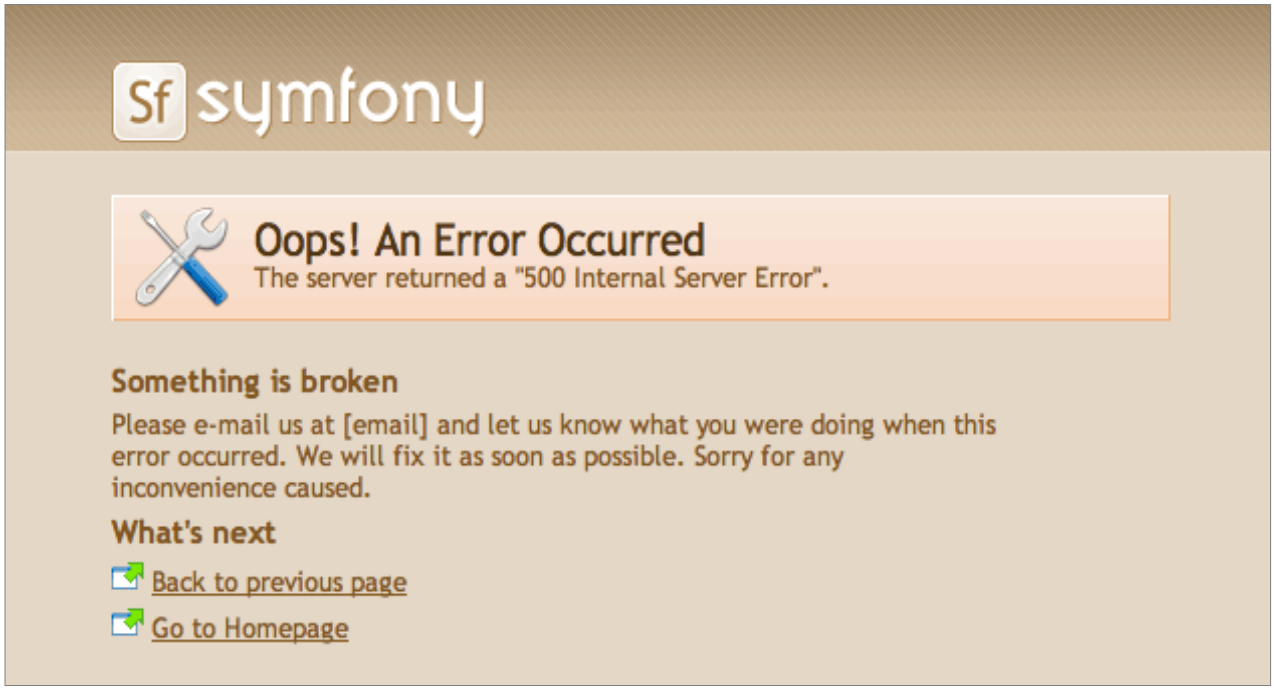
- Το περιβάλλον ανάπτυξης (development environment): Αυτό είναι το περιβάλλον που χρησιμοποιείται από τους web developers όταν εργάζονται στην εφαρμογή για να προσθέσουν νέα χαρακτηριστικά, διορθώσουν σφάλματα,...
- Το περιβάλλον ελέγχου (test environment): Αυτό το περιβάλλον χρησιμοποιείται για να ελέγχει αυτόματα την εφαρμογή.
- Το περιβάλλον σταδιοποίησης (staging environment): Αυτό το περιβάλλον χρησιμοποιείται από τον πελάτη για να ελέγξει την εφαρμογή και να αναφέρει σφάλματα ή χαρακτηριστικά τα οποία λείπουν.
- Το περιβάλλον παραγωγής (production environment): Αυτό είναι το περιβάλλον με το οποίο οι τελικοί χρήστες αλληλεπιδρούν.

Τι κάνει ένα περιβάλλον μοναδικό; Στο περιβάλλον ανάπτυξης για παράδειγμα, η εφαρμογή χρειάζεται να καταγράψει όλες τις λεπτομέρειες ενός αιτήματος για να διευκολύνει την αποσφαλμάτωση (debugging), αλλά το σύστημα cache θα πρέπει να είναι απενεργοποιημένο καθώς όλες οι αλλαγές που γίνονται στον κώδικα θα πρέπει να ληφθούν υπόψιν αμέσως. Έτσι, το περιβάλλον ανάπτυξης θα πρέπει να βελτιστοποιείται για τον developer. Το καλύτερο παράδειγμα είναι σίγουρα όταν μία εξαίρεση | Exception Handling συμβαίνει. Για να βοηθηθεί ένας developer στον εντοπισμό των σφαλμάτων γρηγορότερα, το symfony εμφανίζει την εξαίρεση (exception) με όλες τις πληροφορίες που έχει για το τρέχων αίτημα μέσα τον browser:



Εικόνα 7: Το περιβάλλον παραγωγής του Symfony που μας δείχνει το σφάλμα για να ενεργοποιηθεί

Αλλά στον περιβάλλον παραγωγής (production environment), πρέπει να ενεργοποιηθεί το επίπεδο cache (cache layer) και η εφαρμογή θα πρέπει να εμφανίζει προσαρμοσμένα μηνύματα σφαλμάτων αντί για ακατέργαστες εξαιρέσεις (raw exceptions). Έτσι, το περιβάλλον παραγωγής θα πρέπει να βελτιστοποιηθεί για επίδοση και την εμπειρία του χρήστη.



Εικόνα 8: Μετά της ρυθμίσεις του περιβάλλον παραγωγής (production environment).

3.9 To Project

Όπως είπαμε και παραπάνω το Jobeet είναι project ανοιχτού κώδικα το οποίο σαν θέμα έχει την εύρεση εργασίας. Είναι εύκολο στη χρήση, μπορεί να προσαρμοστεί, να επεκταθεί καθώς και να ενσωματωθεί μέσα στην ιστοσελίδα. Υποστηρίζει πολλαπλές γλώσσες, και χρησιμοποιεί τις τελευταίες τεχνολογίες WEB 2.0 για να ενισχύσει την εμπειρία του χρήστη. Επίσης παρέχει feeds και μια διεπαφή API (Application Programming Interface) για να μπορούμε να αλληλεπιδρούμε με αυτό προγραμματιστικά.

Πριν ασχοληθούμε με τον κώδικα θα δούμε μερικά ακόμα πράγματα για το project. Θα περιγράψουμε τα χαρακτηριστικά που θα εφαρμόσουμε στην πρώτη έκδοση του project με κάποιες απλές ιστορίες.

Το Jobeet website έχει τέσσερα είδη χρηστών:

- **Admin:** Του ανήκει το website και έχει τη "μαγική δύναμη"
- **user:** Επισκέπτεται το site για να βρει μια θέση εργασίας
- **poster:** Επισκέπτεται το site για να τοποθετήσει μία θέση εργασίας
- **affiliate:** Αναδημοσιεύει κάποιες θέσεις εργασίας στο δικό του site.

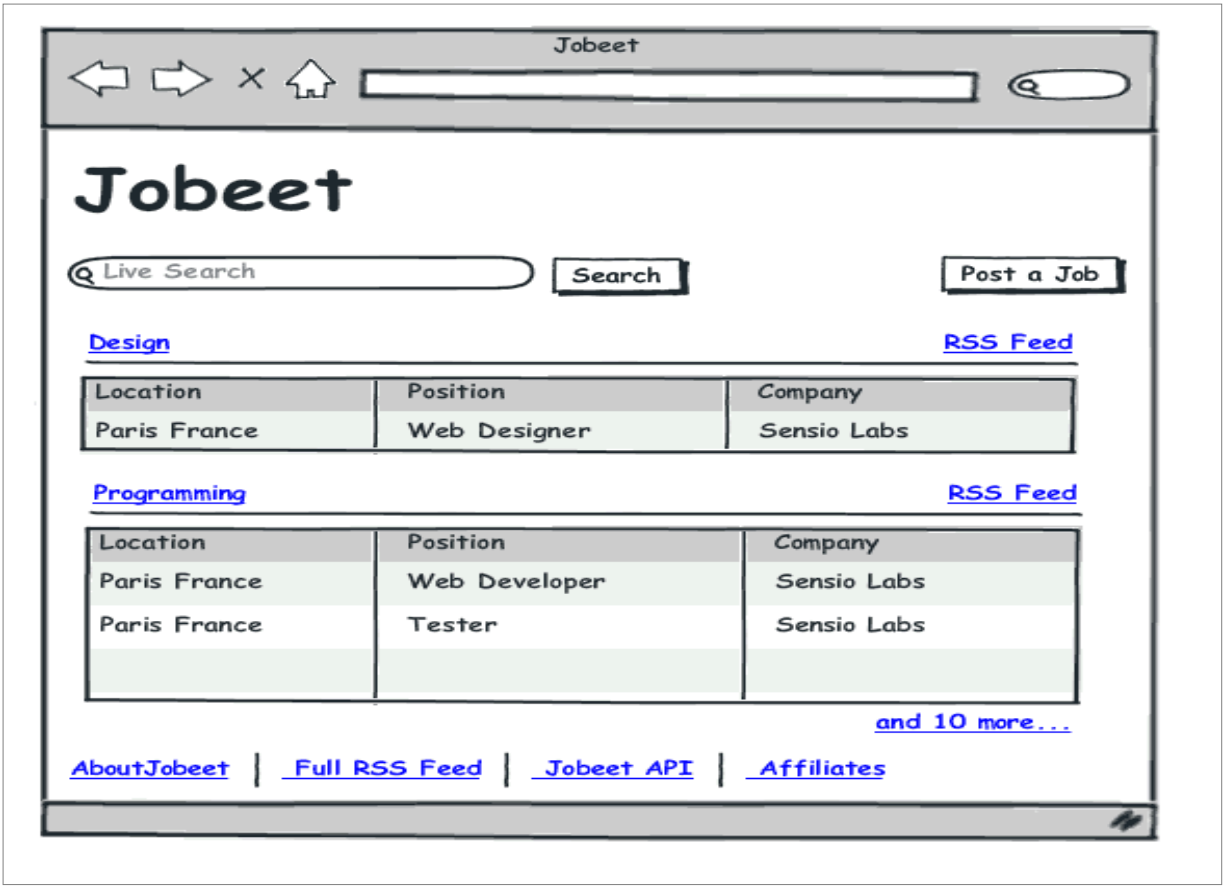
Το Project έχει δύο εφαρμογές: την frontend (ιστορίες F1 έως F7 παρακάτω), όπου οι χρήστες αλληλεπιδρούν με το website, και την backend (ιστορίες B1 έως B3), όπου οι διαχειριστές (admins) διαχειρίζονται το website.Η backend εφαρμογή έχει ασφάλεια και χρειάζονται διαπιστευτήρια για να έχει κάποιος πρόσβαση σε αυτή.

3.9.1 Ιστοριες F

3.9.1.1 Ιστορία F1: Στην αρχική σελίδα ο χρήστης βλέπει τις τελευταίες ενεργές θέσεις εργασίας.

Όταν ο χρήστης εισέρχεται στην ιστοσελίδα, βλέπει μία λίστα από τις ενεργές θέσεις εργασίας. Οι θέσεις εργασίας είναι ταξινομημένες ανά κατηγορία και ανά ημερομηνία δημοσίευσης (οι νεότερες θέσεις εργασίας πρώτα). Για κάθε θέση εργασίας, εμφανίζεται μόνο η τοποθεσία, η θέση εργασίας, και η εταιρεία.

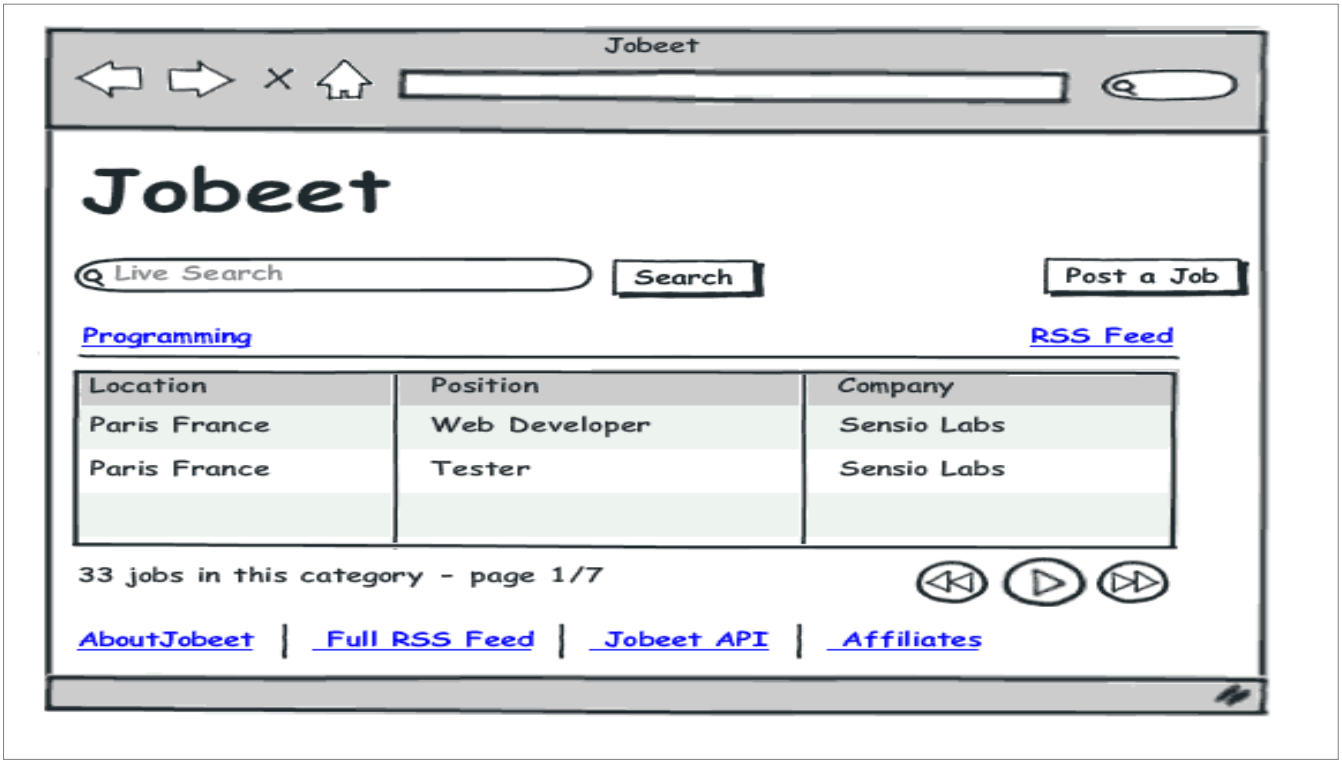
Για κάθε κατηγορία, η λίστα δείχνει μόνο τις πρώτες 10 και ένα link επιτρέπει να προβάλει όλες τις θέσεις εργασίας σε συγκεκριμένη κατηγορία (Ιστορία F2).Στην αρχική σελίδα, ο χρήστης μπορεί να καθαρίσει τη λίστα με τις θέσεις εργασίας (Ιστορία F3), ή να τοποθετήσει μία καινούρια θέση εργασίας (Ιστορία F5).



Εικόνα 9: Ιστορία F1: Στην αρχική σελίδα ο χρήστης βλέπει τις τελευταίες ενεργές θέσεις εργασίας.

3.9.1.2 Ιστορία F2: Ένας χρήστης μπορεί να αναζητήσει όλες τις θέσεις εργασίες σε συγκεκριμένη κατηγορία

Όταν ο χρήστης "πατάει" σε ένα όνομα κατηγορίας ή σε ένα link "more jobs" στην αρχική σελίδα, βλέπει όλες τις θέσεις εργασίας γι αυτή την κατηγορία ταξινομημένες κατά ημερομηνία.



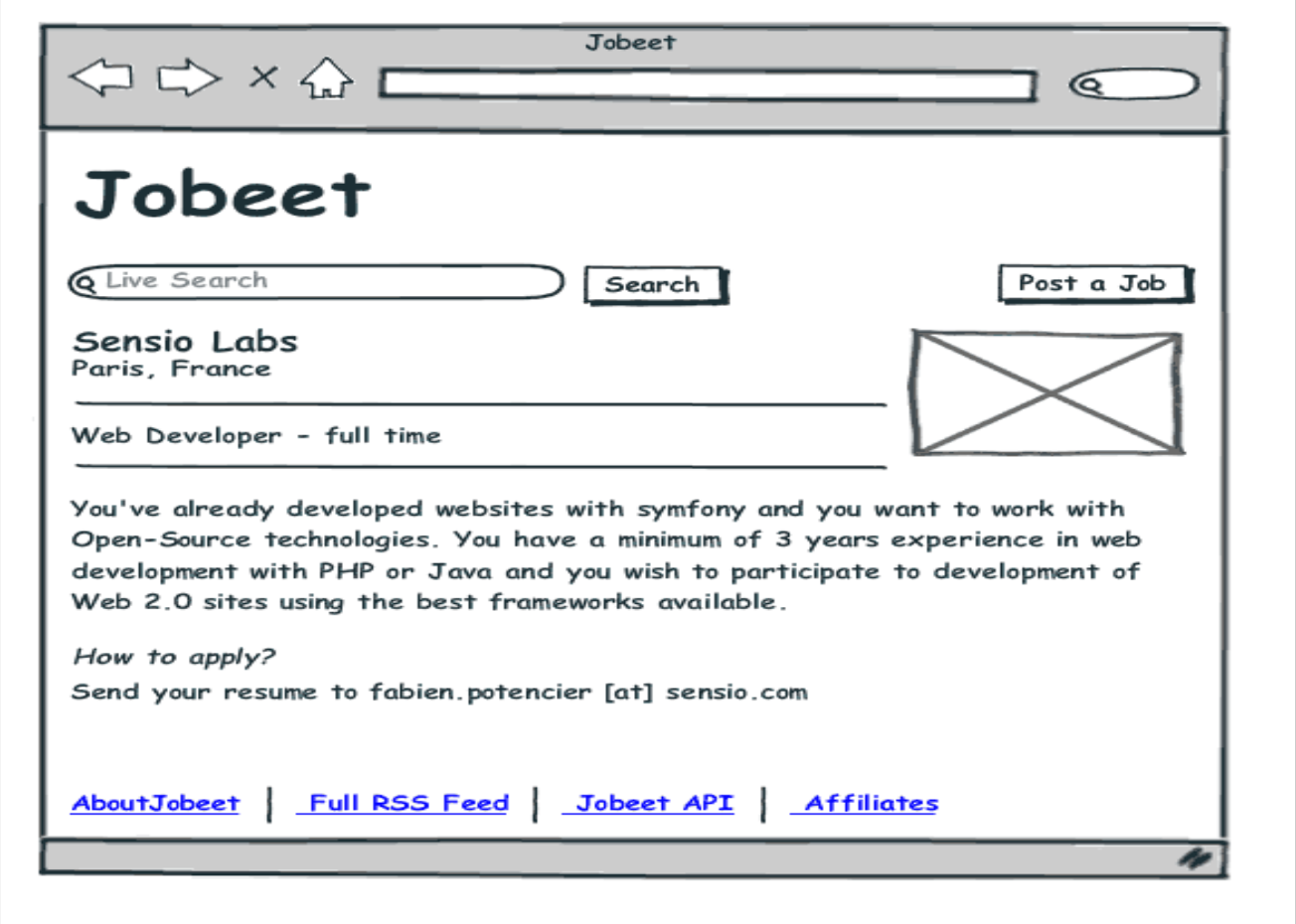
Εικόνα 9: Ιστορία F2: Ένας χρήστης μπορεί να αναζητήσει όλες τις θέσεις εργασίες σε συγκεκριμένη κατηγορία

3.9.1.3 Ιστορία F3: Ένας χρήστης βελτιώνει τα αποτελέσματα της λίστας με κάποιες λέξεις-κλειδιά

Ο χρήστης μπορεί να εισάγει κάποιες λέξεις-κλειδιά για να βελτιώσει την εύρεση. Οι λέξεις-κλειδιά μπορούν να είναι λέξεις που αναφέρονται στην τοποθεσία, την θέση, την κατηγορία, ή τα πεδία τις εταιρείας.

3.9.1.4 Ιστορία F4: Ο χρήστης "πατάει" πάνω σε μια θέση εργασίας για να δει πιο λεπτομερείς πληροφορίες

Ο χρήστης μπορεί να διαλέξει μία θέση εργασίας από τη λίστα για να δει πιο λεπτομερείς πληροφορίες.



Εικόνα 10: Ιστορία F4: Ο χρήστης "πατάει" πάνω σε μια θέση εργασίας για να δει πιο λεπτομερείς πληροφορίες

3.9.1.5 Ιστορία F5: Ο χρήστης τοποθετεί μία θέση εργασίας

Ένας χρήστης μπορεί να τοποθετήσει μία θέση εργασίας. Μια θέση εργασίας απαρτίζεται από αρκετά κομμάτια από πληροφορίες:

- Εταιρεία
- Τύπος εργασίας (full-time, part-time, ή ανεξάρτητη)
- Λογότυπο (προαιρετικά)
- URL (προαιρετικά)
- Θέση

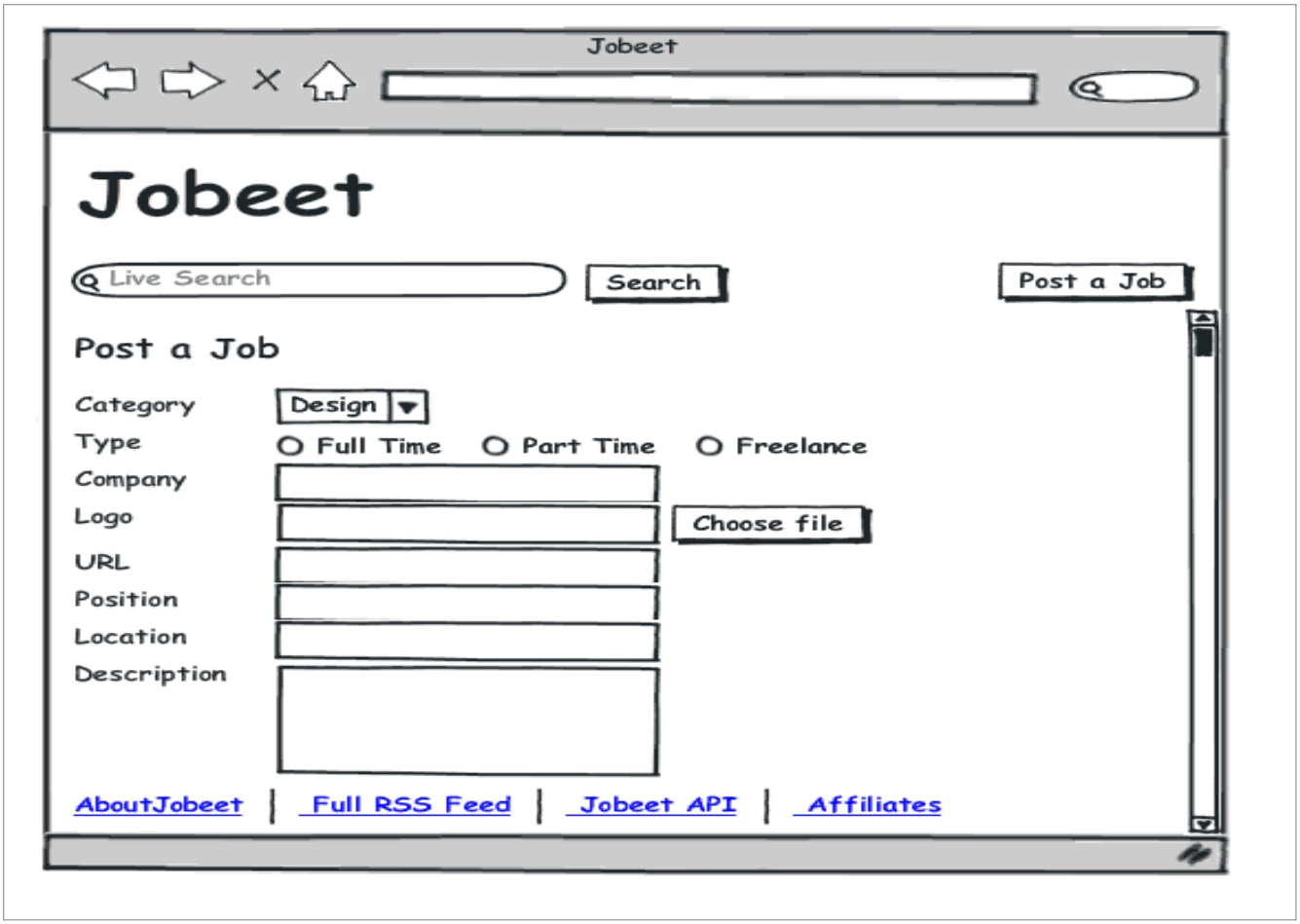
- Τοποθεσία
- Κατηγορία (Ο χρήστης διαλέγει σε λίστα τις πιθανές κατηγορίες)
- Περιγραφή εργασίας (τα URL και τα email συνδέονται αυτόματα)
- Πως να γίνει η αίτηση (τα URL και τα email συνδέονται αυτόματα)
- Email

Δεν χρειάζεται η δημιουργία λογαριασμού για να τοποθετηθεί μία θέση εργασίας.

Η διαδικασία χρειάζεται μόνο δύο βήματα: πρώτα, ο χρήστης συμπληρώνει τη φόρμα με όλες τις απαραίτητες πληροφορίες για να περιγράψει τη θέση εργασίας, μετά επιβεβαιώνει τις πληροφορίες κάνοντας προ-επισκόπηση της τελικής σελίδας για τη θέση εργασίας.

Ακόμα κι αν ο χρήστης δεν έχει λογαριασμό, μία θέση εργασίας μπορεί να τροποποιηθεί κι έπειτα από την τοποθέτησή της χάρις σε ένα συγκεκριμένο URL.

Κάθε αναρτημένη θέση εργασίας παραμένει διαθέσιμη (online) για 30 ημέρες (αυτό ρυθμίζεται από τον διαχειριστή). Ένας χρήστης μπορεί να ξανά ενεργοποιήσει ή να επεκτείνει την εγκυρότητα μιας θέσης εργασίας για ακόμα 30 ημέρες αλλά μόνο όταν η θέση εργασίας λήγει σε λιγότερο από 5 ημέρες.



Εικόνα 11: Ιστορία F5: Ο χρήστης τοποθετεί μία θέση εργασίας

3.9.1.6 στορία F6: Ένας χρήστης κάνει αίτηση για να προωθήσει το site

Ένας χρήστης χρειάζεται να κάνει αίτηση και να εγκριθεί για να μπορέσει να προωθήσει και να χρησιμοποιήσει το Jobeet API. Για να κάνει την αίτηση, θα πρέπει να δώσει τις παρακάτω πληροφορίες:

- Όνομα
- Email
- Website URL

Ο λογαριασμός θα πρέπει να ενεργοποιηθεί από τον διαχειριστή (Ιστορία Β3).

3.9.1.7 Ιστορία F7: Ένας προωθητής παραλαμβάνει την υπάρχουσα λίστα με τις θέσεις εργασίας

Ένας προωθητής μπορεί να παραλάβει την υπάρχουσα λίστα με τις θέσεις εργασίας καλώντας το API με το token. Η λίστα μπορεί να επιστραφεί σε διαμόρφωση XML, JSON ή YAML.
Η λίστα περιέχει τις δημόσιες πληροφορίες που είναι διαθέσιμες για μία θέση εργασίας.

3.9.2 Ιστορίες Β

3.9.2.1 Ιστορία Β1: Ένας διαχειριστής (admin) ρυθμίζει τις παραμέτρους για το website

Ένας διαχειριστής μπορεί να επεξεργαστεί τις κατηγορίες που είναι διαθέσιμες στο website.

3.9.2.2 Ιστορία Β2: Ένας διαχειριστής (admin) διαχειρίζεται τις θέσεις εργασίας

Ένας διαχειριστής μπορεί να επεξεργαστεί και να αφαιρέσει κάθε θέση εργασίας που έχει τοποθετηθεί.

3.9.2.3 Ιστορία Β3: Ένας διαχειριστής (admin) διαχειρίζεται τους προωθητές

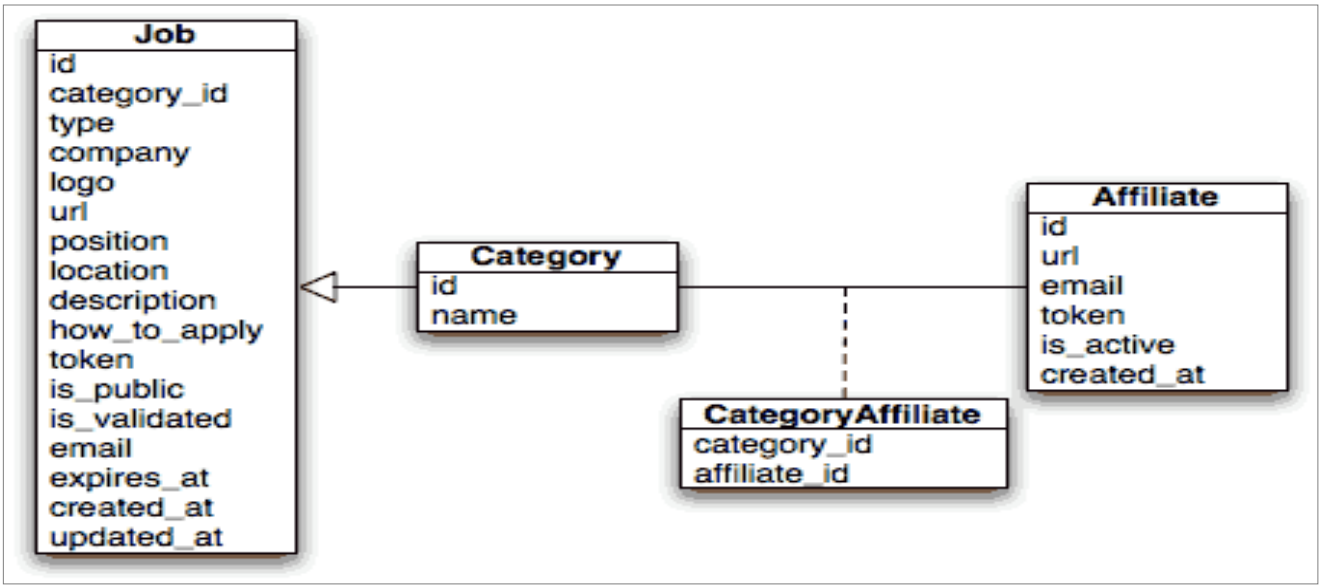
Ο διαχειριστής μπορεί να δημιουργήσει ή να επεξεργαστεί τους προωθητές καθώς είναι υπεύθυνος και για την ενεργοποίηση η απενεργοποίηση αυτών.

Όταν ο διαχειριστής ενεργοποιεί έναν νέο προωθητή, το σύστημα δημιουργεί ένα μοναδικό token για να χρησιμοποιηθεί από τον προωθητή.

4 Το Μοντέλο Δεδομένων (Data Model)

4.1 Το σχεσιακό μοντέλο (relational model)

Οι ιστορίες που είδαμε παραπάνω αναφέρονται στα κύρια αντικείμενα του project: θέση εργασίας (job), προωθήσεις (affiliate) και κατηγορία (category). Εδώ είναι το αντίστοιχο διάγραμμα:



Εικόνα 12: Το Μοντέλο μοντέλο Δεδομένων (Data Model).

4.2 Το σχέδιο (schema)

Για να τα αποθηκεύσουμε αυτά χρειαζόμαστε μια σχεσιακή βάση δεδομένων. Καθώς το symfony είναι ένα αντικειμενοστραφή (Object-Oriented) framework αντί να γράφουμε SQL statements μπορούμε για να ανακτήσουμε εγγραφές από τη βάση δεδομένων προτιμούμε να χρησιμοποιούμε αντικείμενα (objects).

Μπορεί να γίνει αντιστοίχιση των πληροφοριών της σχεσιακής βάσης δεδομένων με ένα object model (μοντέλο αντικειμένου) χρησιμοποιώντας ένα εργαλείο ORM. Το symfony έρχεται με δύο τέτοια εργαλεία: το Propel και το Doctrine. Εμείς θα χρησιμοποιήσουμε το Doctrine.

4.3 Το ORM

χρειάζεται την περιγραφή των tables και τις σχέσεις τους για να δημιουργήσει τις σχετικές classes. Υπάρχουν δύο τρόποι για να δημιουργήσουμε αυτό το σχέδιο της περιγραφής (description schema): είτε με το να αυτοεξετάσουμε μία υπάρχουσα βάση δεδομένων ή δημιουργώντας την από την αρχή.

Καθώς η βάση δεδομένων δεν υπάρχει ακόμα θα δημιουργήσουμε το αρχείο του σχεδίου (schema file) επεξεργάζοντας το κενό αρχείο *config/doctrine/schema.yml*:

```
# config/doctrine/schema.yml
JobeetCategory:
  actAs: { Timestampable: ~ }
  columns:
    name: { type: string(255), notnull: true, unique: true }
JobeetJob:
  actAs: { Timestampable: ~ }
  columns:
    category_id:  { type: integer, notnull: true }
    type:         { type: string(255) }
    company:      { type: string(255), notnull: true }
    logo:         { type: string(255) }
    url:          { type: string(255) }
    position:     { type: string(255), notnull: true }
    location:     { type: string(255), notnull: true }
    description:  { type: string(4000), notnull: true }
    how_to_apply: { type: string(4000), notnull: true }
    token:        { type: string(255), notnull: true, unique: true }
    is_public:    { type: boolean, notnull: true, default: 1 }
    is_activated: { type: boolean, notnull: true, default: 0 }
    email:        { type: string(255), notnull: true }
    expires_at:   { type: timestamp, notnull: true }
  relations:
    JobeetCategory: { onDelete: CASCADE, local: category_id, foreign: id, foreignAlias: JobeetJobs }
JobeetAffiliate:
```

```
actAs: { Timestampable: ~ }
columns:
  url:       { type: string(255), notnull: true }
  email:     { type: string(255), notnull: true, unique: true }
  token:     { type: string(255), notnull: true }
  is_active: { type: boolean, notnull: true, default: 0 }
relations:
  JobeetCategories:
    class: JobeetCategory
    refClass: JobeetCategoryAffiliate
    local: affiliate_id
    foreign: category_id
    foreignAlias: JobeetAffiliates
JobeetCategoryAffiliate:
 columns:
  category_id:  { type: integer, primary: true }
  affiliate_id: { type: integer, primary: true }
 relations:
  JobeetCategory:  { onDelete: CASCADE, local: category_id, foreign: id }
  JobeetAffiliate: { onDelete: CASCADE, local: affiliate_id, foreign: id }
```

Το schema (σχέδιο) είναι η απευθείας μετάφραση (translation) του entity relationship diagram (διαγράμματος σχέσης οντότητας) στην YAML format.

Το αρχείο *schema.yml* περιλαμβάνει την περιγραφή όλων των table και τις στήλες (columns) αυτών. Κάθε στήλη περιγράφεται με τις παρακάτω πληροφορίες:

- **Type:** Ο τύπος των στηλών. (boolean, integer, float, decimal, string, array, object, blob, clob, timestamp, time, date, enum, gzip)
- **Notnull:** Το βάζουμε true αν θέλουμε να απαιτείται η στήλη

- **Unique:** Το βάζουμε true αν θέλουμε να δημιουργήσουμε έναν μοναδικό δείκτη για τη στήλη

4.4 Η Βάση Δεδομένων

Το symfony framework υποστηρίζει όλες τις βάσεις δεδομένων που υποστηρίζονται από PDO. Εμείς θα χρησιμοποιήσουμε MySQL:

$ mysqladmin -uroot -p create jobeet

*Enter password: mYsEcret ## The password will echo as *********

Πρέπει να πούμε στο symfony να χρησιμοποιήσει αυτή τη βάση δεδομένων για το Jobeet project:

$ php symfony configure:database

"mysql:host=localhost;dbname=jobeet" root mYsEcret

Η διεργασία *configure:database* παίρνει τρία arguments: το PDO DSN, το username, και το password για την πρόσβαση στη βάση δεδομένων.

4.3 Το ORM

Χάρη στην περιγραφή της βάσης δεδομένων από το αρχείο *schema.yml*, μπορούμε να χρησιμοποιήσουμε μερικές διεργασίες του Doctrine για να παράγουμε τα SQL statements που χρειάζονται για να δημιουργήσουμε τα tables της βάσης δεδομένων:

Για να παράγουμε το SQL θα πρέπει να δημιουργήσουμε τα models από τα αρχεία schema.

$ php symfony doctrine:build --model

Τώρα μπορούμε να παράγουμε και να εισάγουμε το SQL.

$ php symfony doctrine:build -- sql

Η διεργασία *doctrine:build - - sql* παράγει SQL statements στον φάκελο *data/sql/* για την μηχανή της βάσης δεδομένων που έχουμε ρυθμίσει:

 # snippet from data/sql/schema.sql

CREATE TABLE jobeet_category (id BIGINT AUTO_INCREMENT), name VARCHAR(255)

NOT NULL COMMENT 'test', created_at DATETIME, updated_at DATETIME, slug VARCHAR(255), UNIQUE INDEX sluggable_idx (slug), PRIMARY KEY(id))

ENGINE = INNODB;

Για να δημιουργήσουμε τα tables στην βάση δεδομένων πρέπει να τρέξουμε την διεργασία *doctrine:insert-sql*:

$ php symfony doctrine:insert-sql

Το ORM επίσης παράγει PHP classes που χαρτογραφούν table records σε objects:

$ php symfony doctrine:build - - model

Η διεργασία *doctrine:build - - model* παράγει αρχεία PHP στον φάκελο *lib/model/* τα οποία μπορούν να χρησιμοποιηθούν για να αλληλεπιδράσουν με τη βάση δεδομένων.

Το Doctrine παράγει τρεις classes ανά table. Για το jobeet_job table:

- **JobeetJob:** Ένα αντικείμενο αυτής της κλάσης αντιπροσωπεύει μία εγγραφή του jobeet_job table. Η κλάση είναι άδεια από προεπιλογή.
- **BaseJobeetJob:** Η γονέας κλάση του JobeetJob.
- **JobeetJobTable:** Η κλάση καθιερώνει μεθόδους που κυρίως επιστρέφουν συλλογές από αντικείμενα JobeetJob. Η κλάση είναι άδεια από προεπιλογή.

Η στήλη values μιας εγγραφής μπορεί να "χειραγωγηθεί" (manipulate) με ένα model object χρησιμοποιώντας κάποια *accessors(get*() methods)* και *mutators(set*()methods)*:

$job = new JobeetJob();

$job->setPosition('Web developer');

$job->save();

echo $job->getPosition();

$job->delete();

Μπορούμε επίσης να καθορίσουμε ξένα κλειδιά απευθείας, συνδέοντας τα objects μαζί:

$category = new JobeetCategory();

$category->setName('Programming');

$job = new JobeetJob();

$job->setCategory($category);

Η διεργασία *doctrine:build - - all* είναι μία συντόμευση για τις εργασίες που έχουν ήδη τρέξει σε αυτόν τον τομέα. Οπότε το τρέχουμε για να παράγουμε φόρμες και επικυρωτές για τα model classes του Jobeet:

$php symfony doctrine:build - - all - - no – confirmation

4.5 Τα αρχικά δεδομένα

Τα tables έχουν δημιουργηθεί στην βάση δεδομένων αλλά δεν έχουν καθόλου δεδομένα μέσα. Για κάθε εφαρμογή web, υπάρχουν τρεις τύποι δεδομένων:

- **Initial data (αρχικά δεδομένα):** Τα αρχικά δεδομένα χρειάζονται για να λειτουργήσει η εφαρμογή.
- **Test data (δεδομένα ελέγχου):** Τα δεδομένα ελέγχου χρειάζονται για να ελεγχθεί η εφαρμογή.
- **User data (δεδομένα χρήστη):** Τα δεδομένα χρήστη δημιουργούνται από τους χρήστες κατά τη διάρκεια της "κανονικής ζωής" της εφαρμογής.

Κάθε φορά που το symfony δημιουργεί τα tables στη βάση δεδομένων, όλα τα δεδομένα χάνονται. Για να συμπληρώσουμε την βάση δεδομένων με κάποια αρχικά δεδομένα, θα μπορούσαμε να φτιάξουμε ένα PHP script, ή να εκτελέσουμε κάποια SQL statements με το πρόγραμμα *mysql*. Με το symfony όμως υπάρχει καλύτερος τρόπος: δημιουργούμε αρχεία YAML στον φάκελο *data/fixtures/* και χρησιμοποιώντας την διεργασία *doctrine:data- load* μπορούμε να τα φορτώσουμε μέσα στην βάση δεδομένων.

Πρώτα φτιάχνουμε τα αναπόσπαστα αρχεία:

 # data/fixtures/categories.yml

JobeetCategory:

design:

name: Design

programming:

name: Programming

manager:

name: Manager

administrator:

```
    name: Administrator
# data/fixtures/jobs.yml
JobeetJob:
  job_sensio_labs:
    JobeetCategory: programming
    type:         full-time
    company:      Sensio Labs
    logo:         sensio-labs.gif
    url:          http://www.sensiolabs.com/
    position:     Web Developer
    location:     Paris, France
    description:  |
      You've already developed websites with symfony and you want to work
      with Open-Source technologies. You have a minimum of 3 years
      experience in web development with PHP or Java and you wish to
      participate to development of Web 2.0 sites using the best
      frameworks available.
    how_to_apply: |
      Send your resume to fabien.potencier [at] sensio.com
    is_public:    true
    is_activated: true
    token:        job_sensio_labs
    email:        job@example.com
    expires_at:   '2010-10-10'
  job_extreme_sensio:
    JobeetCategory:  design
    type:         part-time
```

```
company:      Extreme Sensio
logo:         extreme-sensio.gif
url:          http://www.extreme-sensio.com/
position:     Web Designer
location:     Paris, France
description:  |
  Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do
  eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut
  enim ad minim veniam, quis nostrud exercitation ullamco laboris
  nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor
  in reprehenderit in.
  Voluptate velit esse cillum dolore eu fugiat nulla pariatur.
  Excepteur sint occaecat cupidatat non proident, sunt in culpa
  qui officia deserunt mollit anim id est laborum.
how_to_apply: |
  Send your resume to fabien.potencier [at] sensio.com
is_public:    true
is_activated: true
token:        job_extreme_sensio
email:        job@example.com
expires_at:   '2010-10-10'
```

Ένα αρχείο προγράμματος γράφεται σε YAML και ορίζει τα model objects τα οποία επισημαίνονται με ένα μοναδικό όνομα. Η επισήμανση έχει σαν χρήση την σύνδεση συγγενών αντικειμένων χωρίς να χρειάζεται να ορίσουμε πρωτεύοντα κλειδιά.

Σε ένα αρχείο προγράμματος δεν είναι απαραίτητο να ορίσουμε όλες τις τιμές των στηλών μιας και το symfony θα βάλει τις προεπιλεγμένες τιμές, σε ότι είναι κενό, χρησιμοποιώντας το σχέδιο της βάσης δεδομένων που έχουν οριστεί προηγουμένως.

Τα αρχικά δεδομένα (initial data) φορτώνονται μέσα στην βάση δεδομένων εκτελώντας το παρακάτω:

$ php symfony doctrine:data-load

Εκτελώντας το *doctrine:build - - all - - and – load* τα πάντα όπως, φόρμες, φίλτρα, μοντέλα, παράγονται από το σχέδιο (schema), η βάση δεδομένων θα πέσει, και θα ξανά δημιουργηθούν τα tables.

$ php symfony doctrine:build - - all - - and – load

4.6 Τα βλέπουμε στην πράξη στον Browser (περιηγητή).

Τώρα θα μάθουμε πως ο web browser μπορεί να αλληλεπιδράσει με τη βάση δεδομένων. Όπως έχουμε πει πιο πάνω, το symfony project είναι δημιουργημένο από εφαρμογές (applications). Κάθε εφαρμογή διαιρείται σε ενότητες (modules). Κάθε ενότητα αποτελείται από κώδικα PHP ο οποίος αντιπροσωπεύει ένα χαρακτηριστικό της εφαρμογής ή ένα σύνολο από χειρισμούς που μπορεί να κάνει ο χρήστης σε ένα model object.

Το symfony μπορεί να παράγει αυτόματα μια ενότητα (module) για ένα model το οποίο παρέχει βασικά χαρακτηριστικά χειρισμών:

$ php symfony doctrine:generate-module --with-show

--non-verbose-templates frontend job JobeetJob

Το *doctrine:generate-module* παράγει ένα job module στην frontend εφαρμογή για το JobeetJob model. Όπως με τα περισσότερες εργασίες του symfony μερικά αρχεία και φάκελοι έχουν δημιουργηθεί κάτω από τον φάκελο *apps/frontend/modules/job/*:

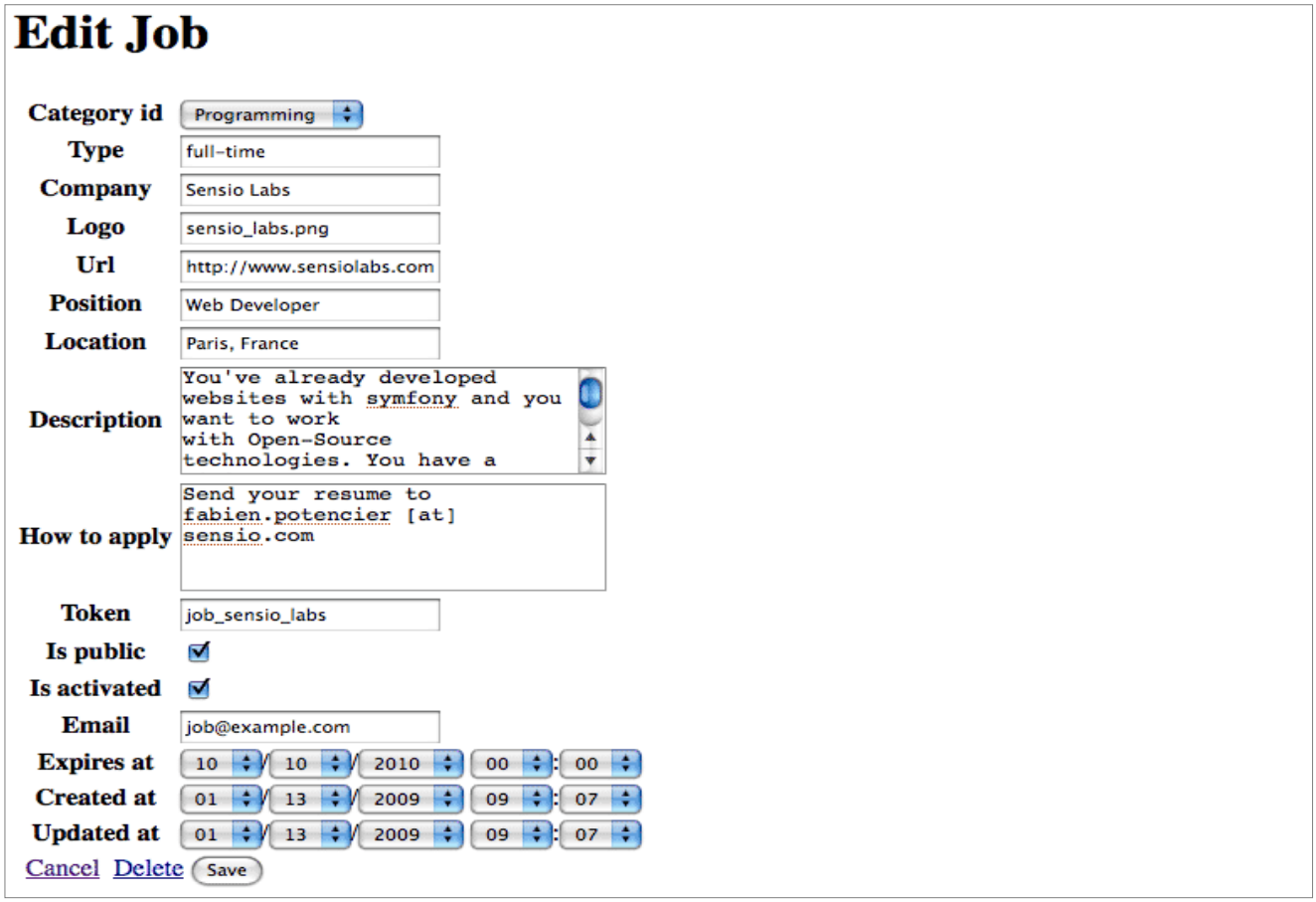
Φάκελος	**Περιγραφή**
actions/:	Η ενότητα ενεργειών
templates/:	Η ενότητα πρότυπα

Το αρχείο *actions/actions.class.php* ορίζει όλες τις διαθέσιμες ενέργειες (action) για το Job module:

Όνομα ενέργειας	Περιγραφή
Index:	Εμφανίζει τις εγγραφές του table
Show:	Εμφανίζει τα πεδία και τις τιμές για συγκεκριμένη εγγραφή
New:	Εμφανίζει μία φόρμα για τη δημιουργία μιας νέας εγγραφής
Create:	Δημιουργεί μία καινούρια εγγραφή
Edit:	Εμφανίζει μία φόρμα για να γίνει επεξεργασία συγκεκριμένης εγγραφής
Update:	Ενημερώνει μία εγγραφή με βάση τις τιμές που έχει ορίσει ο χρήστης
Delete:	Διαγράφει μία συγκεκριμένη εγγραφή από το table

Τώρα μπορούμε να ελέγξουμε το job module στον browser:

http://www.jobeet.com.localhost/frontend_dev.php/job



Εικόνα 13: Στην πράξη στον Browser (περιηγητή).

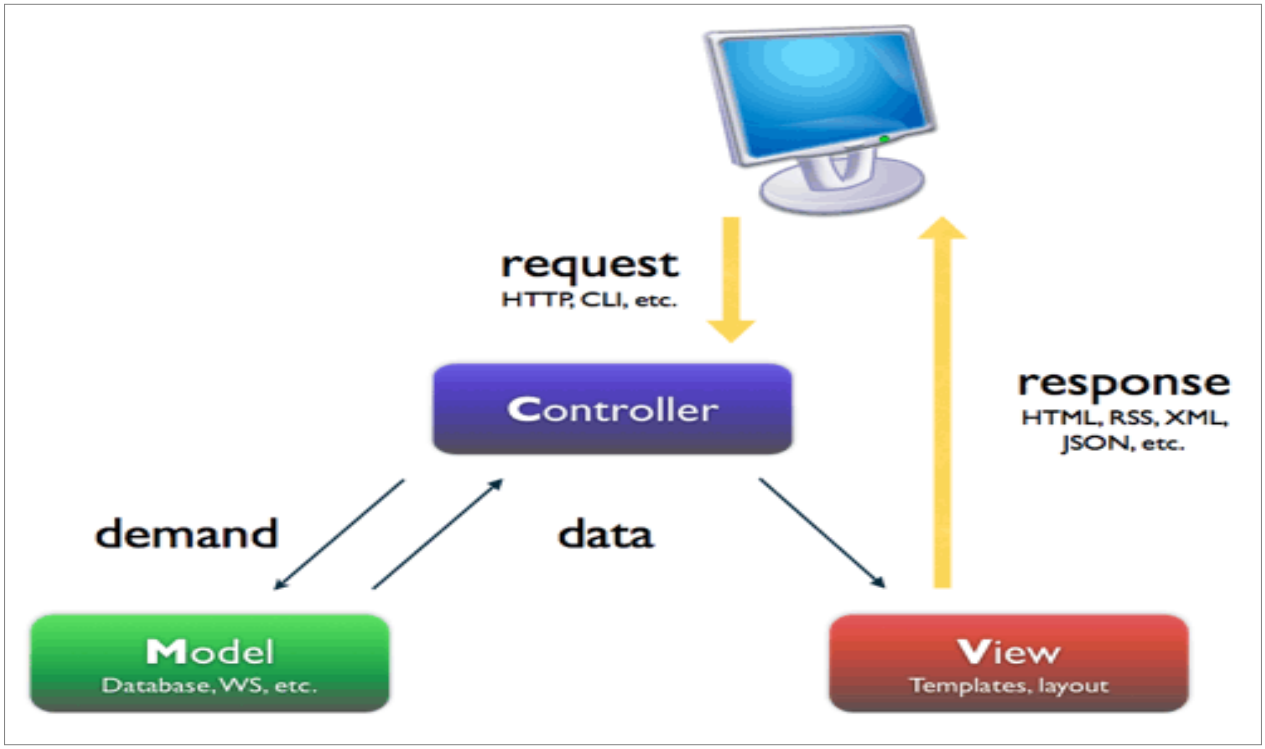
Η λίστα που εμφανίζεται παίρνει τις τιμές επιλογών από τη μέθοδο *_toString()*. Εάν θέλουμε να φτιάξουμε δικιά μας λίστα θα πρέπει να φτιάξουμε μία μέθοδο *_toString()* όπως παρακάτω:

```
// lib/model/doctrine/JobeetJob.class.php
class JobeetJob extends BaseJobeetJob
{
  public function __toString()
  {
    return sprintf('%s at %s (%s)', $this->getPosition(), $this->getCompany(), $this->getLocation());
  }
}
// lib/model/doctrine/JobeetAffiliate.class.php
class JobeetAffiliate extends BaseJobeetAffiliate
{
  public function __toString()
  {
    return $this->getUrl();
  }
```

5.Η Αρχιτεκτονική MVC

Η αρχή της αρχιτεκτονικής του MVC είναι να διαχωρίσει τον κώδικα σε τρία layers, σύμφωνα με τη φύση του. Ο κώδικας λογικής δεδομένων είναι τοποθετημένος μέσα στο model, ο κώδικας της παρουσίασης μέσα στο view, και η λογική της εφαρμογής μέσα στον controller. Άλλα επιπρόσθετα σχεδιαστικά πρότυπα μπορούν να κάνουν την εμπειρία του προγραμματισμού ακόμα ευκολότερη. Τα επίπεδα model, view και controller μπορούν να υποδιαιρεθούν περαιτέρω.

- **Model :** Τα δεδομένα μας ή αλλιώς η επιχειρησιακή λογική της εφαρμογής μας
- **View :** Το επίπεδο παρουσίασης, δηλαδή το πώς εμφανίζονται τα αποτελέσματα / δεδομένα
- **Controller:** Η διεπαφή με τον χρήστη και ουσιαστικά η λειτουργικότητα της εφαρμογής



Εικόνα 14 : Η Αρχιτεκτονική MVC.

5.1 To Layout

Το layout είναι ένας μηχανισμός στο symfony ο οποίος λειτουργεί αποτρεπτικά ως προς το να δημιουργηθούν επικαλύψεις κώδικα (code duplication). Το layout αυτό που κάνει είναι να περιβάλει το template (πρότυπο) όπως δείχνει η εικόνα παρακάτω:

Εικόνα 15: To Layout

Το προεπιλεγμένο layout μίας εφαρμογής ονομάζεται *layout.php* και βρίσκεται στον φάκελο */apps/frontend/templates/*. Αυτός ο φάκελος περιέχει όλα τα καθολικά πρότυπα για μία εφαρμογή.

Αντικαθιστούμε τον προεπιλεγμένο layout του symfony με τον παρακάτω κώδικα:

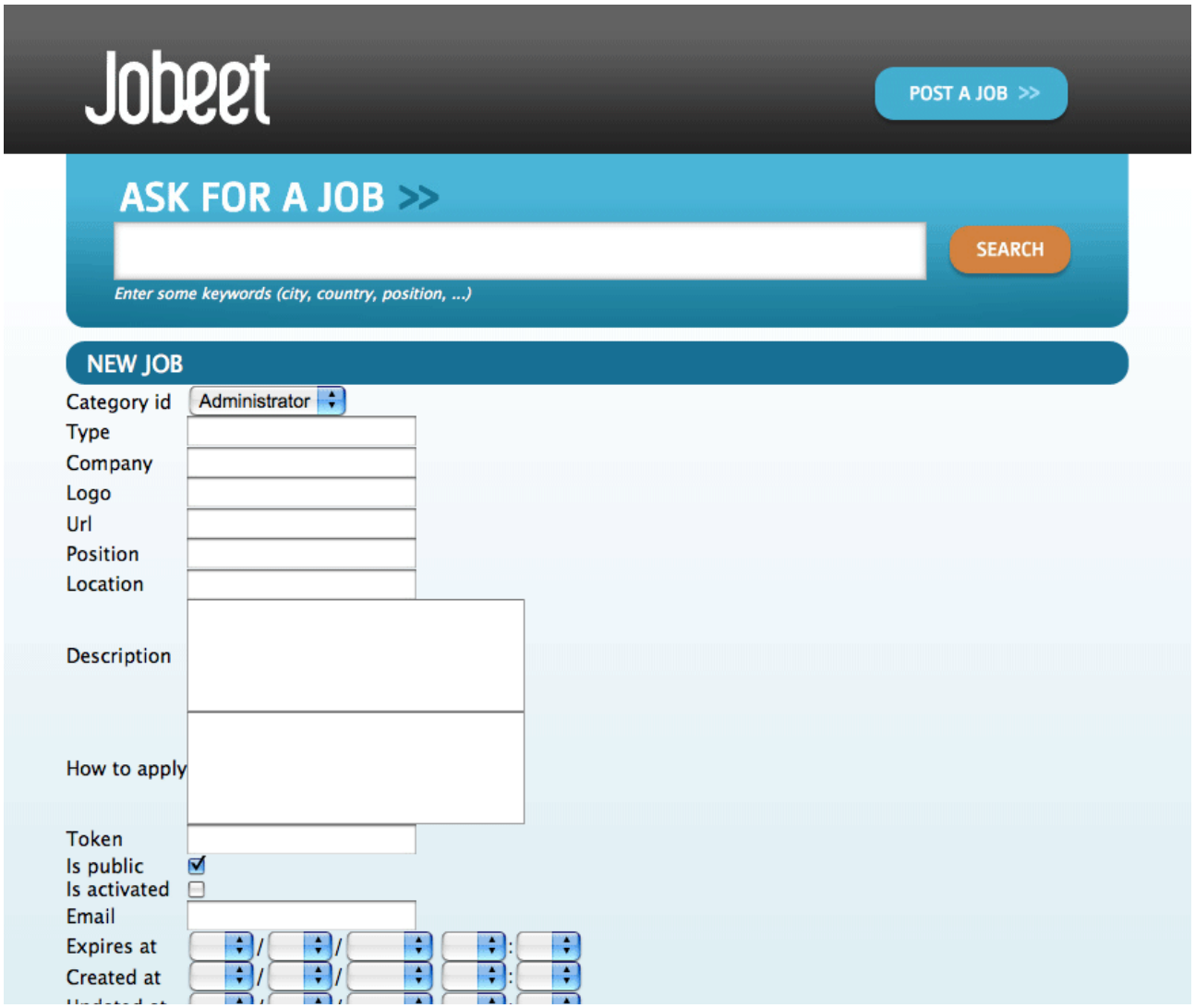
```
<!-- apps/frontend/templates/layout.php -->
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
  <head>
    <title>Jobeet - Your best job board</title>
    <link rel="shortcut icon" href="/favicon.ico" />
    <?php include_javascripts() ?>
    <?php include_stylesheets() ?>
  </head>
  <body>
    <div id="container">
      <div id="header">
        <div class="content">
          <h1><a href="<?php echo url_for('job/index') ?>">
            <img src="/images/logo.jpg" alt="Jobeet Job Board" />
          </a></h1>
          <div id="sub_header">
            <div class="post">
              <h2>Ask for people</h2>
              <div>
                <a href="<?php echo url_for('job/index') ?>">Post a Job</a>
              </div>
            </div>
            <div class="search">
              <h2>Ask for a job</h2>
              <form action="" method="get">
                <input type="text" name="keywords"
```

```
            id="search_keywords" />
           <input type="submit" value="search" />
           <div class="help">
            Enter some keywords (city, country, position, ...)
           </div>
          </form>
         </div>
        </div>
       </div>
      </div>
      <div id="content">
       <?php if ($sf_user->hasFlash('notice')): ?>
        <div class="flash_notice">
         <?php echo $sf_user->getFlash('notice') ?>
        </div>
       <?php endif ?>
       <?php if ($sf_user->hasFlash('error')): ?>
        <div class="flash_error">
         <?php echo $sf_user->getFlash('error') ?>
        </div>
       <?php endif ?>
       <div class="content">
        <?php echo $sf_content ?>
       </div>
      </div>
      <div id="footer">
       <div class="content">
        <span class="symfony">
         <img src="/images/jobeet-mini.png" />
         powered by <a href="http://www.symfony-project.org/">
         <img src="/images/symfony.gif" alt="symfony framework" />
         </a>
        </span>
        <ul>
         <li><a href="">About Jobeet</a></li>
         <li class="feed"><a href="">Full feed</a></li>
         <li><a href="">Jobeet API</a></li>
         <li class="last"><a href="">Affiliates</a></li>
        </ul>
       </div>
      </div>
     </div>
    </body>
   </html>
```

Εάν ανοίξουμε τώρα το (http://www.jobeet.com.localhost/frontend_dev.php/job) θα δούμε ότι όλες οι ενέργειες (actions) έχουν διακοσμηθεί από το layout.

5.2 Τα Stylesheet, οι εικόνες και τα JavaScript

Η function *include_stylesheets()* παράγει <link> tags για το stylesheet και ονομάζεται βοηθός (helper). Ένας helper μπορεί να δεχτεί παραμέτρους και να επιστρέψει HTML κώδικα.



Εικόνα 16: Τα Stylesheet του jobeet.

Το επίπεδο View (View Layer) μπορεί να ρυθμιστεί με την επεξεργασία του *view.yml* αρχείου της εφαρμογής. Εδώ είναι το προεπιλεγμένο που έχει παραχθεί από την διεργασία *generate:app*:

```
# apps/frontend/config/view.yml
default:
  http_metas:
    content-type: text/html
  metas:
    #title:        symfony project
```

```
  #description:  symfony project
  #keywords:     symfony, project
  #language:     en

  #robots:       index, follow

 stylesheets:    [main.css]

 javascripts:    []

 has_layout:     true

 layout:         layout
```

5.3 Η αρχική σελίδα του Job

Η αρχική σελίδα της θέσης εργασίας παράγεται από τη δράση του δείκτη (index action) της ενότητας θέση εργασίας. Η δράση του δείκτη είναι μέρος του Controller της σελίδας και το συνδεδεμένο πρότυπο *indexSuccess.php*, είναι το μέρος του View:

```
apps/
  frontend/
    modules/
      job/
        actions/
          actions.class.php
        templates/
          indexSuccess.php
```

5.4 Η δράση (action)

Κάθε δράση αντιπροσωπεύεται από μία μέθοδο μιας κλάσης. Για την αρχική σελίδα της θέσης εργασίας αυτή η κλάση είναι η *jobActions* και η μέθοδός της είναι η *executeIndex()*. Γίνεται ανάκτηση όλων των θέσεων εργασίας από τη βάση δεδομένων:

```
// apps/frontend/modules/job/actions/actions.class.php
class jobActions extends sfActions
{
  public function executeIndex(sfWebRequest $request)
  {
    $this->jobeet_jobs = Doctrine::getTable('JobeetJob')
      ->createQuery('a')
      ->execute();
  }
   // ...
}
```

Η μέθοδος *executeIndex()* καλεί το Table JobeetJob για να δημιουργήσει ένα ερώτημα για να επιστρέψει όλες τις θέσεις εργασίας. Επιστρέφει μία *Doctrine_Collection* των αντικειμένων JobeetJob τα οποία έχουν ανατεθεί στην ιδιοκτησία του object *jobeet_jobs*. Αυτού του είδους οι ιδιότητες των αντικειμένων περνάνε αυτόματα στο πρότυπο (το View). Για να περάσουμε δεδομένα από τον Controller στο View, απλά δημιουργούμε μία νέα ιδιότητα:

```
public function executeFooBar(sfWebRequest $request)

{
  $this->foo = 'bar';
  $this->bar = array('bar', 'baz');
}
```

Ο κώδικας θα κάνει τις μεταβλητές *$foo* και *$bar* διαθέσιμες στο πρότυπο.

5.5 Το πρότυπο (template)

Το πρότυπο *indexSuccess.php* παράγει ένα HTML table για όλες τις θέσεις εργασίας. Εδώ είναι ο κώδικας του προτύπου:

```
<!-- apps/frontend/modules/job/templates/indexSuccess.php -->
<?php use_stylesheet('jobs.css') ?>
<h1>Job List</h1>
<table>
  <thead>
    <tr>
      <th>Id</th>
      <th>Category</th>
      <th>Type</th>
<!-- more columns here -->
      <th>Created at</th>
      <th>Updated at</th>
    </tr>
  </thead>
  <tbody>
    <?php foreach ($jobeet_jobs as $jobeet_job): ?>
    <tr>
      <td>
        <a href="<?php echo url_for('job/show?id='.$jobeet_job->getId()) ?>">
          <?php echo $jobeet_job->getId() ?>
        </a>
      </td>
      <td><?php echo $jobeet_job->getCategoryId() ?></td>
      <td><?php echo $jobeet_job->getType() ?></td>
<!-- more columns here -->
      <td><?php echo $jobeet_job->getCreatedAt() ?></td>
      <td><?php echo $jobeet_job->getUpdatedAt() ?></td>
    </tr>
    <?php endforeach ?>
  </tbody>
</table>
<a href="<?php echo url_for('job/new') ?>">New</a>
```

Στον κώδικα του προτύπου, το κάθε στοιχείο επαναλαμβάνεται μέσα από τη λίστα των αντικειμένων Job (*$jobeet_jobs*), και για κάθε θέση εργασίας, κάθε τιμή μιας στήλης γίνεται έξοδος. Η πρόσβαση στην τιμή μιας στήλης γίνεται απλά με το να καλούμε μία μέθοδο εκτίμησης της οποίας το όνομα ξεκινάει με το *get* και το όνομα της στήλης *camelCased*.

Ας το ξεκαθαρίσουμε αυτό με το να προβάλουμε μόνο ένα υποσύνολο από τις διαθέσιμες στήλες:

```
<!-- apps/frontend/modules/job/templates/indexSuccess.php -->
<?php use_stylesheet('jobs.css') ?>
<div id="jobs">
  <table class="jobs">
    <?php foreach ($jobeet_jobs as $i => $job): ?>
      <tr class="<?php echo fmod($i, 2) ? 'even' : 'odd' ?>">
        <td class="location"><?php echo $job->getLocation() ?></td>
        <td class="position">
          <a href="<?php echo url_for('job/show?id='.$job->getId()) ?>">
            <?php echo $job->getPosition() ?>
          </a>
        </td>
        <td class="company"><?php echo $job->getCompany() ?></td>
      </tr>
    <?php endforeach ?>
  </table>
</div>
```

5.6 Το πρότυπο της σελίδας της θέσης εργασίας

Ας προσαρμόσουμε το πρότυπο της σελίδας της θέσης εργασίας. Ανοίγουμε το αρχείο *showSuccess.php* και αντικαθιστούμε το περιεχόμενο του με τον παρακάτω κώδικα:

```
<!-- apps/frontend/modules/job/templates/showSuccess.php -->
<?php use_stylesheet('job.css') ?>
<?php use_helper('Text') ?>

<div id="job">
  <h1><?php echo $job->getCompany() ?></h1>
  <h2><?php echo $job->getLocation() ?></h2>
  <h3>
    <?php echo $job->getPosition() ?>
    <small> - <?php echo $job->getType() ?></small>
  </h3>

  <?php if ($job->getLogo()): ?>
    <div class="logo">
      <a href="<?php echo $job->getUrl() ?>">
        <img src="/uploads/jobs/<?php echo $job->getLogo() ?>"
          alt="<?php echo $job->getCompany() ?> logo" />
      </a>
```

```
    </div>
  <?php endif ?>

  <div class="description">
    <?php echo simple_format_text($job->getDescription()) ?>
  </div>

  <h4>How to apply?</h4>

  <p class="how_to_apply"><?php echo $job->getHowToApply() ?></p>
  <div class="meta">
    <small>posted on <?php echo $job->getDateTimeObject('created_at')->format('m/d/Y')
?></small>
  </div>
   <div style="padding: 20px 0">
    <a href="<?php echo url_for('job/edit?id='.$job->getId()) ?>">
     Edit
    </a>
  </div>
</div>
```

Αυτό το πρότυπο χρησιμοποιεί την μεταβλητή *$job* που περάστηκε από τη δράση για να προβάλει τις πληροφορίες για τη θέση εργασίας. Καθώς έχουμε μετονομάσει την μεταβλητή που περάστηκε στο πρότυπο από το *$jobeet_job* στο *$job*, χρειάζεται επίσης να γίνει και αυτή η αλλαγή στην δράση προβολής:

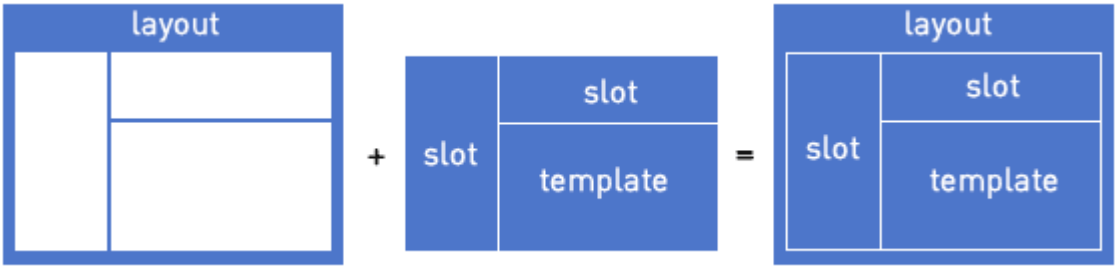
```
// apps/frontend/modules/job/actions/actions.class.php
public function executeShow(sfWebRequest $request)
{
  $this->job = Doctrine::getTable('JobeetJob')-> find($request->getParameter('id'));
  $this->forward404Unless($this->job);
}
```

Παρατηρούμε ότι οι στήλες της ημερομηνίας μπορούν να μετατραπούν σε υποδείξεις αντικειμένων PHP *DateTime*. Καθώς έχουμε καθορίσει τη λίστα *created_at* σαν timestamp, μπορούμε να μετατρέψουμε την τιμή της στήλης σε ένα αντικείμενο *DateTime* χρησιμοποιώντας τη μέθοδο *getDateTimeObject()* και στη συνέχεια να καλέσουμε τη μέθοδο *format()* η οποία παίρνει ένα διαμορφωμένο πρότυπο ημερομηνίας σαν πρώτο της επιχείρημα (argument):

```
$job->getDateTimeObject('created_at')->format('m/d/Y');
```

5.7 Slots

Ο τίτλος όλων των σελίδων καθορίζονται στην ετικέτα *<title>* του layout:



Εικόνα 17: Το Slots

<title>Jobeet – Your best job board</title>

Αλλά για τη σελίδα της θέσης εργασίας, θέλουμε να παρέχουμε πιο χρήσιμες πληροφορίες, όπως το όνομα της εταιρείας και τη θέση εργασίας. Στο symfony, όταν μία ζώνη του layout εξαρτάται από το πρότυπο που πρέπει να προβληθεί, πρέπει να καθορίσουμε ένα slot:

Προσθέτουμε ένα slot στο layout για να επιτρέψουμε να γίνει δυναμικός ο τίτλος:

// apps/frontend/templates/layout.php

<title><?php include_slot('title') ?></title>

Κάθε slot καθορίζεται από το όνομα (τίτλος) και μπορεί να προβληθεί χρησιμοποιώντας τον helper *include_slot()*. Τώρα, στην αρχή του προτύπου *showSuccess.php*, χρησιμοποιούμε τον helper *slot()* για να καθορίσουμε το περιεχόμενο του slot για την σελίδα εργασίας:

// apps/frontend/modules/job/templates/showSuccess.php

```
<?php slot(
  'title',
  sprintf('%s is looking for a %s', $job->getCompany(), $job->getPosition()))
?>
```

Εάν ο τίτλος είναι περίπλοκος για να παραχθεί, μπορεί επίσης να χρησιμοποιηθεί ο helper *slot()* με ένα μπλοκ από κώδικα:

```
// apps/frontend/modules/job/templates/showSuccess.php

<?php slot('title') ?>

  <?php echo sprintf('%s is looking for a %s', $job->getCompany(), $job->getPosition()) ?>

<?php end_slot() ?>
```

Για κάποιες σελίδες, όπως για παράδειγμα την αρχική σελίδα, χρειαζόμαστε απλά έναν γενικό τίτλο. Αντί να επαναλαμβάνουμε τον ίδιο τίτλο ξανά και ξανά σε πρότυπα, μπορούμε να ορίσουμε έναν προεπιλεγμένο τίτλο στο layout:

```
// apps/frontend/templates/layout.php

<title>

  <?php include_slot('title', 'Jobeet - Your best job board') ?>

</title>
```

Το δεύτερο επιχείρημα της μεθόδου *include_slot()* είναι η προεπιλεγμένη τιμή για το slot εάν δεν έχει προκαθοριστεί. Εάν η προεπιλεγμένη τιμή είναι μακρύτερη ή έχει κάποιες ετικέτες HTML (HTML tags), μπορούμε επίσης να το ορίσουμε με τον παρακάτω κώδικα:

```
// apps/frontend/templates/layout.php

<title>

  <?php if (!include_slot('title')): ?>

    Jobeet - Your best job board

  <?php endif ?>

</title>
```

Ο helper *include_slot()* επιστρέφει *true* εάν το slot έχει οριστεί. Έτσι, όταν ορίζουμε το περιεχόμενο του slot *title* σε ένα πρότυπο, χρησιμοποιείται αυτό. Εάν όχι, τότε χρησιμοποιείται ο προκαθορισμένος τίτλος.

5.8 Η Δράση Της Σελίδας Θέσης Εργασίας

Η σελίδα της εργασίας παράγεται από την δράση *show* (*show* action), η οποία ορίζεται στη μέθοδο *executeShow()* της ενότητας της θέσης εργασίας (job module):

```
class jobActions extends sfActions
{
  public function executeShow(sfWebRequest $request)
  {
    $this->job = Doctrine::getTable('JobeetJob')-> find($request->getParameter('id'));
    $this->forward404Unless($this->job);
  }
  // ...
}
```

Όπως και στη δράση του *index* (*index* action), η κλάση του *JobeetJob* table χρησιμοποιείται για την ανάκτηση μιας θέσης εργασίας, χρησιμοποιώντας την μέθοδο *find()*. Η παράμετρος αυτής της μεθόδου, το μοναδικό αναγνωριστικό για την εργασία, είναι το πρωτεύον της κλειδί.

Εάν η θέση εργασίας δεν υπάρχει στην βάση δεδομένων, θέλουμε να προωθήσουμε τον χρήστη σε μία σελίδα 404, το οποίο είναι ακριβώς αυτό που κάνει η μέθοδος *forward404unless()*. Παίρνει μία Boolean σαν το πρώτο της επιχείρημα και σταματάει την τρέχουσα ροή εκτελέσεων, εκτός αν είναι *true*. Καθώς οι προωθητικές μέθοδοι σταματάνε την εκτέλεση μιας δράσης αμέσως, δίνοντας ένα *sfError404Exception*, δεν χρειάζεται να εμείς να επιστρέψουμε παρ' όλα αυτά.

Για τις εξαιρέσεις, η σελίδα που προβάλλεται στον χρήστη είναι διαφορετική στο *prod environment* και στο *dev environment*:

5.9 Το αίτημα και η απάντηση

Όταν περιηγηθούμε στις σελίδες */job* ή */job/show/id/1* μέσα από τον browser μας, ξεκινούμε ένα ταξίδι με τον web server. Ο περιηγητής αποστέλλει ένα αίτημα και ο server στέλνει πίσω μία απάντηση.

Το symfony ενθυλακώνει το αίτημα σε ένα αντικείμενο *sfWebRequest* και καθώς το symfony είναι ένα αντικειμενοστραφή framework η απάντηση είναι επίσης αντικείμενο, της κλάσης *sfWebResponse*. Μπορούμε να έχουμε πρόσβαση σε ένα αντικείμενο απάντησης σε μία δράση καλώντας το *$this->getResponse()*.

Αυτά τα αντικείμενα παρέχουν πολλές κατάλληλες μεθόδους για να προσπελάσουμε πληροφορίες από PHP functions και PHP καθολικές μεταβλητές.

5.9.1 Το αίτημα (Request)

Η κλάση *sfWebRequest* καλύπτει τις καθολικές συστοιχίες *$_SERVER*, *$_COOKIE*, *$_GET*, *$_POST*, και *$_FILES PHP*:

Method name	PHP equivalent
getMethod():	$_SERVER['REQUEST_METHOD']
getUri():	$_SERVER['REQUEST_URI']
getReferer():	$_SERVER['HTTP_REFERER']
getHost():	$_SERVER['HTTP_HOST']
getLanguages():	$_SERVER['HTTP_ACCEPT_LANGUAGE']
getCharsets():	$_SERVER['HTTP_ACCEPT_CHARSET']
isXmlHttpRequest():	$_SERVER['X_REQUESTED_WITH'] == 'XMLHttpRequest'
getHttpHeader():	$_SERVER
getCookie():	$_COOKIE
isSecure():	$_SERVER['HTTPS']
getFiles():	$_FILES
getGetParameter():	$_GET
getPostParameter():	$_POST
getUrlParameter():	$_SERVER['PATH_INFO']
getRemoteAddress():	$_SERVER['REMOTE_ADDR']

5.9.2 Η απάντηση (Response)

Η κλάση *sfWebResponse* καλύπτει τις μεθόδους *header()* και *setrawcookie() PHP*:

Method name	PHP equivalent
setCookie()	setrawcookie()
setStatusCode()	header()
setHttpHeader()	header()
setContentType()	header()
addVaryHttpHeader()	header()
addCacheControlHttpHeader()	header()

6 Η δρομολόγηση (Routing)

6.1 URLS

Σε ένα πλαίσιο web, ένα URL είναι ένα μοναδικό προσδιοριστικό μιας πηγής web. Όταν πηγαίνουμε σε ένα URL, ζητάμε από τον browser να φέρει μία πηγή που προσδιορίζεται από αυτό το URL. Καθώς το URL είναι η διεπαφή μεταξύ της ιστοσελίδας και του χρήστη, θα πρέπει να μεταβιβάσει σημαντικές πληροφορίες για την πηγή στην οποία αναφέρεται.

Τα URL είναι τόσο σημαντικά στο symfony όπου έχει ένα ολόκληρο framework αφιερωμένο στην διαχείρισή τους: το routing framework. Το routing διαχειρίζεται εσωτερικά URIs και εξωτερικά URLs. Όταν έρχεται ένα αίτημα, το routing αναλύει το URL και το μετατρέπει σε ένα εσωτερικό URI.

Εσωτερικό URI της σελίδας εργασίας στο πρότυπο *indexSuccess.php*:

'job/show?id='.$job->getId()

Ο helper *url_for()* μετατρέπει αυτό το εσωτερικό URI σε ένα κατάλληλο URL:

/job/show/id/1

Το εσωτερικό URI αποτελείται από διάφορα μέρη: το *job* είναι το module, το *show* είναι το action και το *query string* προσθέτει παραμέτρους για να περάσει στην δράση. Το γενικό πρότυπο για τα εσωτερικά URI είναι:

MODULE/ACTION?key=value&key_1=value_1&...

Καθώς το routing του symfony είναι μια αμφίδρομη διεργασία, μπορούμε να αλλάξουμε τα URLs χωρίς να αλλάξουμε την τεχνική εκτέλεση. Αυτό είναι ένα από τα κύρια πλεονεκτήματα το πρότυπο σχέδιο του front-controller.

6.2 Ρύθμιση παραμέτρων του Routing

Η χαρτογράφηση μεταξύ των εσωτερικών URI και των εξωτερικών URL γίνεται από το αρχείο ρύθμισης παραμέτρων *routing.yml*:

```
# apps/frontend/config/routing.yml

homepage:

  url:   /

  param: { module: default, action: index }

default_index:

  url:   /:module

  param: { action: index }

default:

  url:   /:module/:action/*
```

Το αρχείο *routing.yml* περιγράφει διαδρομές. Μία διαδρομή έχει ένα όνομα, ένα υπόδειγμα, και κάποιες παραμέτρους.

Όταν εισέρχεται ένα αίτημα, το routing επιχειρεί να ταιριάξει ένα υπόδειγμα σε ένα δοθέν URL. Η πρώτη διαδρομή που ταιριάζει κερδίζει, οπότε η σειρά μέσα στο *routing.yml* είναι σημαντική.

Για παράδειγμα εάν στείλουμε αίτημα για την σελίδα */job/show/id/1*, το symfony θα ταιριάξει το τελευταίο υπόδειγμα: */:module/:action/**. Στο υπόδειγμα, ο αστερίσκος (*) ταιριάζει μία συλλογή από ζεύγη μεταβλητών και τιμών που διαχωρίζονται με πλάγιους (slashes /):

Request parameter	Value
Module:	job
Action:	show
Id:	1

Το URL */job/show/id/1* μπορεί να δημιουργηθεί από ένα πρότυπο χρησιμοποιώντας την παρακάτω κλήση στο *url_for()helper*:

```
url_for('job/show?id='.$job->getId())
```

Μπορούμε επίσης να χρησιμοποιήσουμε το όνομα της διαδρομής βάζοντας μπροστά το σύμβολο @:

```
url_for('@default?module=job&action-show&id='.$job->getId())
```

6.3 Προσαρμογές της διαδρομής (Route Customizations)

Για να αλλάξουμε τη προεπιλεγμένη σελίδα συγχαρητηρίων του symfony στην αρχική σελίδα του *Jobeet* τροποποιούμε την μεταβλητή *module* της διαδρομής της αρχικής σελίδας σε εργασία:

```
# apps/frontend/config/routing.yml

homepage:

  url:   /

  param: { module: job, action: index }
```

Τώρα μπορούμε να αλλάξουμε τον σύνδεσμο του λογοτύπου του Jobeet στο layout για να χρησιμοποιήσει τη διαδρομή της αρχικής σελίδας:

```
<!-- apps/frontend/templates/layout.php -->

<h1>

  <a href="<?php echo url_for('homepage') ?>">

    <img src="/images/logo.jpg" alt="Jobeet Job Board" />

  </a>

</h1>
```

Ας αλλάξουμε το URL της σελίδας εργασίας σε κάτι πιο σημαντικό:

```
/job/sensio-labs/paris-france/1/web-developer
```

Το παρακάτω υπόδειγμα ταιριάζει σε ένα τέτοιο URL:

```
/job/:company/:location/:id/:position
```

Επεξεργαζόμαστε το αρχείο *routing.yml* και προσθέτουμε τη διαδρομή *job_show_user* στην αρχή του αρχείου:

```
job_show_user:

  url:   /job/:company/:location/:id/:position

  param: { module: job, action: show }
```

Εάν ανανεώσουμε την αρχική σελίδα του *Jobeet* παρατηρούμε ότι οι σύνδεσμοι στις εργασίες δεν έχουν αλλάξει. Αυτό συμβαίνει διότι για να παράγουμε μία διαδρομή, πρέπει να περάσουμε όλες τις απαιτούμενες μεταβλητές. Οπότε, πρέπει να αλλάξουμε την κλήση *url_for()* μέσα στο *indexSyccess.php* σε:

```
url_for('job/show?id='.$job->getId().'&company='.$job->getCompany().

  '&location='.$job->getLocation().'&position='.$job->getPosition())
```

Ένα εσωτερικό URI μπορεί επίσης να εκφραστεί σαν μια συστοιχία:

```
url_for(array(

  'module'   => 'job',

  'action'   => 'show',

  'id'       => $job->getId(),

  'company'  => $job->getCompany(),

  'location' => $job->getLocation(),

  'position' => $job->getPosition(),

))
```

6.4 Απαιτήσεις

Το routing system έχει ένα ενσωματωμένο χαρακτηριστικό επικύρωσης. Κάθε μεταβλητή υποδείγματος μπορεί να επικυρωθεί με μία κανονική έκφραση η οποία ορίζεται χρησιμοποιώντας την είσοδο απαιτήσεων ενός ορισμού διαδρομής:

```
job_show_user:

  url:   /job/:company/:location/:id/:position

  param: { module: job, action: show }

  requirements:

    id: \d+
```

6.5 Route Class

Κάθε διαδρομή που έχει οριστεί στο *routing.yml* έχει μετατραπεί εσωτερικά σε αντικείμενο της κλάσης *sfRoute*. Αυτή η κλάση μπορεί να αλλαχθεί ορίζοντας μία είσοδο κλάσης (class entry) στον ορισμό της διαδρομής.

Για να περιορίσουμε μία διαδρομή να ταιριάξει για συγκεκριμένες μεθόδους αίτησης, μπορούμε να αλλάξουμε την κλάση διαδρομή (route class) σε *sfRequestRoute* και να προσθέσουμε μία απαίτηση για την εικονική μεταβλητή (virtual variable) *sf_method*:

```
job_show_user:
  url:   /job/:company/:location/:id/:position
  class: sfRequestRoute
  param: { module: job, action: show }
  requirements:
    id: \d+
    sf_method: [get]
```

7.6 Object Route Class

Το νέο εσωτερικό URI για μια εργασία είναι αρκετά μακρύ και κουραστικό για να γραφεί, αλλά όπως είδαμε, η route class μπορεί να αλλαχθεί. Για τη διαδρομή *job_show_user*, είναι καλύτερο να χρησιμοποιήσουμε σαν κλάση την *sfDoctrineRoute* καθώς είναι βελτιστοποιημένη για διαδρομές που αντιπροσωπεύουν αντικείμενα Doctrine ή συλλογές από αντικείμενα Doctrine:

```
job_show_user:
  url:    /job/:company/:location/:id/:position
  class:  sfDoctrineRoute
  options: { model: JobeetJob, type: object }
  param:  { module: job, action: show }
  requirements:
    id: \d+
    sf_method: [get]
```

Η καταχώρηση options προσαρμόζει την συμπεριφορά της διαδρομής. Εδώ, η επιλογή model καθορίζει την model class (*JobeetJob*) που σχετίζεται με τη διαδρομή, και η επιλογή type καθορίζει ότι αυτή η διαδρομή είναι "δεμένη" σε ένα αντικείμενο (object).

Η διαδρομή *job_show_user* γνωρίζει τη σχέση της με το *JobeetJob* οπότε και μπορούμε να απλοποιήσουμε την κλήση του helper *url_for()|url_for* σε:

url_for(array('sf_route' => 'job_show_user', 'sf_subject' => $job))

ή απλά:

url_for('job_show_user'. $job)

Αν κοιτάξουμε στα URL που έχουν παραχθεί, ακόμα δεν έχουν τη μορφή που επιθυμούμε:

http://www.jobeet.com.localhost/frontend_dev.php/job/Sensio+Labs/Paris%2C+France/1/Web+Developer

Πρέπει να αλλάξουμε τις τιμές των στηλών αντικαθιστώντας όλους τους μη ASCII χαρακτήρες με ένα [-]. Ανοίγουμε το αρχείο *JobeetJob* και προσθέτουμε τις παρακάτω μεθόδους στην κλάση:

```
// lib/model/doctrine/JobeetJob.class.php

public function getCompanySlug()

{

  return Jobeet::slugify($this->getCompany());

}

 public function getPositionSlug()

{

  return Jobeet::slugify($this->getPosition());

}

public function getLocationSlug()

{

  return Jobeet::slugify($this->getLocation());

}
```

Στη συνέχεια δημιουργούμε το αρχείο *lib/Jobeet.class.php* και προσθέτουμε την ίδια μέθοδο:

```
// lib/Jobeet.class.php

class Jobeet

{
```

```
  static public function slugify($text)
  {
    // replace all non letters or digits by -
    $text = preg_replace('/\W+/', '-', $text);
    // trim and lowercase
    $text = strtolower(trim($text, '-'));
    return $text;
  }
}
```

Έχουμε ορίσει τρεις νέους "εικονικούς" accessors: τους *getCompanySlug()*, *getPositionSlug()*, και *getLocationSlug()*. Οι accessors αυτοί επιστρέφουν την αντίστοιχη τιμή της στήλης μετά την εφαρμογή της μεθόδου *slugify()*. Τώρα μπορούμε να αντικαταστήσουμε τα πραγματικά ονόματα των στηλών με αυτά των εικονικών στη διαδρομή *job_show_user*:

```
job_show_user:
  url:    /job/:company_slug/:location_slug/:id/:position_slug
  class:  sfDoctrineRoute
  options: { model: JobeetJob, type: object }
  param:  { module: job, action: show }
  requirements:
    id: \d+
    sf_method: [get]
```

Πλέον έχουμε τα αναμενόμενα URL:

http://www.jobeet.com.localhost/frontend_dev.php/job/sensio-labs/paris-france/1/web-developer

Η διαδρομή είναι ικανή να παράξει ένα URL βασισμένο σε ένα αντικείμενο, αλλά επίσης μπορεί να βρει και το αντικείμενο που σχετίζεται σε ένα δοσμένο URL. Το συσχετισμένο αντικείμενο μπορεί να ανακτηθεί με τη μέθοδο *getObject()* του route object. Όταν αναλύει ένα εισερχόμενο αίτημα, το routing αποθηκεύει το "ταιριαστό" route object για να χρησιμοποιηθεί στις actions. Οπότε, αλλάζουμε τη μέθοδο *executeShow()* για να χρησιμοποιήσει το route object να ανακτήσει το Jobeet object:

```
class jobActions extends sfActions
{
  public function executeShow(sfWebRequest $request)
  {
    $this->job = $this->getRoute()->getObject();
    $this->forward404Unless($this->job);
  }
  // ...
```

Εάν προσπαθήσουμε να πάρουμε μία εργασία με άγνωστο *id*, θα δούμε μία σελίδα σφάλματος 404, αλλά το μήνυμα σφάλματος θα έχει αλλάξει:

Αυτό συμβαίνει διότι το σφάλμα 404 έχει ριχτεί αυτόματα από τη μέθοδο *getRoute()*. Οπότε μπορούμε να απλοποιήσουμε τη μέθοδο *executeShow* ακόμα περισσότερο:

```
class jobActions extends sfActions
{
  public function executeShow(sfWebRequest $request)
  {
    $this->job = $this->getRoute()->getObject();
  }
  // ...
}
```

6.7 Routing σε Actions και σε Templates

Σε ένα πρότυπο (template), ο helper *url_for()* μετατρέπει ένα εσωτερικό URI σε ένα εξωτερικό URL. Κάποιοι άλλοι helpers του symfony επίσης παίρνουν ένα εσωτερικό URI σαν argument, όπως ο helper *link_to()* ο οποίος παράγει μία ετικέτα <a>:

```
<?php echo link_to($job->getPosition(), 'job_show_user', $job) ?>
```

Παράγει τον παρακάτω κώδικα HTML:

```
<a href="/job/sensio-labs/paris-france/1/web-developer">Web Developer</a>
```

Επίσης τα *url_for()* και *link_to()* μπορούν επίσης να παράγουν απόλυτα URL:

```
url_for('job_show_user', $job, true);

link_to($job->getPosition(), 'job_show_user', $job, true);
```

Εάν θελήσουμε να παραχθεί ένα URL από μία action, μπορούμε να χρησιμοποιήσουμε τη μέθοδο *generateURL()*:

```
$this->redirect($this->generateUrl('job_show_user', $job));
```

6.8 Συλλογή Route Class

Για το *job module*, έχουμε ήδη προσαρμόσει την *route show action*, αλλά τα URLs για τις άλλες μεθόδους (index, new, edit, create, update και delete) διαχειρίζονται ακόμα από την προεπιλεγμένη route:

```
default:

  url: /:module/:action/*
```

Καθώς όλες οι *job actions* συσχετίζονται με την *model class JobeetJob*, μπορούμε εύκολα να ορίσουμε μία route *sfDoctrineRoute* για την κάθε μία όπως έχουμε ήδη κάνει για την *show action*. Καθώς όμως το *job module* ορίζει τις κλασσικές εφτά πιθανές actions για ένα model, μπορούμε επίσης να χρησιμοποιήσουμε την class *sfDoctrineRouteCollection*. Ανοίγουμε το αρχείο *routing.yml* και το τροποποιούμε ώστε να δείχνει ως εξής:

```
# apps/frontend/config/routing.yml

job:

  class:   sfDoctrineRouteCollection

  options: { model: JobeetJob }

job_show_user:

  url:     /job/:company_slug/:location_slug/:id/:position_slug

  class:   sfDoctrineRoute

  options: { model: JobeetJob, type: object }

  param:   { module: job, action: show }
```

```
  requirements:
    id: \d+
    sf_method: [get]
  # default rules
  homepage:
    url:   /
    param: { module: job, action: index }
  default_index:
    url:   /:module
    param: { action: index }
  default:
    url:   /:module/:action/*
```

Η job route παραπάνω, είναι μία συντόμευση που παράγει αυτόματα τις παρακάτω εφτά *sfDoctrineRoute* routes:

```
  job:
    url:     /job.:sf_format
    class:   sfDoctrineRoute
    options: { model: JobeetJob, type: list }
    param:   { module: job, action: index, sf_format: html }
    requirements: { sf_method: get }

  job_new:
    url:     /job/new.:sf_format
    class:   sfDoctrineRoute
    options: { model: JobeetJob, type: object }
    param:   { module: job, action: new, sf_format: html }
```

requirements: { sf_method: get }

job_create:

url: /job.:sf_format

class: sfDoctrineRoute

options: { model: JobeetJob, type: object }

param: { module: job, action: create, sf_format: html }

requirements: { sf_method: post }

job_edit:

url: /job/:id/edit.:sf_format

class: sfDoctrineRoute

options: { model: JobeetJob, type: object }

param: { module: job, action: edit, sf_format: html }

requirements: { sf_method: get }

job_update:

url: /job/:id.:sf_format

class: sfDoctrineRoute

options: { model: JobeetJob, type: object }

param: { module: job, action: update, sf_format: html }

requirements: { sf_method: put }

job_delete:

url: /job/:id.:sf_format

class: sfDoctrineRoute

options: { model: JobeetJob, type: object }

```
  param:   { module: job, action: delete, sf_format: html }

  requirements: { sf_method: delete }
```

job_show:

```
  url:     /job/:id.:sf_format

  class:   sfDoctrineRoute

  options: { model: JobeetJob, type: object }

  param:   { module: job, action: show, sf_format: html }

  requirements: { sf_method: get }
```

Οι routes *job_delete* και *job_update* απαιτούν μεθόδους HTTP οι οποίες δεν υποστηρίζονται από τους browsers. Λειτουργεί όμως διότι το symfony τις εξομοιώνει. Ανοίγουμε το πρότυπο *the_form.php* για να δούμε ένα παράδειγμα:

```
// apps/frontend/modules/job/templates/_form.php

<form action="..." ...>

<?php if (!$form->getObject()->isNew()): ?>

  <input type="hidden" name="sf_method" value="PUT" />

<?php endif; ?>

<?php echo link_to(

  'Delete',

  'job/delete?id='.$form->getObject()->getId(),

  array('method' => 'delete', 'confirm' => 'Are you sure?')

) ?>
```

Μπορούμε να "πούμε" σε όλους τους symfony helpers να εξομοιώσουν οποιαδήποτε μέθοδο HTTP που θέλουμε περνώντας την ειδική παράμετρο *sf_method*.

6.9 Αποσφαλμάτωση Route (Debugging)

Όταν χρησιμοποιούμε collection routes, είναι χρήσιμο να καταγράφουμε τις routes που έχουν παραχθεί. Η διεργασία *app:routes* εξάγει όλες τις routes για μια συγκεκριμένη εφαρμογή:

```
$ php symfony app:routes frontend
```

Επίσης μπορούμε να έχουμε αρκετές πληροφορίες αποσφαλμάτωσης για μία route περνώντας το όνομα της σαν ένα επιπρόσθετο argument:

$ php symfony app:routes frontend job_edit

6.10 Προεπιλεγμένα Routes

Μια καλή πρακτική είναι το να ορίσουμε διαδρομές για όλα τα URLs μας. Καθώς η job route ορίζει όλες τις διαδρομές που χρειάζονται για να περιγράψουν την εφαρμογή Jobeet, μπορούμε να αφαιρέσουμε, ή να ορίσουμε σαν σχόλια, τις προεπιλεγμένες διαδρομές στο αρχείο ρυθμίσεως παραμέτρων *routing.yml*:

```
# apps/frontend/config/routing.yml

#default_index:

#  url:   /:module

#  param: { action: index }

#default:

#  url:   /:module/:action/*
```

7 To Doctrine Query Object

Ενώ παραπάνω είχαμε πει στις απαιτήσεις πως όταν ένας χρήστης εισέρχεται στο site θα βλέπει μία λίστα μόνο με τις ενεργές εργασίες, ακόμα εμφανίζονται όλες οι εργασίες, είτε είναι ενεργές, είτε όχι:

```
// apps/frontend/modules/job/actions/actions.class.php

class jobActions extends sfActions

{

  public function executeIndex(sfWebRequest $request)

  {

    $this->jobeet_jobs = Doctrine::getTable('JobeetJob')

      ->createQuery('a')

      ->execute();

  }

  // ...

}
```

Μία ενεργή εργασία είναι αυτή που έχει αναρτηθεί σε λιγότερο από 30 ημέρες πριν. Η μέθοδος *~Doctrine_Query~::execute()* θα στείλει ένα αίτημα στην βάση δεδομένων. Στον παραπάνω κώδικα, δεν καθορίζουμε καμία κατάσταση για το που, το οποίο σημαίνει πως όλες οι εγγραφές ανακτούνται από τη βάση δεδομένων.

Ας το αλλάξουμε για να επιλέγουμε μόνο ενεργές εργασίες:

```
public function executeIndex(sfWebRequest $request)
{
  $q = Doctrine_Query::create()
    ->from('JobeetJob j')
    ->where('j.created_at > ?', date('Y-m-d H:i:s', time() - 86400 * 30));
  $this->jobeet_jobs = $q->execute();
}
```

7.1 Αποσφαλματώνοντας SQL που δημιουργήθηκε από Doctrine

Καθώς δεν γράφουμε στο χέρι SQL statements, το Doctrine θα αναλάβει τις διαφορές μεταξύ των μηχανών της βάσης δεδομένων και θα παράγει SQL statements βελτιστοποιημένα για τη μηχανή της βάσης δεδομένων που έχουμε επιλέξει. Είναι συχνά χρήσιμο να βλέπουμε το SQL που έχει παραχθεί από το Doctrine, για παράδειγμα, να κάνουμε αποσφαλμάτωση σε query που δεν λειτουργεί όπως θα έπρεπε. Στο περιβάλλον *dev*, το symfony καταγράφει αυτά τα queries στον φάκελο *log/*. Υπάρχει ένα αρχείο καταγραφής

για κάθε συνδυασμό μιας εφαρμογής κι ένα περιβάλλον. Το αρχείο που θέλουμε ονομάζεται *frontend_dev.log*:

```
# log/frontend_dev.log
Dec 04 13:58:33 symfony [info] {sfDoctrineLogger} executeQuery : SELECT
j.id AS j__id, j.category_id AS j__category_id, j.type AS j__type,
j.company AS j__company, j.logo AS j__logo, j.url AS j__url,
j.position AS j__position, j.location AS j__location,
j.description AS j__description, j.how_to_apply AS j__how_to_apply,
j.token AS j__token, j.is_public AS j__is_public,
j.is_activated AS j__is_activated, j.email AS j__email,
j.expires_at AS j__expires_at, j.created_at AS j__created_at,
```

```
j.updated_at AS j__updated_at FROM jobeet_job j

WHERE j.created_at > ? (2008-11-08 01:13:35)
```

Χάρη στο symfony web debug toolbar, όλες οι πληροφορίες που χρειαζόμαστε είναι επίσης διαθέσιμες μέσα στον browser:

7.2 Object Serialization

Όταν θέλουμε να κάνουμε κάτι αυτόματα πριν ένα Doctrine object σειριοποιηθεί στην βάση δεδομένων, μπορούμε να παρακάμψουμε τη μέθοδο *save()* της model class:

```
// lib/model/doctrine/JobeetJob.class.php

class JobeetJob extends BaseJobeetJob

{

  public function save(Doctrine_Connection $conn = null)

  {

    if ($this->isNew() && !$this->getExpiresAt())

    {

      $now = $this->getCreatedAt() ? $this->getDateTimeObject('created_at')->format('U') : time();

      $this->setExpiresAt(date('Y-m-d H:i:s', $now + 86400 * 30));

    }

    return parent::save($conn);

  }
```

Η μέθοδος *isNew()* επιστρέφει true όταν το αντικείμενο δεν έχει σειριοποιηθεί ακόμα στη βάση δεδομένων, και false διαφορετικά. Ας αλλάξουμε την action για να χρησιμοποιήσουμε τη στήλη *expires_at* αντί για τη *created_at* για να επιλέξουμε τις ενεργές εργασίες:

```
public function executeIndex(sfWebRequest $request)

{

  $q = Doctrine_Query::create()

    ->from('JobeetJob j')

    ->where('j.expires_at > ?', date('Y-m-d H:i:s', time()));
```

```
  $this->jobeet_jobs = $q->execute();
}
```

7.3 Περισσότερα με τα Fixtures

Ανανεώνοντας την αρχική σελίδα του Jobeet στον browser δεν θα αλλάξει τίποτα όσο οι εργασίες στη βάση δεδομένων έχουν αναρτηθεί μόνο λίγες μέρες πριν. Ας αλλάξουμε τα fixtures για να προσθέσουμε μία εργασία η οποία έχει ήδη λήξει:

```
# data/fixtures/jobs.yml

JobeetJob:

  # other jobs

  expired_job:

    JobeetCategory: programming

    company:       Sensio Labs

    position:      Web Developer

    location:      Paris, France

    description:   Lorem ipsum dolor sit amet, consectetur adipisicing elit.

    how_to_apply:  Send your resume to lorem.ipsum [at] dolor.sit

    is_public:     true

    is_activated:  true

    created_at:    '2005-12-01 00:00:00'

    token:         job_expired

    email:         job@example.com
```

Η τιμή της στήλης *created_at* μπορεί να οριστεί ακόμα κι αν έχει συμπληρωθεί αυτόματα από το Doctrine. Η ορισμένη τιμή θα αντικαταστήσει την προεπιλεγμένη. Ξανά φορτώνουμε τα fixtures και ανανεώνουμε τον browser για να σιγουρευτούμε ότι η παλιά εργασία δεν εμφανίζεται:

```
$ php symfony doctrine:data-load
```

Μπορούμε επίσης να εκτελέσουμε το παρακάτω query για να σιγουρευτούμε ότι η στήλη *expires_at* έχει συμπληρωθεί αυτόματα από τη μέθοδο *save()*, βασιζόμενη στην τιμή *created_at*:

```
SELECT 'position', 'created_at', 'expires_at' FROM 'jobeet_job';
```

7.4 Custom Configuration

Στη μέθοδο *JobeetJob::save()*, έχουμε σκληρό-κωδικοποιήσει τον αριθμό των ημερών μέχρι να λήξει η εργασία. Θα ήταν καλύτερο να κάνουμε τις 30 ημέρες διαμορφώσιμες. Το symfony framework παρέχει ένα ενσωματωμένο αρχείο ρυθμίσεως παραμέτρων για συγκεκριμένες ρυθμίσεις της application, το αρχείο *app.yml*. Αυτό το αρχείο YAML μπορεί να περιέχει οποιαδήποτε ρύθμιση θέλουμε:

```
# apps/frontend/config/app.yml

all:

  active_days: 30
```

Στην εφαρμογή, οι ρυθμίσεις αυτές είναι διαθέσιμες μέσα από την καθολική κλάση *sfConfig*:

```
sfConfig::get('app_active_days')
```

Η ρύθμιση έχει προταχθεί από το *app_* διότι η class *sfConfig* παρέχει επίσης πρόσβαση στις ρυθμίσεις του symfony. Ας ενημερώσουμε τον κώδικα για να μπει στον λογαριασμό αυτή η νέα ρύθμιση:

```
public function save(Doctrine_Connection $conn = null)

{

  if ($this->isNew() && !$this->getExpiresAt())

  {

    $now = $this->getCreatedAt() ? $this->getDateTimeObject('created_at')->format('U') : time();

    $this->setExpiresAt(date('Y-m-d H:i:s', $now + 86400 * sfConfig::get('app_active_days')));

  }

  return parent::save($conn);

}
```

7.5 Refactoring

Ο κώδικας *Doctrine_Query* δεν ανήκει στην action (Controller layer), ανήκει στο Model layer. Στο μοντέλο MVC, το Model ορίζει την επιχειρηματική λογική, και ο Controller καλεί μόνο το Model για να αντλήσει δεδομένα από αυτό. Καθώς ο κώδικας επιστρέφει μία συλλογή από εργασίες, ας μεταφέρουμε τον κώδικα στην *JobeetJobTable* class και ας δημιουργήσουμε μία μέθοδο *getActiveJobs()*:

```
// lib/model/doctrine/JobeetJobTable.class.php

class JobeetJobTable extends Doctrine_Table

{

  public function getActiveJobs()

  {

    $q = $this->createQuery('j')

      ->where('j.expires_at > ?', date('Y-m-d H:i:s', time()));

    return $q->execute();

  }

}
```

Τώρα ο action κώδικας μπορεί να χρησιμοποιήσει αυτή τη νέα μέθοδο για να ανακτήσει τις ενεργές εργασίες.

```
public function executeIndex(sfWebRequest $request)

{

  $this->jobeet_jobs = Doctrine_Core::getTable('JobeetJob')->getActiveJobs();

}
```

Το refactoring έχει αρκετά πλεονεκτήματα σε σχέση με τον προηγούμενο κώδικα:

- Η λογική του να λαμβάνει ενεργές εργασίες τώρα υπάρχει στο Model, εκεί που ανήκει.
- Ο κώδικας στον controller είναι λεπτότερος και πιο ευανάγνωστος.
- Η μέθοδος *getActiveJobs()* μπορεί να επαναχρησιμοποιηθεί.
- Ο κώδικας του model μπορεί πλέον να δοκιμαστεί στη μονάδα.

Ας κατανείμουμε τις εργασίες κατά τη στήλη *expires_at*:

```
public function getActiveJobs()
{
  $q = $this->createQuery('j')
    ->where('j.expires_at > ?', date('Y-m-d H:i:s', time()))
    ->orderBy('j.expires_at DESC');
  return $q->execute();
}
```

7.6 Κατηγορίες στην Αρχική Σελίδα

Από τις απαιτήσεις η αρχική σελίδα θα πρέπει να προβάλει τις εργασίες ανά κατηγορία. Πρώτα, χρειάζεται να κάνουμε όλες τις κατηγορίες να έχουν τουλάχιστον μία ενεργή εργασία.

Ανοίγουμε την κλάση *JobeetCategoryTable* και προσθέτουμε τη μέθοδο *getWithJobs()*:

```
// lib/model/doctrine/JobeetCategoryTable.class.php
class JobeetCategoryTable extends Doctrine_Table
{
  public function getWithJobs()
  {
    $q = $this->createQuery('c')
      ->leftJoin('c.JobeetJobs j')
      ->where('j.expires_at > ?', date('Y-m-d H:i:s', time()));
    return $q->execute();
  }
}
```

Αλλάζουμε την *index* action:

```
// apps/frontend/modules/job/actions/actions.class.php
public function executeIndex(sfWebRequest $request)
{
  $this->categories = Doctrine_Core::getTable('JobeetCategory')->getWithJobs();
}
```

Στο template, χρειάζεται να επαναλάβουμε μέσα από όλες τις κατηγορίες και να προβάλουμε τις ενεργές εργασίες:

```
// apps/frontend/modules/job/templates/indexSuccess.php
<?php use_stylesheet('jobs.css') ?>
<div id="jobs">
  <?php foreach ($categories as $category): ?>
    <div class="category_<?php echo Jobeet::slugify($category->getName()) ?>">
      <div class="category">
        <div class="feed">
          <a href="">Feed</a>
        </div>
        <h1><?php echo $category ?></h1>
      </div>
      <table class="jobs">
        <?php foreach ($category->getActiveJobs() as $i => $job): ?>
          <tr class="<?php echo fmod($i, 2) ? 'even' : 'odd' ?>">
            <td class="location">
              <?php echo $job->getLocation() ?>
            </td>
            <td class="position">
```

```
            <?php echo link_to($job->getPosition(), 'job_show_user', $job) ?>
          </td>
          <td class="company">
            <?php echo $job->getCompany() ?>
          </td>
        </tr>
      <?php endforeach; ?>
    </table>
  </div>
<?php endforeach; ?>
</div>
```

Για να λειτουργήσει αυτό, θα πρέπει να προσθέσουμε τη μέθοδο *getActiveJobs()* στην κλάση *JobeetCategory*:

```
// lib/model/doctrine/JobeetCategory.class.php
public function getActiveJobs()
{
  $q = Doctrine_Query::create()
    ->from('JobeetJob j')
    ->where('j.category_id = ?', $this->getId());
  return Doctrine_Core::getTable('JobeetJob')->getActiveJobs($q);
}
```

Η μέθοδος *JobeetCategory::getActiveJobs()*χρησιμοποιεί την μέθοδο *Doctrine_core::getTable('JobeetJob')->getActiveJobs()* για να ανακτήσει της ενεργές εργασίες σε συγκεκριμένη κατηγορία.

Όταν καλούμε την *Doctrine_core::getTable('JobeetJob')->getActiveJobs()*, θέλουμε να περιορίσουμε τη συνθήκη ακόμα περισσότερο παρέχοντας μία κατηγορία. Αντί να περάσουμε το category object, έχουμε επιλέξει να περάσουμε ένα *Doctrine_Query* object καθώς είναι ο καλύτερος τρόπος για να ενθυλακώσουμε μία γενική συνθήκη.

Η *getActiveJobs()* χρειάζεται να ενσωματώσει αυτό το *Doctrine_Query* object με τη δικιά του query. Καθώς το *Doctrine_Query* είναι αντικείμενο, αυτό είναι σχετικά απλό:

```
// lib/model/doctrine/JobeetJobTable.class.php

public function getActiveJobs(Doctrine_Query $q = null)

{

  if (is_null($q))

  {

    $q = Doctrine_Query::create()

      ->from('JobeetJob j');

  }

  $q->andWhere('j.expires_at > ?', date('Y-m-d H:i:s', time()))

    ->addOrderBy('j.expires_at DESC');

  return $q->execute();

}
```

7.7 Περιορίζοντας τα Αποτελέσματα

Για κάθε κατηγορία είπαμε, θέλουμε τη λίστα να δείχνει τις πρώτες δέκα εργασίες κι ένα σύνδεσμο για να επιτρέπει να τις εμφανίζει όλες.

Αυτό είναι αρκετά απλό για να το προσθέσουμε στη μέθοδο *getActiveJobs()*:

```
// lib/model/doctrine/JobeetCategory.class.php

public function getActiveJobs($max = 10)

{

  $q = Doctrine_Query::create()

    ->from('JobeetJob j')

    ->where('j.category_id = ?', $this->getId())
```

```
    ->limit($max);

  return Doctrine_Core::getTable('JobeetJob')->getActiveJobs($q);

}
```

Η κατάλληλη LIMIT ρήτρα είναι πλέον σκληρό-κωδικοποιημένη μέσα στο Model, αλλά είναι καλύτερα αυτή η τιμή να είναι διαμορφώσιμη. Αλλάζουμε το template για να περάσει τον μέγιστο αριθμό από εργασίες στο *app.yml*:

```
<!-- apps/frontend/modules/job/templates/indexSuccess.php -->

<?php foreach ($category->getActiveJobs(sfConfig::get('app_max_jobs_on_homepage')) as $i => $job): ?>
```

και προσθέτουμε μία καινούρια ρύθμιση στο *app.yml*:

```
all:

  active_days:         30

  max_jobs_on_homepage: 10
```

7.8 Δυναμικά Fixtures

Εκτός κι αν κατεβάσουμε τη ρύθμιση *max_jobs_on_homepage* σε ένα, δεν θα δούμε καμία διαφορά. Θα πρέπει να προσθέσουμε ένα σωρό από εργασίες στο fixture. Τα αρχεία YAML μπορούν να περιέχουν PHP κώδικα ο οποίος θα αξιολογηθεί πριν το πέρασμα του αρχείο. Επεξεργαζόμαστε το fixture αρχείο *jobs.yml* και προσθέτουμε τον παρακάτω κώδικα στο τέλος του:

```
# Starts at the beginning of the line (no whitespace before)

<?php for ($i = 100; $i <= 130; $i++): ?>

  job_<?php echo $i ?>:

    JobeetCategory: programming

    company:      Company <?php echo $i."\n" ?>

    position:     Web Developer

    location:     Paris, France

    description:  Lorem ipsum dolor sit amet, consectetur adipisicing elit.

    how_to_apply: |

      Send your resume to lorem.ipsum [at] company_<?php echo $i ?>.sit
```

is_public: true

is_activated: true

token: job_<?php echo $i."\n" ?>

email: job@example.com

<?php endfor ?>

Όταν προσθέτουμε PHP κώδικα σε ένα αρχείο YAML, θα πρέπει να προσέχουμε τα παρακάτω:

- Τα *<?php?>* statements θα πρέπει πάντα να ξεκινάνε τη γραμμή ή να είναι ενσωματωμένα σε μία τιμή.
- Εάν ένα *<?php?>* statement τελειώνει μία γραμμή, θα πρέπει να εξάγουμε μία νέα γραμμή (*"\n"*)

Τώρα μπορούμε να ξανά φορτώσουμε τα fixtures με την διεργασία *doctrine:data-load* και να δούμε αν εμφανίζονται μόνο 10 εργασίες στην αρχική σελίδα για την κατηγορία Programming. Στο παρακάτω screenshot, έχουμε αλλάξει τον μέγιστο αριθμό εργασιών σε πέντε για να κάνουμε την εικόνα μικρότερη:

7.9 Ασφαλίζοντας τη Σελίδα Θέσης Εργασίας

Όταν μία εργασία λήγει, ακόμα κι αν γνωρίζουμε το URL, δεν θα πρέπει να είναι προσβάσιμη.

/frontend_dev.php/job/sensio-labs/paris-france/ID/web-developer-expired

Αντί να προβάλλεται η εργασία, θα πρέπει να προωθούμε τον χρήστη σε μία σελίδα 404.

 # apps/frontend/config/routing.yml

job_show_user:

url: /job/:company_slug/:location_slug/:id/:position_slug

class: sfDoctrineRoute

options:

model: JobeetJob

type: object

method_for_query: retrieveActiveJob

param: { module: job, action: show }

```
  requirements:

    id: \d+

    sf_method: [GET]
```

Η μέθοδος *retrieveActiveJob()* θα λάβει το *Doctrine_Query* object που δημιουργήθηκε από την route:

```
// lib/model/doctrine/JobeetJobTable.class.php
class JobeetJobTable extends Doctrine_Table
{
  public function retrieveActiveJob(Doctrine_Query $q)
  {
    $q->andWhere('a.expires_at > ?', date('Y-m-d H:i:s', time()));
    return $q->fetchOne();
  }
  // ...
```

Τώρα εάν προσπαθήσουμε να πάρουμε μία ληγμένη εργασία, θα προωθηθούμε σε μία σελίδα 404.

8 Η Κατηγορία Route

Πρώτα, θα πρέπει να προσθέσουμε μία διαδρομή για να ορίσουμε ένα όμορφο URL για την σελίδα των κατηγοριών. Το προσθέτουμε στην αρχή του αρχείου routing:

```
# apps/frontend/config/routing.yml

category:

  url:     /category/:slug

  class:    sfDoctrineRoute

  param:    { module: category, action: show }

  options:  { model: JobeetCategory, type: object }
```

Μία διαδρομή μπορεί να χρησιμοποιήσει οποιαδήποτε στήλη σαν παράμετρο από τα συσχετισμένα αντικείμενά της. Μπορεί επίσης να χρησιμοποιήσει οποιαδήποτε άλλη τιμή εάν υπάρχει συσχετισμένος accessor ο οποίος έχει οριστεί στην object class. Επειδή η παράμετρος slug δεν έχει αντίστοιχη στήλη στο category table, θα πρέπει να προσθέσουμε έναν εικονικό accessor στη *JobeetCategory* για να κάνουμε τη διαδρομή να λειτουργήσει:

```
// lib/model/doctrine/JobeetCategory.class.php

public function getSlug()

{
```

```
  return Jobeet::slugify($this->getName());
}
```

8.1 Ο Σύνδεσμος της Κατηγορίας

Τώρα, ας επεξεργαστούμε το template *indexSuccess.php* του module job για να προσθέσουμε τον σύνδεσμο στην σελίδα category:

```
<!-- some HTML code -->
      <h1>
        <?php echo link_to($category, 'category', $category) ?>
      </h1>
<!-- some HTML code -->
    </table>
    <?php if (($count = $category->countActiveJobs() -
sfConfig::get('app_max_jobs_on_homepage')) > 0): ?>
      <div class="more_jobs">
        and <?php echo link_to($count, 'category', $category) ?>
        more...
      </div>
    <?php endif; ?>
  </div>
  <?php endforeach; ?>
</div>
```

Προσθέτουμε τον σύνδεσμο αυτόν μόνο εάν υπάρχουν παραπάνω από 10 εργασίες στη συγκεκριμένη κατηγορία για να προβληθούν. Ο σύνδεσμος περιέχει τον αριθμό των εργασιών που δεν προβάλλονται. Για να λειτουργήσει το συγκεκριμένο template, θα πρέπει να προσθέσουμε τη μέθοδο *countActiveJobs()* στο *JobeetCategory*:

```
// lib/model/doctrine/JobeetCategory.class.php
public function countActiveJobs()
{
```

```
  $q = Doctrine_Query::create()
    ->from('JobeetJob j')
    ->where('j.category_id = ?', $this->getId());
  return Doctrine_Core::getTable('JobeetJob')->countActiveJobs($q);
}
```

Η μέθοδος *countActiveJobs()* χρησιμοποιεί μία μέθοδο *countActiveJobs()* η οποία δεν υπάρχει ακόμα στο *JobeetJobTable*. Αντικαθιστούμε το περιεχόμενο του αρχείου *JobeetJobTable.php* με τον παρακάτω κώδικα:

```
// lib/model/doctrine/JobeetJobTable.class.php
class JobeetJobTable extends Doctrine_Table
{
  public function retrieveActiveJob(Doctrine_Query $q)
  {
    return $this->addActiveJobsQuery($q)->fetchOne();
  }
  public function getActiveJobs(Doctrine_Query $q = null)
  {
    return $this->addActiveJobsQuery($q)->execute();
  }
  public function countActiveJobs(Doctrine_Query $q = null)
  {
    return $this->addActiveJobsQuery($q)->count();
  }
  public function addActiveJobsQuery(Doctrine_Query $q = null)
  {
```

```
    if (is_null($q))
    {
      $q = Doctrine_Query::create()
        ->from('JobeetJob j');
    }
    $alias = $q->getRootAlias();
    $q->andWhere($alias . '.expires_at > ?', date('Y-m-d H:i:s', time()))
      ->addOrderBy($alias . '.created_at DESC');
    return $q;
  }
}
```

Στη μέθοδο *countActiveJobs()* αντί να χρησιμοποιούμε το *execute()* και ύστερα να μετράμε τον αριθμό των αποτελεσμάτων, έχουμε χρησιμοποιήσει την πιο γρήγορη *count()* μέθοδο.

Έχουμε αλλάξει αρκετά αρχεία, απλά για αυτό το απλό χαρακτηριστικό. Αλλά κάθε φορά που έχουμε προσθέσει κάποιον κώδικα, έχουμε προσπαθήσει να τον τοποθετήσουμε στο σωστό layer της εφαρμογής καθώς και να τον κάνουμε να είναι επαναχρησιμοποιήσιμος. Στην επεξεργασία έχουμε ήδη αναδιαρθρώσει κάποιον ήδη υπάρχοντα κώδικα. Αυτή είναι μία τυπική ροή εργασίας όταν εργαζόμαστε με το symfony project.

8.2 Δημιουργία Category Module της Θέσης Εργασίας

Δημιουργία του category module:

$ php symfony generate:module frontend category

Όταν έχουμε πρόσβαση στην category page, η category route θα πρέπει να βρει την κατηγορία που συνδέεται με την αιτούσα slug variable. Καθώς το slug δεν αποθηκεύεται μέσα στη βάση δεδομένων, και επειδή δεν μπορούμε να ανάγουμε το όνομα της κατηγορίας από το slug, δεν υπάρχει τρόπος να βρούμε την κατηγορία που συνδέεται με το slug.

8.3 Ενημέρωση της Βάσης Δεδομένων

Θα πρέπει να προσθέσουμε ένα slug column για το **category table:** Μία Doctrine behavior που ονομάζεται *Sluggable* μπορεί να "προσέχει" αυτό το slug column. Απλά θα πρέπει να ενεργοποιήσουμε τη behavior στο *JobeetCategory* model και αυτό θα το αναλάβει για εμάς.

```
# config/doctrine/schema.yml

JobeetCategory:

  actAs:

    Timestampable: ~

    Sluggable:

      fields: [name]

  columns:

    name:

      type: string(255)

      notnull:  true
```

Τώρα που το slug είναι πραγματικό column, θα πρέπει να αφαιρέσουμε τη μέθοδο *getSlyug()* από το *JobeetCategory*.

Χρησιμοποιούμε την διεργασία *doctrine:build - - all - -and – load* για να ενημερώσουμε τα tables της βάσης δεδομένων με τα fixtures:

```
$ php symfony doctrine:build - - all - - and – load - - no – confirmation
```

Τώρα είναι όλα έτοιμα για να δημιουργήσουμε τη μέθοδο *executeShow()*. Αντικαθιστούμε το περιεχόμενο από το αρχείο των category actions με τον παρακάτω κώδικα:

```
// apps/frontend/modules/category/actions/actions.class.php

class categoryActions extends sfActions

{

  public function executeShow(sfWebRequest $request)

  {

    $this->category = $this->getRoute()->getObject();

  }

}
```

Το τελευταίο βήμα είναι να δημιουργήσουμε το template *showSuccess.php*:

```
// apps/frontend/modules/category/templates/showSuccess.php
<?php use_stylesheet('jobs.css') ?>
<?php slot('title', sprintf('Jobs in the %s category', $category->getName())) ?>
 <div class="category">
  <div class="feed">
    <a href="">Feed</a>
  </div>
  <h1><?php echo $category ?></h1>
</div>
<table class="jobs">
  <?php foreach ($category->getActiveJobs() as $i => $job): ?>
    <tr class="<?php echo fmod($i, 2) ? 'even' : 'odd' ?>">
      <td class="location">
        <?php echo $job->getLocation() ?>
      </td>
      <td class="position">
        <?php echo link_to($job->getPosition(), 'job_show_user', $job) ?>
      </td>
      <td class="company">
        <?php echo $job->getCompany() ?>
      </td>
    </tr>
  <?php endforeach; ?>
</table>
```

8.4 Partials

Όταν θέλουμε να επαναχρησιμοποιήσουμε ένα μέρος ενός template, θα πρέπει να δημιουργήσουμε ένα partial. Το partial είναι ένα απόσπασμα από κώδικα template το οποίο μπορεί να διαμοιραστεί σε διάφορα templates. Το partial είναι ακόμα ένα template που ξεκινάει με την κάτω παύλα (_).

Δημιουργούμε το αρχείο *_list.php*:

```
// apps/frontend/modules/job/templates/_list.php
<table class="jobs">
  <?php foreach ($jobs as $i => $job): ?>
    <tr class="<?php echo fmod($i, 2) ? 'even' : 'odd' ?>">
      <td class="location">
        <?php echo $job->getLocation() ?>
      </td>
      <td class="position">
        <?php echo link_to($job->getPosition(), 'job_show_user', $job) ?>
      </td>
      <td class="company">
        <?php echo $job->getCompany() ?>
      </td>
    </tr>
  <?php endforeach; ?>
</table>
```

Μπορούμε να συμπεριλάβουμε ένα partial χρησιμοποιώντας τον helper *include_partial()*:

```
<?php include_partial('job/list', array('jobs' => $jobs)) ?>
```

Το πρώτο argument του *include_partial()* είναι το όνομα του partial, και το δεύτερο argument είναι συστοιχία από μεταβλητές που θα περάσουν στο partial.

Αντικαθιστούμε τον κώδικα HTML <*table*> κι από τα δύο templates με την κλήση του *include_partial()*:

```
// in apps/frontend/modules/job/templates/indexSuccess.php

<?php include_partial('job/list', array('jobs' => $category->getActiveJobs(sfConfig::get('app_max_jobs_on_homepage')))) ?>

// in apps/frontend/modules/category/templates/showSuccess.php

<?php include_partial('job/list', array('jobs' => $category->getActiveJobs())) ?>
```

8.5 Σελιδοποίηση της Λίστας

Για να γίνει η σελιδοποίηση μιας λίστας από Doctrine objects, το symfony παρέχει μία αφιερωμένη class: *sfDoctrinePager*. Στην category action, αντί να περάσουμε τα job objects στο *showSuccess* template, περνάμε ένα pager:

```
// apps/frontend/modules/category/actions/actions.class.php

public function executeShow(sfWebRequest $request)

{

  $this->category = $this->getRoute()->getObject();

  $this->pager = new sfDoctrinePager(

    'JobeetJob',

    sfConfig::get('app_max_jobs_on_category')

  );

  $this->pager->setQuery($this->category->getActiveJobsQuery());

  $this->pager->setPage($request->getParameter('page', 1));

  $this->pager->init();

}
```

Ο *sfDoctrinePager* constructor παίρνει ένα model class και τον μέγιστο αριθμό από αντικείμενα για να επιστρέψει ανά σελίδα. Προσθέτουμε την τελευταία τιμή στο αρχείο ρύθμισης παραμέτρων:

```
# apps/frontend/config/app.yml

all:

  active_days:         30

  max_jobs_on_homepage: 10

  max_jobs_on_category: 20
```

Η μέθοδος *sfDoctrinePager::setQuery()* παίρνει ένα *Doctrine_Query* object για να χρησιμοποιήσει όταν επιλέγονται αντικείμενα από τη βάση δεδομένων.

Προσθέτουμε τη μέθοδο *getActiveJobsQuery()*:

```
// lib/model/doctrine/JobeetCategory.class.php

public function getActiveJobsQuery()

{

  $q = Doctrine_Query::create()

    ->from('JobeetJob j')

    ->where('j.category_id = ?', $this->getId());

  return Doctrine_Core::getTable('JobeetJob')->addActiveJobsQuery($q);

}
```

Τώρα που έχουμε ορίσει τη μέθοδο *getActiveJobsQuery()*, μπορούμε να αναδιαρθρώσουμε άλλες μεθόδους *JobeetCategory* για να τις χρησιμοποιήσουμε:

```
// lib/model/doctrine/JobeetCategory.class.php

public function getActiveJobs($max = 10){

  $q = $this->getActiveJobsQuery()

    ->limit($max);

  return $q->execute();

}

public function countActiveJobs()

{

  return $this->getActiveJobsQuery()->count();}
```

Τέλος, ας ενημερώσουμε το template:

```
<!-- apps/frontend/modules/category/templates/showSuccess.php -->
<?php use_stylesheet('jobs.css') ?>
<?php slot('title', sprintf('Jobs in the %s category', $category->getName())) ?>
<div class="category">
  <div class="feed">
    <a href="">Feed</a>
  </div>
  <h1><?php echo $category ?></h1>
</div>
<?php include_partial('job/list', array('jobs' => $pager->getResults())) ?>
<?php if ($pager->haveToPaginate()): ?>
  <div class="pagination">
    <a href="<?php echo url_for('category', $category) ?>?page=1">
      <img src="/images/first.png" alt="First page" title="First page" />
    </a>
    <a href="<?php echo url_for('category', $category) ?>?page=<?php echo $pager->getPreviousPage() ?>">
      <img src="/images/previous.png" alt="Previous page" title="Previous page" />
    </a>
    <?php foreach ($pager->getLinks() as $page): ?>
      <?php if ($page == $pager->getPage()): ?>
        <?php echo $page ?>
      <?php else: ?>
        <a href="<?php echo url_for('category', $category) ?>?page=<?php echo $page ?>"><?php echo $page ?></a>
      <?php endif; ?>
    <?php endforeach; ?>
```

```
    <a href="<?php echo url_for('category', $category) ?>?page=<?php echo $pager->getNextPage() ?>">
      <img src="/images/next.png" alt="Next page" title="Next page" />
    </a>
    <a href="<?php echo url_for('category', $category) ?>?page=<?php echo $pager->getLastPage() ?>">
      <img src="/images/last.png" alt="Last page" title="Last page" />
    </a>
  </div>
 <?php endif; ?>
 <div class="pagination_desc">
  <strong><?php echo count($pager) ?></strong> jobs in this category
  <?php if ($pager->haveToPaginate()): ?>
   - page <strong><?php echo $pager->getPage() ?>/<?php echo $pager->getLastPage() ?></strong>
  <?php endif; ?>
 </div>
```

Το περισσότερο μέρος του κώδικα αυτού ασχολείται με τους συνδέσμους σε άλλες σελίδες. Εδώ είναι η λίστα με τις μεθόδους *sfDoctrinePager* που χρησιμοποιούνται σε αυτό το template:

- ***getResulats()*:** Επιστρέφει ένα array από Doctrine objects για τη συγκεκριμένη σελίδα.
- ***getNbResults()*:** Επιστρέφει το συνολικό αριθμό των αποτελεσμάτων.
- ***haveToPaginate()*:** Επιστρέφει true εάν υπάρχουν παραπάνω από μία σελίδα.
- ***getLinks()*:** Επιστρέφει μία λίστα από συνδέσμους σελίδας για να τους προβάλλει.
- ***getPage()*:** Επιστρέφει τον αριθμό της τρέχουσας σελίδας.
- ***getPreviousPage()*:** Επιστρέφει τον αριθμό της προηγούμενης σελίδας.
- ***getNextPage()*:** Επιστρέφει τον αριθμό της επόμενης σελίδας.
- ***GetLastPage()*:** Επιστρέφει τον αριθμό της τελευταίας σελίδας.

Καθώς το *sfDoctrinePager* υλοποιεί τις διεπαφές Iterator και Countable, μπορούμε να χρησιμοποιήσουμε την function *count()* για να πάρουμε τον αριθμό των αποτελεσμάτων αντί για τη μέθοδο *getNbResults()*.

9 Έλεγχοι στο Symfony

Υπάρχουν δύο διαφορετικά είδη αυτοματοποιημένων τεστ στο symfony: unit tests | Unit Testing και functional tests.

Τα unit tests επαληθεύουν ότι η κάθε method και function λειτουργεί κανονικά. Το κάθε τεστ θα πρέπει να είναι όσο ανεξάρτητο γίνεται από τα άλλα.

Από την άλλη, τα functional τεστ επαληθεύουν το ότι η εφαρμογή των αποτελεσμάτων συμπεριφέρεται σωστά.

Όλα τα τεστ στο symfony βρίσκονται κάτω από τον φάκελο test/ του project. Συμπεριλαμβάνει δύο υπό-φακέλους, έναν για τα unit tests (*test/unit/*) κι ένα για τα functional tests (*test/functional/*).

9.1 Έλεγχοι Μονάδας

9.1.1 To lime Testing Framework

Όλα τα unit tests που έχουν γραφτεί με το lime framework ξεκινούν με τον ίδιο κώδικα:

```
require_once dirname(__FILE__).'/../bootstrap/unit.php';

$t = new lime_test(1);
```

Πρώτα το bootstrap αρχείο *unit.php* συμπεριλαμβάνεται για να αρχικοποιήσει κάποια πράγματα. Έπειτα δημιουργείται ένα καινούριο lime_test object και οι αριθμοί των ελέγχων που σχεδιάστηκαν να λανσαριστούν περνάνε σαν argument.

Ο έλεγχος πραγματοποιείται με την κλήση μίας μεθόδου η μίας function με ένα σύνολο από προκαθορισμένες εισόδους και στη συνέχεια με τη σύγκριση των αποτελεσμάτων με την αναμενόμενη έξοδο. Αυτή η σύγκριση καθορίζει το εάν ο έλεγχος περνάει η αποτυγχάνει.

Για να διευκολύνουμε τη σύγκριση, το *lime_test* object παρέχει αρκετές μεθόδους:

Method	**Περιγραφή**
ok($test):	Ελέγχει μία συνθήκη και περνάει εάν είναι true
is($value1, $value2):	Συγκρίνει δύο τιμές και περνάει εάν είναι ίσες (==)
isnt($value1, $value2):	Συγκρίνει δύο τιμές και περνάει εάν είναι άνισες
like($string, $regexp):	Ελέγχει ένα string ενάντια σε μία κανονική έκφραση
unlike($string, $regexp):	Ελέγχει εάν ένα string δεν ταιριάζει με μία κανονική έκφραση
is_deeply($array1, $array2):	Ελέγχει εάν τα δύο arrays έχουν τις ίδιες τιμές

Το lime_test object παρέχει επίσης άλλες κατάλληλες μεθόδους:

Μέθοδος	**Περιγραφή**
fail():	Πάντα αποτυγχάνει
pass():	Πάντα περνάει
skip($msg. $nb_tests):	Μετράει καθώς το $nb_tests ελέγχει
todo():	Μετράει σαν ένας έλεγχος

9.2 Εκτελώντας Ελέγχους Μονάδας

Για να επεξηγήσουμε το unit testing, θα ελέγξουμε το Jobeet class.

Δημιουργούμε ένα αρχείο ελέγχου */unit/JobeetTest.php* και αντιγράφουμε μέσα του τον παρακάτω κώδικα:

```
// test/unit/JobeetTest.php

require_once dirname(__FILE__).'/../bootstrap/unit.php';

 $t = new lime_test(1);

$t->pass('This test always passes.');
```

Για να τρέξουμε τους ελέγχους, μπορούμε να εκτελέσουμε το αρχείο:

$ php test/unit/JobeetTest.php

Ή να χρησιμοποιήσουμε την διεργασία test:unit:

$ php symfony test:unit Jobeet

9.4 Ελέγχοντας το slugify

Αντικαθιστούμε το περιεχόμενο του αρχείου ελέγχου με τον παρακάτω κώδικα:

```
// test/unit/JobeetTest.php

require_once dirname(__FILE__).'/../bootstrap/unit.php';

$t = new lime_test(6);

$t->is(Jobeet::slugify('Sensio'), 'sensio');

$t->is(Jobeet::slugify('sensio labs'), 'sensio-labs');

$t->is(Jobeet::slugify('sensio   labs'), 'sensio-labs');

$t->is(Jobeet::slugify('paris,france'), 'paris-france');

$t->is(Jobeet::slugify('  sensio'), 'sensio');

$t->is(Jobeet::slugify('sensio  '), 'sensio');
```

Παρατηρούμε ότι η κάθε γραμμή ελέγχει μόνο ένα πράγμα.

Μπορούμε τώρα να εκτελέσουμε το αρχείο ελέγχου. Εάν περάσουν όλοι οι έλεγχοι, θα δούμε ένα "green bar" διαφορετικά το "red bar" θα μας ειδοποιήσει ότι κάποιοι έλεγχοι δεν πέρασαν και θα πρέπει να τα διορθώσουμε. .

Εάν αποτύχει ένας έλεγχος, η έξοδος θα μας δώσει κάποιες πληροφορίες για τον λόγο για τον οποίο απέτυχε.

Όλες οι μέθοδοι των lime test παίρνουν ένα string σαν το τελευταίο τους argument το οποίο εξυπηρετεί σαν την περιγραφή του ελέγχου. Είναι πολύ βολικό καθώς μας ωθεί στο να περιγράψουμε στο τι ακριβώς ελέγχουμε. Μπορεί επίσης να εξυπηρετήσει σαν μία μορφή τεκμηρίωσης για την αναμενόμενη συμπεριφορά της μεθόδου. Προσθέτουμε μερικά μηνύματα στο αρχείο ελέγχου *slugify*:

```
require_once dirname(__FILE__).'/../bootstrap/unit.php';

$t = new lime_test(6);

$t->comment('::slugify()');

$t->is(Jobeet::slugify('Sensio'), 'sensio', '::slugify() converts all characters to lower case');

$t->is(Jobeet::slugify('sensio labs'), 'sensio-labs', '::slugify() replaces a white space by a -');

$t->is(Jobeet::slugify('sensio   labs'), 'sensio-labs', '::slugify() replaces several white spaces by a single -');

$t->is(Jobeet::slugify('  sensio'), 'sensio', '::slugify() removes - at the beginning of a string');

$t->is(Jobeet::slugify('sensio  '), 'sensio', '::slugify() removes - at the end of a string');

$t->is(Jobeet::slugify('paris,france'), 'paris-france', '::slugify() replaces non-ASCII characters by a -');
```

Το string της περιγραφής ελέγχου είναι ένα πολύτιμο εργαλείο όταν προσπαθούμε να καταλάβουμε το τι να ελέγξουμε.

9.5 Προσθέτοντας Ελέγχους για Καινούρια Χαρακτηριστικά

Το slug για ένα άδειο string είναι ένα άδειο string. Αλλά ένα άδειο string σε ένα URL δεν είναι καλή ιδέα. Ας αλλάξουμε την μέθοδο *slugify()* έτσι ώστε να επιστρέφει το string "*n-a*" στην περίπτωση που υπάρχει άδειο string.

Μπορούμε να γράψουμε τον έλεγχο πρώτα, έπειτα να ενημερώσουμε τη μέθοδο, ή και αντίστροφα.

```
$t->is(Jobeet::slugify(''), 'n-a', '::slugify() converts the empty string to n-a');
```

Αυτή η μεθοδολογία ανάπτυξης όπου γράφουμε πρώτα τους ελέγχους και στη συνέχεια εφαρμόζουμε τα χαρακτηριστικά, είναι γνωστή ως Test Driven Development (TDD).

Επεξεργαζόμαστε την Jobeet class και προσθέτουμε την παρακάτω συνθήκη στην αρχή:

```
// lib/Jobeet.class.php

static public function slugify($text)

{

  if (empty($text))

  {

    return 'n-a';

  }  // ..
```

9.6 Έλεγχοι Doctrine της Μονάδας

9.6.1 Ρύθμιση Παραμέτρων της Βάσης Δεδομένων

Ο έλεγχος της μονάδας σε ένα Doctrine model class είναι λίγο πιο πολύπλοκο καθώς απαιτεί μία σύνδεση βάσης δεδομένων. Θα δημιουργήσουμε μία ειδική βάση δεδομένων για τους ελέγχους:

```
$ php symfony configure:database --name=doctrine --class=sfDoctrineDatabase --env=test "mysql:host=localhost;dbname=jobeet_test" root mYsEcret
```

Η επιλογή *env* λέει στην διεργασία ότι οι ρυθμίσεις της βάσης δεδομένων είναι μόνο για το περιβάλλον του ελέγχου.

Εφόσον ρυθμίσαμε την βάση δεδομένων, μπορούμε να την κάνουμε bootstrap χρησιμοποιώντας τη διεργασία *doctrine:insert-sql*:

```
$ mysqladmin -uroot -pmYsEcret create jobeet_test

$ php symfony doctrine:insert-sql –env=test
```

9.7 Έλεγχος Δεδομένων

Τώρα που έχουμε την ειδική βάση δεδομένων για τους ελέγχους, χρειαζόμαστε έναν τρόπο για να φορτώσουμε μερικά δεδομένα ελέγχου.

Η διεργασία *doctrine:data-load* εσωτερικά της χρησιμοποιεί τη μέθοδο *Doctrine_Core::loadData()* για να φορτώσει τα δεδομένα:

```
Doctrine_Core::loadData(sfConfig::get('sf_test_dir').'/fixtures');
```

Η μέθοδος *loadData()* παίρνει σαν πρώτο της argument έναν φάκελο ή ένα αρχείο. Μπορεί επίσης να πάρει ένα array από φακέλους και/ή αρχεία.

Για τους ελέγχους θα βάλουμε τα fixtures μέσα στον φάκελο *test/fixtures/*. Αυτά τα fixtures θα χρησιμοποιηθούν για το Doctrine unit και τους λειτουργικούς ελέγχους.

Αντιγράφουμε τα αρχεία από τον φάκελο *data/fixtures* στον φάκελο *test/fixtures/*

9.8 Ελέγχοντας το JobeetJob

Θα δημιουργήσουμε κάποιους ελέγχους μονάδας για την model class JobeetJob.

Καθώς όλοι οι Doctrine έλεγχοι μονάδας θα ξεκινούν με τον ίδιο κώδικα, δημιουργούμε ένα αρχείο *Doctrine.php* στον *test* φάκελο *bootstrap/* με τον παρακάτω κώδικα:

```
// test/bootstrap/Doctrine.php

include(dirname(__FILE__).'/unit.php');

$configuration = ProjectConfiguration::getApplicationConfiguration( 'frontend', 'test', true);

new sfDatabaseManager($configuration);

Doctrine_Core::loadData(sfConfig::get('sf_test_dir').'/fixtures');
```

Το script είναι αυτονόητο:

- Για τους front controllers, αρχικοποιούμε ένα configuration object για το test environment:
- *$configuration = ProjectConfiguration::getApplicationConfiguration('frontend', 'test', true);*
- Δημιουργούμε έναν διαχειριστή βάσης δεδομένων. Αρχικοποιεί την σύνδεση Doctrine φορτώνοντας το αρχείο *databases.yml*
- *new sfDatabaseManager($configuration);*
- Φορτώνουμε τα δεδομένα ελέγχου χρησιμοποιώντας το *Doctrine_Core::loadData()*:

```
Doctrine_Core::loadData(sfConfig::get('sf_test_dir').'/fixtures');
```

Τώρα μπορούμε να ξεκινήσουμε τους ελέγχους στο JobeetJob class.

Αρχικά δημιουργούμε το αρχείο *JobeetJobTest.php* στον φάκελο *test/unit/model*:

```
// test/unit/model/JobeetJobTest.php

include(dirname(__FILE__).'/../../bootstrap/Doctrine.php');

$t = new lime_test(1);
```

Στη συνέχεια προσθέτουμε έναν έλεγχο στη μέθοδο *getCompanySlug()*:

```
$t->comment('->getCompanySlug()');

$job = Doctrine_Core::getTable('JobeetJob')->createQuery()->fetchOne();

$t->is($job->getCompanySlug(), Jobeet::slugify($job->getCompany()), '->getCompanySlug() return the slug for the company');
```

Παρατηρούμε ότι ελέγχουμε μόνο τη μέθοδο *getCompanySlug()* και όχι εάν το slug είναι σωστό ή όχι καθώς αυτό το έχουμε ήδη ελέγξει κάπου αλλού. Η γραφή των ελέγχων για τη μέθοδο *save()* είναι λίγο πιο περίπλοκη:

```
$t->comment('->save()');

$job = create_job();

$job->save();

$expiresAt = date('Y-m-d', time() + 86400 * sfConfig::get('app_active_days'));

$t->is($job->getDateTimeObject('expires_at')->format('Y-m-d'), $expiresAt, '->save() updates expires_at if not set');

$job = create_job(array('expires_at' => '2008-08-08'));

$job->save();

$t->is($job->getDateTimeObject('expires_at')->format('Y-m-d'), '2008-08-08', '->save() does not update expires_at if set');

 function create_job($defaults = array())

{

  static $category = null;

   if (is_null($category))

  {

    $category = Doctrine_Core::getTable('JobeetCategory')

      ->createQuery()

      ->limit(1)

      ->fetchOne();

  }
```

```
    $job = new JobeetJob();
    $job->fromArray(array_merge(array(
      'category_id'  => $category->getId(),
      'company'      => 'Sensio Labs',
      'position'     => 'Senior Tester',
      'location'     => 'Paris, France',
      'description'  => 'Testing is fun',
      'how_to_apply' => 'Send e-Mail',
      'email'        => 'job@example.com',
      'token'        => rand(1111, 9999),
      'is_activated' => true,
    ), $defaults));
    return $job;
  }
```

9.9 Harness Έλεγχων Μονάδας

Η διεργασία *test:unit* μπορεί να χρησιμοποιηθεί για να πραγματοποιηθούν όλα τα unit tests σε ένα project:

```
$ php symfony test:unit
```

Η έξοδος της διεργασίες μας δείχνει ένα ο έλεγχος περνάει ή όχι:

10 Λειτουργικοί Ελέγχοι

Τα functional tests είναι ένα καλό εργαλείο για τον έλεγχο της εφαρμογής από άκρη σε άκρη: από το request που έγινε από έναν browser μέχρι το response που απέστειλε ο server. Ελέγχουν όλα τα layers μιας εφαρμογής: το routing, το model, τις actions, και τα templates.

10.1 Το sfBrowser

Στο symfony, τα functional tests τρέχουν μέσα από έναν ειδικό browser, που εκτελείται μέσα από τη *sfBrowser* class. Ενεργεί σαν έναν προσαρμοσμένο browser για την εφαρμογή μας ο οποίος είναι και συνδεδεμένος μαζί της, χωρίς την ανάγκη για έναν web server. Μας δίνει πρόσβαση σε όλα τα symfony objects πριν και μετά την αίτηση, δίνοντας μας την ευκαιρία να τα εξετάσει και να κάνει τους ελέγχους που θέλουμε προγραμματιστικά.

Το *sfBrowser* παρέχει μεθόδους οι οποίες εξομοιώνουν την περιήγηση που γίνεται σε έναν κλασικό browser:

Method	Description
get():	Λαμβάνει ένα URL
post():	Τοποθετεί σε ένα URL
call():	Καλεί ένα URL
back():	Πάει μία σελίδα πίσω
forward():	Πάει μία σελίδα μπροστά
reload():	Φορτώνει ξανά την τρέχουσα σελίδα
click():	Κάνει κλικ σε ένα link ή ένα button
select():	Επιλέγει ένα radiobutton ή ένα checkbox
deselect():	Αποεπιλέγει ένα radiobutton ή ένα checkbox
restart():	Επανεκκινεί τον browser

Μερικά χρήσιμα παραδείγματα από τις μεθόδους *sfBrowser*:

```
$browser = new sfBrowser();

$browser->

  get('/')->

  click('Design')->

  get('/category/programming?page=2')->

  get('/category/programming', array('page' => 2))->
```

post('search', array('keywords' => 'php'));

Το *sfBrowser* περιέχει επιπρόσθετες μεθόδους για να ρυθμίζουν τη συμπεριφορά του browser:

Method	Description
setHttpHeader():	*Ορίζει έναν HTTP header*
setAuth():	*Ορίζει τα βασικά πιστοποιητικά επικύρωσης*
setCookie():	*Ορίζει ένα cookie*
removeCookie():	*Αφαιρεί ένα cookie*
clearCookies():	*Καθαρίζει όλα τα cookies*
followRedirect():	*Ακολουθεί ένα redirect*

10.2 Η κλάση sfTestFunctional

Έχουμε έναν browser, αλλά χρειαζόμαστε έναν τρόπο για να εξετάσει τα symfony objects τα οποία θα κάνουν τον πραγματικό έλεγχο.

Οι test methods παρέχονται από μία άλλη class, η *sfTestFunctional* που παίρνει μία υπόδειξη *sfBrowser* μέσα στον constructor. Η class *sfTestFunctional* αναθέτει τους ελέγχους σε tester objects. Αρκετοί ελεγκτές συνοδεύονται με το symfony, αλλά επίσης μπορούμε να δημιουργήσουμε τους δικούς μας.

Τα functional tests όπως έχουμε δει αποθηκεύονται κάτω από τον φάκελο *test/functional/*. Για το Jobeet, οι έλεγχοι βρίσκονται στον υπό-φάκελο *test/functional/frontend/* καθώς κάθε εφαρμογή έχει τον δικό της υπό-φάκελο. Αυτός ο φάκελος περιέχει ήδη δύο αρχεία: το *categoryActionsTest.php*, και το *jobActionsTest.php* καθώς όλες οι διεργασίες δημιουργούν ένα module αυτόματα δημιουργούν ένα βασικό αρχείο functional test:

```
// test/functional/frontend/categoryActionsTest.php

include(dirname(__FILE__).'/../../bootstrap/functional.php');

$browser = new sfTestFunctional(new sfBrowser());

$browser->

  get('/category/index')->

  with('request')->begin()->

    isParameter('module', 'category')->

    isParameter('action', 'index')->
```

```
  end()->

  with('response')->begin()->

    isStatusCode(200)->
    checkElement('body', '!/This is a temporary page/')->
  end()
;
```

Οι methods του *sfBrowser* και *sfTestFunctional* εφαρμόζουν μία "ρευστή" διεπαφή επιστρέφοντας πάντα το *$this*. Μας επιτρέπει να "αλυσοδέσουμε" τα method calls για καλύτερη αξιοπιστία. Το παραπάνω απόσπασμα είναι ισοδύναμο με το:

```
// test/functional/frontend/categoryActionsTest.php

include(dirname(__FILE__).'/../../bootstrap/functional.php');

$browser = new sfTestFunctional(new sfBrowser());

$browser->get('/category/index');

$browser->with('request')->begin();

$browser->isParameter('module', 'category');

$browser->isParameter('action', 'index');

$browser->end();

$browser->with('response')->begin();

$browser->isStatusCode(200);

$browser->checkElement('body', '!/This is a temporary page/');

$browser->end();
```

Οι έλεγχοι τρέχουν μέσα σε ένα tester block context. Ένα tester block context ξεκινάει με το *(' TESTER NAME ') -> begin()* και τελειώνει με το *end()*:

```
$browser->

  with('request')->begin()->

    isParameter('module', 'category')->

    isParameter('action', 'index')->

  end();
```

Ο κώδικας ελέγχει εάν το module που αιτείται παράμετρο ισούται με το category και εάν το action ισούται με το index.

10.3 Ο ελεγκτής του αιτήματος (Request Tester)

Ο request tester παρέχει μεθόδους ελεγκτή για να αυτό-ελεγχθεί και να ελέγξει το *sfWebRequest* object:

Method	Description
isParameter():	Ελέγχει την τιμή της παραμέτρου ενός αιτήματος
isFormat():	Ελέγχει τη διαμόρφωση ενός αιτήματος
isMethod():	Ελέγχει τη μέθοδο
hasCookie():	Ελέγχει εάν το αίτημα έχει ένα cookie με το όνομα
isCookie():	Ελέγχει την τιμή ενός cookie

10.4.Ο ελεγκτής της Απάντησης (Response Tester)

Υπάρχει επίσης μία κλάση απαντητικού ελεγκτή που παρέχει μεθόδους ελεγκτή απέναντι στο *sfWebResponse* object:

Method	Description
checkElement():	Ελέγχει ένας απαντητικός επιλογέας CSS ταιριάζει με κάποια κριτήρια
checkForm():	Ελέγχει ένα sfForm form object
debug():	Τυπώνει την έξοδο της απάντησης για να διευκολύνει την αποσφαλμάτωση
matches():	Ελέγχει την απάντηση απέναντι σε ένα regexp
isHeader():	Ελέγχει την τιμή ενός header
isStatusCode():	Ελέγχει τον κώδικα της κατάστασης της απάντησης
isRedirected():	Ελέγχει εάν η τρέχουσα απάντηση είναι ανακατεύθυνση
isValid():	Ελέγχει εάν η απάντηση είναι καλά συγκροτημένη

10.5 Εκτελώντας Λειτουργικούς Ελέγχους

Όπως και για τα unit tests, τα functional tests μπορούν να τρέξουν εκτελώντας κατευθείαν το αρχείο ελέγχου:

$ php test/functional/frontend/categoryActionsTest.php

Ή χρησιμοποιώντας τη διεργασία *test:functional*
$ php symfony test:functional frontend categoryActions

10.6 Έλεγχος Δεδομένων

Όπως και στα Doctrine unit tests, θα πρέπει να φορτώσουμε δεδομένα ελέγχου κάθε φορά που τρέχουμε ένα functional test. Μπορούμε να επαναχρησιμοποιήσουμε τον κώδικα που έχουμε γράψει προηγουμένως:

```
include(dirname(__FILE__).'/../../bootstrap/functional.php');

$browser = new sfTestFunctional(new sfBrowser());

Doctrine_Core::loadData(sfConfig::get('sf_test_dir').'/fixtures');
```

Το να φορτώσουμε δεδομένα σε ένα functional test είναι λίγο πιο εύκολο από ότι σε ένα unit test καθώς η βάση δεδομένων έχει ήδη αρχικοποιηθεί από το bootstrapping script.

Όπως και στα unit tests, δεν θα αντιγράψουμε και θα επικολλήσουμε αυτό το απόσπασμα από κώδικα σε κάθε αρχείο ελέγχου, αλλά θα δημιουργήσουμε την δικιά μας functional class η οποία "κληρονομεί" από το *sfTestFunctional*:

```
// lib/test/JobeetTestFunctional.class.php

class JobeetTestFunctional extends sfTestFunctional

{

  public function loadData()

  {

    Doctrine_Core::loadData(sfConfig::get('sf_test_dir').'/fixtures');

     return $this;

  }

}
```

10.7 Γράφοντας Λειτουργικούς Έλεγχους

Θα κάνουμε έλεγχο της αρχικής σελίδας του Jobeet επεξεργάζοντας το αρχείο ελέγχου *JobActionsTest.php*. Αντικαθιστούμε τον κώδικα με τον παρακάτω:

10.8 Οι ληγμένες εργασίες δεν εμφανίζονται

```
// test/functional/frontend/jobActionsTest.php

include(dirname(__FILE__).'/../../bootstrap/functional.php');

 $browser = new JobeetTestFunctional(new sfBrowser());

$browser->loadData();

 $browser->info('1 - The homepage')->

  get('/')->

  with('request')->begin()->

    isParameter('module', 'job')->

    isParameter('action', 'index')->

  end()->

  with('response')->begin()->

    info('  1.1 - Expired jobs are not listed')->

    checkElement('.jobs td.position:contains("expired")', false)->

  end()

;
```

Όπως και με το lime, ένα μήνυμα πληροφορίας μπορεί να εισαχθεί καλώντας τη μέθοδο info() για να γίνει η έξοδος πιο ευανάγνωστη. Για να επαληθεύσουμε τον αποκλεισμό των ληγμένων εργασιών από την αρχική σελίδα, ελέγχουμε ότι ο CSS επιλογέας *.jobs td.position:contains("expired")* δεν ταιριάζει πουθενά στο HTML απαντητικό περιεχόμενο. Όταν το δεύτερο argument από την μέθοδο *checkElement()* είναι Boolean, η μέθοδος ελέγχει την ύπαρξη κόμβων που να ταιριάζουν με τον CSS επιλογέα.

10.9 Μόνο n Θέσεις Εργασίας Εμφανίζονται για μία Κατηγορία

Προσθέτουμε τον παρακάτω κώδικα στο τέλος του αρχείου ελέγχου:

```
// test/functional/frontend/jobActionsTest.php
$max = sfConfig::get('app_max_jobs_on_homepage');
$browser->info('1 - The homepage')->
  get('/')->
  info(sprintf('  1.2 - Only %s jobs are listed for a category', $max))->
  with('response')->
    checkElement('.category_programming tr', $max)
;
```

10.10 Μία Κατηγορία έχει έναν Σύνδεσμο στην Σελίδα Κατηγοριών Μόνο όταν υπάρχουν πολλές Θέσεις Εργασίας

```
// test/functional/frontend/jobActionsTest.php
$browser->info('1 - The homepage')->
  get('/')->
  info('  1.3 - A category has a link to the category page only if too many jobs')->
  with('response')->begin()->
    checkElement('.category_design .more_jobs', false)->
    checkElement('.category_programming .more_jobs')->
  end()
;
```

10.11 Οι Θέσεις Εργασίας Ταξινομούνται ανά Ημερομηνία

```
$q = Doctrine_Query::create()
  ->select('j.*')
  ->from('JobeetJob j')
  ->leftJoin('j.JobeetCategory c')
  ->where('c.slug = ?', 'programming')
  ->andWhere('j.expires_at > ?', date('Y-m-d', time()))
  ->orderBy('j.created_at DESC');
$job = $q->fetchOne();
$browser->info('1 - The homepage')->
  get('/')->
  info('  1.4 - Jobs are sorted by date')->
  with('response')->begin()->
    checkElement(sprintf('.category_programming tr:first a[href*="/%d/"]', $job->getId()))->
  end();
```

Θα μεταφέρουμε τον κώδικα στην *JobeetTestFunctional* class που έχουμε δημιουργήσει. Αυτή η class λειτουργεί σαν Domain Specific λειτουργική ελεγκτική κλάση για το Jobeet:

```
// lib/test/JobeetTestFunctional.class.php
class JobeetTestFunctional extends sfTestFunctional
{
  public function getMostRecentProgrammingJob()
  {
    $q = Doctrine_Query::create()
      ->select('j.*')
      ->from('JobeetJob j')
      ->leftJoin('j.JobeetCategory c')
      ->where('c.slug = ?', 'programming');
```

```
    $q = Doctrine_Core::getTable('JobeetJob')->addActiveJobsQuery($q);
    return $q->fetchOne();   }    // ...
```

Μπορούμε τώρα να αντικαταστήσουμε τον προηγούμενο κώδικα ελέγχου με τον παρακάτω:

```
// test/functional/frontend/jobActionsTest.php
$browser->info('1 - The homepage')->
  get('/')->
  info('  1.4 - Jobs are sorted by date')->
  with('response')->begin()->
    checkElement(sprintf('.category_programming tr:first a[href*="/%d/"]',
      $browser->getMostRecentProgrammingJob()->getId()))->
  end()
;
```

10.12 Κάθε Θέση Εργασίας στην Αρχική Σελίδα μπορεί να πατηθεί

```
$job = $browser->getMostRecentProgrammingJob();
$browser->info('2 - The job page')->
  get('/')->
  info('  2.1 - Each job on the homepage is clickable and give detailed information')->
  click('Web Developer', array(), array('position' => 1))->
  with('request')->begin()->
    isParameter('module', 'job')->
    isParameter('action', 'show')->
    isParameter('company_slug', $job->getCompanySlug())->
    isParameter('location_slug', $job->getLocationSlug())->
    isParameter('position_slug', $job->getPositionSlug())->
    isParameter('id', $job->getId())->
  end();
```

Παρακάτω, παρουσιάζεται όλος ο κώδικας που χρειάζεται για να ελέγξει την εργασία και τις σελίδες των κατηγοριών:

```
// lib/test/JobeetTestFunctional.class.php
class JobeetTestFunctional extends sfTestFunctional
{
  public function loadData()
  {
    Doctrine_Core::loadData(sfConfig::get('sf_test_dir').'/fixtures');
    return $this;
  }
  public function getMostRecentProgrammingJob()
  {
    $q = Doctrine_Query::create()
      ->select('j.*')
      ->from('JobeetJob j')
      ->leftJoin('j.JobeetCategory c')
      ->where('c.slug = ?', 'programming');
    $q = Doctrine_Core::getTable('JobeetJob')->addActiveJobsQuery($q);
    return $q->fetchOne();
  }
  public function getExpiredJob()
  {
    $q = Doctrine_Query::create()
      ->from('JobeetJob j')
      ->where('j.expires_at < ?', date('Y-m-d', time()));
    return $q->fetchOne();
  }
}
```

```
// test/functional/frontend/jobActionsTest.php
include(dirname(__FILE__).'/../../bootstrap/functional.php');
$browser = new JobeetTestFunctional(new sfBrowser());
$browser->loadData();
$browser->info('1 - The homepage')->
  get('/')->
  with('request')->begin()->
    isParameter('module', 'job')->
    isParameter('action', 'index')->
  end()->
  with('response')->begin()->
    info('  1.1 - Expired jobs are not listed')->
    checkElement('.jobs td.position:contains("expired")', false)->
  end();
$max = sfConfig::get('app_max_jobs_on_homepage');
$browser->info('1 - The homepage')->
  info(sprintf('  1.2 - Only %s jobs are listed for a category', $max))->
  with('response')->
    checkElement('.category_programming tr', $max);
$browser->info('1 - The homepage')->
  get('/')->
  info('  1.3 - A category has a link to the category page only if too many jobs')->
  with('response')->begin()->
    checkElement('.category_design .more_jobs', false)->
    checkElement('.category_programming .more_jobs')->
  end();
$browser->info('1 - The homepage')->
  info('  1.4 - Jobs are sorted by date')->
  with('response')->begin()->
    checkElement(sprintf('.category_programming tr:first a[href*="/%d/"]', $browser-
>getMostRecentProgrammingJob()->getId()))->
  end();
$job = $browser->getMostRecentProgrammingJob();
 $browser->info('2 - The job page')->
  get('/')->
   info('  2.1 - Each job on the homepage is clickable and give detailed information')->
  click('Web Developer', array(), array('position' => 1))->
  with('request')->begin()->
    isParameter('module', 'job')->
    isParameter('action', 'show')->
    isParameter('company_slug', $job->getCompanySlug())->
```

```
      isParameter('location_slug', $job->getLocationSlug())->
      isParameter('position_slug', $job->getPositionSlug())->
      isParameter('id', $job->getId())->
    end()->
     info('  2.2 - A non-existent job forwards the user to a 404')->
    get('/job/foo-inc/milano-italy/0/painter')->
    with('response')->isStatusCode(404)->
     info('  2.3 - An expired job page forwards the user to a 404')->
    get(sprintf('/job/sensio-labs/paris-france/%d/web-developer', $browser->getExpiredJob()->getId()))->
    with('response')->isStatusCode(404);

  // test/functional/frontend/categoryActionsTest.php
  include(dirname(__FILE__).'/../../bootstrap/functional.php');
   $browser = new JobeetTestFunctional(new sfBrowser());
  $browser->loadData();
   $browser->info('1 - The category page')->
    info('  1.1 - Categories on homepage are clickable')->
    get('/')->
    click('Programming')->
    with('request')->begin()->
      isParameter('module', 'category')->
      isParameter('action', 'show')->
      isParameter('slug', 'programming')->
    end()->
     info(sprintf('  1.2 - Categories with more than %s jobs also have a "more" link', sfConfig::get('app_max_jobs_on_homepage')))->
    get('/')->
    click('27')->
    with('request')->begin()->
      isParameter('module', 'category')->
      isParameter('action', 'show')->
      isParameter('slug', 'programming')->
    end()->
    info(sprintf('  1.3 - Only %s jobs are listed', sfConfig::get('app_max_jobs_on_category')))->
    with('response')->checkElement('.jobs tr', sfConfig::get('app_max_jobs_on_category'))->
     info('  1.4 - The job listed is paginated')->
    with('response')->begin()->
      checkElement('.pagination_desc', '/32 jobs/')->
      checkElement('.pagination_desc', '#page 1/2#')->
    end()->
     click('2')->
```

```
  with('request')->begin()->
    isParameter('page', 2)->
  end()->
  with('response')->checkElement('.pagination_desc', '#page 2/2#')
;
```

10.13 Αποσφαλμάτωση Λειτουργικών Ελέγχων

Το symfony παρέχει τη μέθοδο *~debug|Debug~()* για να εξάγει τον response header και το περιεχόμενο:

$browser->with('response')->debug();

Η μέθοδος *debug()* μπορεί να εισαχθεί οπουδήποτε μέσα σε ένα response tester block και θα σταματήσει την εκτέλεση του script.

10.14 Harness Λειτουργικών Ελέγχων

Η διεργασία *test:functional* μπορεί επίσης να χρησιμοποιηθεί για να τρέξει όλα τα functional tests για μία εφαρμογή:

$ php symfony test:functional frontend

Η διεργασία εξάγει μία μονή γραμμή για κάθε αρχείο ελέγχου:.

10.15 Harness Έλεγχοι

Υπάρχει επίσης διεργασία που τρέχει όλους τους ελέγχους σε ένα project (unit και functional):

$ php symfony test:all.

Οι διεργασίες *test:all* έχουν την επιλογή *--only-failed* η οποία εξαναγκάζει τη διεργασία να τρέξει ξανά μόνο τα test τα οποία απέτυχαν σε προηγούμενη εκτέλεση:

$ php symfony test:all –only-failed

11 Το Form Framework

Το form framework είναι κατασκευασμένο από τρία μέρη:

- **validation:** Το validation sub-framework παρέχει classes για να επικυρώσει τις εισόδους (integer, string, email address, ...)
- **widgets:** Το widget sub-framework παρέχει classes για να εξάγει πεδία HTML (input, textarea, select, ...)

- **forms:** Οι form classes αναπαριστούν φόρμες που έχουν δημιουργηθεί από widgets και validators και παρέχουν μεθόδους οι οποίες βοηθάνε στη διαχείριση της φόρμας. Το κάθε πεδίο της φόρμας έχει το δικό του validator και widget.

11.1 Forms

Μία φόρμα του symfony είναι μια class η οποία έχει δημιουργηθεί από πεδία. Το κάθε πεδίο έχει ένα όνομα, ένα validator και ένα widget. Μία απλή *ContactForm* μπορεί να οριστεί με την παρακάτω class:

```
class ContactForm extends sfForm
{
  public function configure()
  {
    $this->setWidgets(array(
      'email'   => new sfWidgetFormInputText(),
      'message' => new sfWidgetFormTextarea(),
    ));
    $this->setValidators(array(
      'email'   => new sfValidatorEmail(),
      'message' => new sfValidatorString(array('max_length' => 255)),
    ));
  }
}
```

Τα πεδία της φόρμας ρυθμίζονται από τη μέθοδο *configure()* χρησιμοποιώντας τις μεθόδους *setValidators()* και *setWidgets()*.

11.2 Doctrine Forms

Καθώς το symfony γνωρίζει τα πάντα για το μοντέλο της βάσης δεδομένων μας, μπορεί αυτόματα να παράγει φόρμες βασισμένες πάνω σε αυτή την πληροφορία. Όταν τρέχουμε τη διεργασία *doctrine:build --all* το symfony αυτόματα καλεί τη διεργασία *doctrine:build --forms*:

$ php symfony doctrine:build –forms

Η διεργασία *doctrine:build - -forms* παράγει φόρμα από τις classes στον φάκελο *lib/form/*. Η οργάνωση αυτών των παραγόμενων αρχείων είναι παρόμοια με αυτή του *lib/model*. Κάθε model class έχει μία συσχετισμένη form class, η οποία είναι άδεια από προεπιλογή καθώς κληρονομεί από μία base class:

```
// lib/form/doctrine/JobeetJobForm.class.php

class JobeetJobForm extends BaseJobeetJobForm

{

  public function configure()

  {

  }

}
```

11.3.Προσαρμόζοντας τη Form της Θέσης Εργασίας

Πρώτα αλλάζουμε τον σύνδεσμο “Post a job” στο layout έτσι ώστε να μπορούμε να ελέγξουμε τις αλλαγές απευθείας στον browser:

```
<!-- apps/frontend/templates/layout.php -->

<a href="<?php echo url_for('job_new') ?>">Post a Job</a>
```

Αφαιρούμε πεδία από μία form έτσι ώστε μερικά από τα table columns να μην μπορεί να τα επεξεργαστεί ο τελικός χρήστης:

```
// lib/form/doctrine/JobeetJobForm.class.php

class JobeetJobForm extends BaseJobeetJobForm

{

  public function configure()

  {
```

```
    unset(
      $this['created_at'], $this['updated_at'],
      $this['expires_at'], $this['is_activated']
    );
  }
}
```

Όταν αφαιρούμε ένα πεδίο σημαίνει πως αφαιρούνται επίσης τα πεδία widget και validator.

Εκτός από την αφαίρεση των πεδίων που δεν θέλουμε να προβάλλονται μπορούμε επίσης να θέσουμε τα πεδία που μόνο θέλουμε να εμφανίζονται χρησιμοποιώντας τη μέθοδο *useFields()*:

```
// lib/form/doctrine/JobeetJobForm.class.php
class JobeetJobForm extends BaseJobeetJobForm
{
  public function configure()
  {
    $this->useFields(array('category_id', 'type', 'company', 'logo', 'url', 'position', 'location', 'description', 'how_to_apply', 'token', 'is_public', 'email'));
  }
}
```

Αλλάζουμε το προεπιλεγμένο *sfValidatorString* σε ένα *sfValidatorEmail*:

```
// lib/form/doctrine/JobeetJobForm.class.php
public function configure()
{
  // ...
  $this->validatorSchema['email'] = new sfValidatorEmail();
}
```

Είναι καλύτερο πάντα να προσθέτουμε έναν καινούριο validator στους ήδη υπάρχοντες χρησιμοποιώντας τον ειδικό validator *sfValidatorAnd*:

```
// lib/form/doctrine/JobeetJobForm.class.php
public function configure()
{
  // ...
  $this->validatorSchema['email'] = new sfValidatorAnd(array(
    $this->validatorSchema['email'],
    new sfValidatorEmail(),
  ));
}
```

Ακόμα κι αν ο τύπος του column είναι *varchar* στο schema, θέλουμε την τιμή του να είναι περιορισμένη σε μία λίστα από επιλογές: full time, part time, ή freelance.

Πρώτα ορίζουμε τις πιθανές τιμές στο *JobeetJobTable*:

```
// lib/model/doctrine/JobeetJobTable.class.php
class JobeetJobTable extends Doctrine_Table
{
  static public $types = array(
    'full-time' => 'Full time',
    'part-time' => 'Part time',
    'freelance' => 'Freelance',
  );
  public function getTypes()
  {
    return self::$types;
  }
  // ...
```

```
}
```

Έπειτα χρησιμοποιούμε το *sfWidgetFormChoise* για τον τύπο του widget:

```
$this->widgetSchema['type'] = new sfWidgetFormChoice(array(
  'choices'  => Doctrine_Core::getTable('JobeetJob')->getTypes(),
  'expanded' => true,
));
```

Το *sfWidgetFormChoice* αντιπροσωπεύει ένα choice widget το οποίο μπορεί να καταστεί από ένα διαφορετικό widget σύμφωνα με κάποιες επιλογές προσαρμογής (expanded και multiple):

- Dropdown list (<select>): *array('multiple' => false, 'expanded' => false)*
- Dropdown box (<select multiple="multiple">): *array('multiple' => true, 'expanded' => false)*
- List of radio buttons: *array('multiple' => false, 'expanded' => true)*
- List of checkboxes: *array('multiple' => true, 'expanded' => true)*

Ας αλλάξουμε τον validator έτσι ώστε να έχει περιορισμένες πιθανές επιλογές για λόγους ασφαλείας:

```
$this->validatorSchema['type'] = new sfValidatorChoice(array(
  'choices' => array_keys(Doctrine_Core::getTable('JobeetJob')->getTypes()),
));
```

Καθώς το logo column θα αποθηκεύσει το όνομα αρχείου του logo το οποίο συνδέεται με την εργασία, θα πρέπει να αλλάξουμε το widget σε μία ετικέτα αρχείου εισόδου:

```
$this->widgetSchema['logo'] = new sfWidgetFormInputFile(array(
  'label' => 'Company logo',
));
```

Για κάθε πεδίο, το symfony παράγει αυτόματα μία ετικέτα. Αυτό μπορεί να αλλαχτεί με μία επιλογή ετικέτας.

Μπορούμε επίσης να αλλάξουμε τις ετικέτες σε μία δέσμη με τη μέθοδο *setLabels()* ενός widget array:

```
$this->widgetSchema->setLabels(array(
  'category_id'    => 'Category',
  'is_public'      => 'Public?',
  'how_to_apply'   => 'How to apply?',
));
```

Χρειάζεται επίσης να αλλάξουμε τον προεπιλεγμένο validator:

```
$this->validatorSchema['logo'] = new sfValidatorFile(array(
  'required'   => false,
  'path'       => sfConfig::get('sf_upload_dir').'/jobs',
  'mime_types' => 'web_images',
));
```

Το *sfValidatorFile* είναι ενδιαφέρον καθώς κάνει κάποια πράγματα:

- Επικυρώνει το ότι το ανεβασμένο αρχείο είναι ένα είδωλο σε ένα web format(mime_types)
- Μετονομάζει το αρχείο σε κάτι το μοναδικό
- Αποθηκεύει το αρχείο σε ένα δεδομένο path
- Ενημερώνει το logo column με ένα παραγόμενο όνομα

Καθώς ο validator αποθηκεύει μόνο το όνομα στην βάση δεδομένων, ας αλλάξουμε το path που χρησιμοποιείται στην *showSuccess* template:

```
// apps/frontend/modules/job/templates/showSuccess.php

<img src="/uploads/jobs/<?php echo $job->getLogo() ?>" alt="<?php echo $job->getCompany() ?> logo" />
```

Ας προσθέσουμε ένα βοηθητικό μήνυμα για το column *is_public*:

$this->widgetSchema->setHelp('is_public', 'Whether the job can also be published on affiliate websites or not.');

Η τελική *JobeetJobForm* class έχει ως εξής:

```
// lib/form/doctrine/JobeetJobForm.class.php
class JobeetJobForm extends BaseJobeetJobForm
{
  public function configure()
  {
    unset(
      $this['created_at'], $this['updated_at'],
      $this['expires_at'], $this['is_activated']
    );
    $this->validatorSchema['email'] = new sfValidatorAnd(array(
      $this->validatorSchema['email'],
      new sfValidatorEmail(),
    ));
    $this->widgetSchema['type'] = new sfWidgetFormChoice(array(
      'choices'  => Doctrine_Core::getTable('JobeetJob')->getTypes(),
      'expanded' => true,
    ));
    $this->validatorSchema['type'] = new sfValidatorChoice(array(
      'choices' => array_keys(Doctrine_Core::getTable('JobeetJob')->getTypes()),
    ));
    $this->widgetSchema['logo'] = new sfWidgetFormInputFile(array(
      'label' => 'Company logo',
    ));
```

```
    $this->widgetSchema->setLabels(array(
      'category_id'    => 'Category',
      'is_public'      => 'Public?',
      'how_to_apply'   => 'How to apply?',
    ));
    $this->validatorSchema['logo'] = new sfValidatorFile(array(
      'required'   => false,
      'path'       => sfConfig::get('sf_upload_dir').'/jobs',
      'mime_types' => 'web_images',
    ));
    $this->widgetSchema->setHelp('is_public', 'Whether the job can also be published on affiliate websites or not.');
  }
}
```

11.4 To Form Template

Τώρα που η form class έχει προσαρμοστεί, θα πρέπει να την προβάλουμε. Το template για τη φόρμα είναι το ίδιο είτε θελήσουμε να δημιουργήσουμε καινούρια εργασία είτε να επεξεργαστούμε μία ήδη υπάρχουσα. Στην πραγματικότητα και τα δύο τα templates, το *newSuccess.php* και το *editSuccess.php* είναι σχεδόν όμοια:

```
<!-- apps/frontend/modules/job/templates/newSuccess.php -->
<?php use_stylesheet('job.css') ?>
<h1>Post a Job</h1>
<?php include_partial('form', array('form' => $form)) ?>
```

Η φόρμα από μόνη της καθίσταται στο partial *the_form*. Αντικαθιστούμε το περιεχόμενο του *generated_form* partial με τον παρακάτω κώδικα:

```
<!-- apps/frontend/modules/job/templates/_form.php -->
<?php use_stylesheets_for_form($form) ?>
<?php use_javascripts_for_form($form) ?>
<?php echo form_tag_for($form, '@job') ?>
```

```
<table id="job_form">
  <tfoot>
    <tr>
      <td colspan="2">
        <input type="submit" value="Preview your job" />
      </td>
    </tr>
  </tfoot>
  <tbody>
    <?php echo $form ?>
  </tbody>
</table>
</form>
```

11.5 Η Form Action

Έχουμε τώρα μία form class κι ένα template το οποίο την καθιστά. Τώρα είναι η ώρα να το κάνουμε να λειτουργήσει με κάποιες ενέργειες.

Η φόρμα εργασίας διαχειρίζεται από πέντε μεθόδους στο module εργασίας:

- **new:** Προβάλει μία λευκή φόρμα για να δημιουργήσει μία νέα εργασία
- **edit:** Προβάλει μία φόρμα για την επεξεργασία μιας υπάρχουσας εργασίας
- **create:** Δημιουργεί μία νέα εργασία με τις τιμές που έχει υποβάλλει ο χρήστης
- **update:** Ενημερώνει μία υπάρχουσα εργασία με τις τιμές που έχει υποβάλλει ο χρήστης
- **processForm:** Καλείται από το create και το update, επεξεργάζεται τη φόρμα

Όλες οι φόρμες έχουν τον παρακάτω κύκλο ζωής:

Μπορούμε να απλοποιήσουμε τον κώδικα για τη διαχείριση της φόρμας:

```
// apps/frontend/modules/job/actions/actions.class.php
public function executeNew(sfWebRequest $request)
{
```

```
  $this->form = new JobeetJobForm();
}
public function executeCreate(sfWebRequest $request)
{
  $this->form = new JobeetJobForm();
  $this->processForm($request, $this->form);
  $this->setTemplate('new');
}
public function executeEdit(sfWebRequest $request)
{
  $this->form = new JobeetJobForm($this->getRoute()->getObject());
}
public function executeUpdate(sfWebRequest $request)
{
  $this->form = new JobeetJobForm($this->getRoute()->getObject());
  $this->processForm($request, $this->form);
  $this->setTemplate('edit');
}
public function executeDelete(sfWebRequest $request)
{
  $request->checkCSRFProtection();
  $job = $this->getRoute()->getObject();
  $job->delete();
  $this->redirect('job/index');
}
protected function processForm(sfWebRequest $request, sfForm $form)
```

```
{
  $form->bind(
    $request->getParameter($form->getName()),
    $request->getFiles($form->getName())
  );
  if ($form->isValid())
  {
    $job = $form->save();
    $this->redirect('job_show', $job);
  }
}
```

Αλλάζουμε τη μέθοδο *executeNew()* για να ορίσουμε το full-time σαν προεπιλεγμένη τιμή για τον τύπο της στήλης:

```
// apps/frontend/modules/job/actions/actions.class.php
public function executeNew(sfWebRequest $request)
{
  $job = new JobeetJob();
  $job->setType('full-time');
  $this->form = new JobeetJobForm($job);
}
```

11.6.Προστατεύοντας τη Form Θέσης Εργασίας με ένα Token

Ενημερώνουμε τη μέθοδο *save()* του *JobeetJob* και προσθέτουμε τη λογική που παράγει το token πριν αποθηκευτεί η καινούρια εργασία:

```
// lib/model/doctrine/JobeetJob.class.php
public function save(Doctrine_Connection $conn = null)
{
  // ...
```

```
  if (!$this->getToken())
  {
    $this->setToken(sha1($this->getEmail().rand(11111, 99999)));
  }
  return parent::save($conn);
}
```

Τώρα μπορούμε να αφαιρέσουμε το πεδίο token από τη φόρμα:

```
// lib/form/doctrine/JobeetJobForm.class.php
class JobeetJobForm extends BaseJobeetJobForm
{
  public function configure()
  {
    unset(
      $this['created_at'], $this['updated_at'],
      $this['expires_at'], $this['is_activated'],
      $this['token']
    );
    // ...
  }
  // ...
}
```

Μία route *sfDoctrineRouteCollection* παράγει URLs με το πρωτεύον κλειδί, αλλά μπορεί να αλλαχτεί σε οποιαδήποτε μοναδική στήλη περνώντας την επιλογή στήλης:

```
# apps/frontend/config/~routing|Routing~.yml
job:
  class:        sfDoctrineRouteCollection
  options:      { model: JobeetJob, column: token }
  requirements: { token: \w+ }
```

Πλέον η διαδρομή για να επεξεργαστούμε μία εργασία γίνεται από το παρακάτω pattern:

http://www.jobeet.com.localhost/job/TOKEN/edit

Θα πρέπει επίσης να αλλάξουμε τον σύνδεσμο "Edit" στο template *showSuccess*:

```
<!-- apps/frontend/modules/job/templates/showSuccess.php -->

<a href="<?php echo url_for('job_edit', $job) ?>">Edit</a>
```

11.7 Η Σελίδα Προεπισκόπησης

Η σελίδα της προεπισκόπησης είναι ίδια με την σελίδα της εργασίας. Χάρη στο routing, όταν ένας χρήστης εισέρχεται με το σωστό token, θα είναι προσβάσιμο στην παράμετρο του token request.

Όταν ένας χρήστης εισέρχεται με ένα tokenized URL, θα προσθέσουμε ένα admin bar στην κορυφή. Στην αρχή του template *showSuccess*, προσθέτουμε ένα partial για να φιλοξενήσει το admin bar και αφαιρούμε τον σύνδεσμο της επεξεργασίας στο τέλος:

```
<!-- apps/frontend/modules/job/templates/showSuccess.php -->

<?php if ($sf_request->getParameter('token') == $job->getToken()): ?>

  <?php include_partial('job/admin', array('job' => $job)) ?>

<?php endif ?>
```

Έπειτα, δημιουργούμε το *_admin* partial:

```
<!-- apps/frontend/modules/job/templates/_admin.php -->

<div id="job_actions">

  <h3>Admin</h3>

  <ul>

    <?php if (!$job->getIsActivated()): ?>

      <li><?php echo link_to('Edit', 'job_edit', $job) ?></li>

      <li><?php echo link_to('Publish', 'job_edit', $job) ?></li>

    <?php endif ?>
```

```
    <li><?php echo link_to('Delete', 'job_delete', $job, array('method' => 'delete', 'confirm' => 'Are you sure?')) ?></li>

    <?php if ($job->getIsActivated()): ?>

      <li<?php $job->expiresSoon() and print ' class="expires_soon"' ?>>

        <?php if ($job->isExpired()): ?>

          Expired

        <?php else: ?>

          Expires in <strong><?php echo $job->getDaysBeforeExpires() ?></strong> days

        <?php endif ?>

        <?php if ($job->expiresSoon()): ?>

         - <a href="">Extend</a> for another <?php echo sfConfig::get('app_active_days') ?> days

        <?php endif ?>

      </li>

    <?php else: ?>

      <li>

        [Bookmark this <?php echo link_to('URL', 'job_show', $job, true) ?> to manage this job in the future.]

      </li>

    <?php endif ?>

  </ul>

</div>
```

Για να κάνουμε το template πιο ευανάγνωστο, προσθέσαμε ένα σωρό από μεθόδους συντόμευσης στο *JobeetJob* class:

```
// lib/model/doctrine/JobeetJob.class.php

public function getTypeName()

{

  $types = Doctrine_Core::getTable('JobeetJob')->getTypes();

  return $this->getType() ? $types[$this->getType()] : '';

}

public function isExpired()

{

  return $this->getDaysBeforeExpires() < 0;

}

public function expiresSoon()

{

  return $this->getDaysBeforeExpires() < 5;

}

public function getDaysBeforeExpires()

{

  return ceil(($this->getDateTimeObject('expires_at')->format('U') - time()) / 86400);

}
```

Το admin bar προβάλει τις διαφορετικές ενέργειες βασιζόμενο στην κατάσταση της εργασίας:

11.8.Ενεργοποίηση και Δημοσίευση της Θέσης Εργασίας

Υπάρχει ένας σύνδεσμος για να δημοσιοποιήσουμε την εργασίας. Ο σύνδεσμος χρειάζεται να αλλαχτεί για να δείχνει σε ένα καινούριο publish action. Αντί να δημιουργήσουμε μία καινούρια διαδρομή μπορούμε απλά να ρυθμίσουμε την ήδη υπάρχουσα job route:

```
# apps/frontend/config/routing.yml
job:
  class:   sfDoctrineRouteCollection
  options:
    model:         JobeetJob
    column:        token
    object_actions: { publish: put }
  requirements:
    token: \w+
```

Το *object_actions* παίρνει έναν πίνακα με επιπρόσθετες ενέργειες για το δεδομένο αντικείμενο. Τώρα μπορούμε να αλλάξουμε τον σύνδεσμο του συνδέσμου "Publish":

```
<!-- apps/frontend/modules/job/templates/_admin.php -->
<li>
  <?php echo link_to('Publish', 'job_publish', $job, array('method' => 'put')) ?>
</li>
```

Το τελευταίο βήμα είναι να δημιουργήσουμε την *publish* action:

```
// apps/frontend/modules/job/actions/actions.class.php
public function executePublish(sfWebRequest $request)
{
  $request->checkCSRFProtection();
   $job = $this->getRoute()->getObject();
  $job->publish();
```

```
    $this->getUser()->setFlash('notice', sprintf('Your job is now online for %s days.', sfConfig::get('app_active_days')));

    $this->redirect('job_show_user', $job);

  }
```

Παρατηρούμε ότι ο σύνδεσμος "Publish" παρέχεται με την HTTP put method. Για να εξομοιώσουμε την put method, ο σύνδεσμος μετατρέπεται αυτόματα σε form όταν κάνουμε κλικ σε αυτόν.

Επειδή έχουμε ενεργοποιήσει την προστασία CSRF, ο helper *link_to()* ενσωματώνει ένα CSRF token μέσα στον σύνδεσμο και η μέθοδος *checkCSRFProtection()* του request object ελέγχει την εγκυρότητα του στην υποβολή.

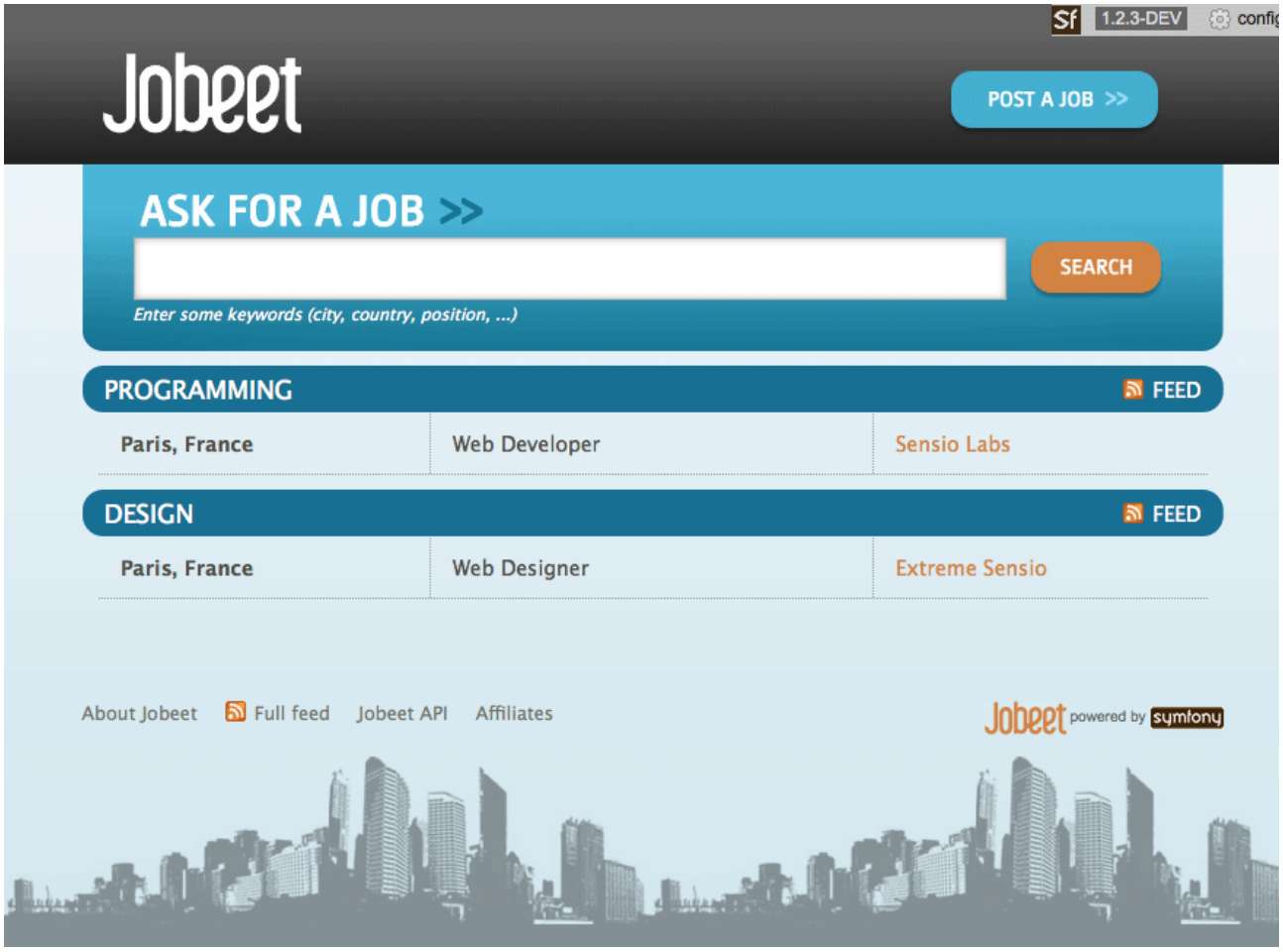
Η μέθοδος *executePublish()* χρησιμοποιεί μία καινούρια μέθοδο *publish()* η οποία μπορεί να οριστεί όπως παρακάτω:

```
  // lib/model/doctrine/JobeetJob.class.php

  public function publish()

  {

    $this->setIsActivated(true);

    $this->save();

  }
```

ώρα μπορούμε να ελέγξουμε το χαρακτηριστικό της δημοσίευσης στον browser.

Πρέπει να φτιάξουμε και κάτι άλλο. Η μη ενεργοποιημένες εργασίες δεν θα πρέπει να είναι προσβάσιμες, το οποίο σημαίνει ότι δεν πρέπει να εμφανίζονται στην αρχική σελίδα του Jobeet, και δεν θα πρέπει να είναι προσβάσιμες από το URL τους. Καθώς έχουμε δημιουργήσει μία μέθοδο *addActiveJobsQuery()* για να περιορίσουμε ένα *Doctrine_Query* σε ενεργές εργασίες, μπορούμε απλά να το επεξεργαστούμε και να προσθέσουμε νέες απαιτήσεις στο τέλος:

```
  // lib/model/doctrine/JobeetJobTable.class.php
  public function addActiveJobsQuery(Doctrine_Query $q = null)
  {
    // ...
    $q->andWhere($alias . '.is_activated = ?', 1);
    return $q;
  }
```



Εικόνα 18 :Η τελικη σελιδα του Jobeet.

12 Συστήματα διαχείρισης Περιεχομένου – Drupal

Παρακάτω παρουσιάζεται μια σύντομη περιγραφή του συστήματος διαχείρισης περιεχομένου (Drupal), όπου θα χρησιμοποιηθεί στο υλοποιημένο σύστημα. Πιο συγκεκριμένα παρουσιάζονται οι τεχνολογίες, η μορφή των αρχείων που περιλαμβάνει και οι διάφοροι εννοιολογικοί όροι που χρησιμοποιεί το Drupal. Τέτοιοι όροι είναι οι nodes, hooks, blocks και themes.

12.1 Εισαγωγή στο Σύστημα Διαχείρισης Περιεχομένου (Content Management System - CMS)

Η αυτοματοποίηση των διαδικασιών δημιουργίας των πληροφοριών, που αποτελούν το περιεχόμενο του διαδικτύου, δημοσίευσης τους και παρουσίασης τους συνιστά το επόμενο βήμα στις προηγούμενες απαιτήσεις. Ο μεγάλος όγκος της πληροφορίας σε συνδυασμό με την απαιτούμενη τεχνική γνώση δεν επέτρεπε στους οργανισμούς να επιτύχουν την ισορροπία ανάμεσα σε ένα εύχρηστο και ελκυστικό περιβάλλον παρουσίασης και σε ένα συνεχώς ανανεώσιμο περιεχόμενο, που θα τους εξασφάλιζε μία σταθερή βάση επισκεψιμότητας στη ιστοσελίδα τους. Όταν δε έμπαινε και ο παράγοντας του ελέγχου της ροής της πληροφορίας από πολλαπλά άτομα, η κατάσταση γινόταν ακόμη πιο δύσκολη. Αποτέλεσμα ήταν η δημιουργία μεγάλων ιστοχώρων με καλή σχεδίαση, αλλά ξεπερασμένο χρονικά περιεχόμενο, ή με κακή σχεδίαση χωρίς μεγάλα περιθώρια ευελιξίας, αλλά με υπέρ-ανανεωμένο περιεχόμενο. Η έλλειψη τεχνικών γνώσεων από τα στελέχη του οργανισμού οδηγούσε τις επιχειρήσεις σε δημιουργία γραφείων ή σε εκμίσθωση ειδικευμένων εταιριών για την διατήρηση των ιστοσελίδων τους. Εκτός από το φανερό κόστος της κίνησης αυτής, η λύση της δημιουργίας ενός ειδικού γραφείου παρουσίαζε σημαντικά προβλήματα. Λίγα άτομα με τεχνικές γνώσεις επιμερίζονταν τον τεράστιο όγκο των πληροφοριών του ιστοχώρου, ενώ επιμερίζονταν ταυτόχρονα και όλες τις λειτουργίες, από την εύρεση του περιεχομένου, την επεξεργασία του, την δημοσίευση του και την αποθήκευση του για μελλοντική χρήση. Συνέπεια ήταν να μην μπορεί το γραφείο πολλές φορές να διαχειριστεί τον τεράστιο όγκο των πληροφοριών, αυτές να δημοσιεύονται με καθυστέρηση και να μην υπάρχει πολυφωνία και πλούτος περιεχομένου. Ιδιαίτερα, αν η ιστοσελίδα ήταν μεγάλη, τότε πολλές φορές το περιεχόμενο της διαμοιράζονταν σε πολλά γραφεία, με αποτέλεσμα να έχουμε έναν ιστόχωρο με έλλειψη διασύνδεσης και πολλές φορές χωρίς καμία συνοχή. Το τοπίο λοιπόν ήταν γόνιμο για την δημιουργία των ηλεκτρονικών εργαλείων, που θα έδιναν λύση στο πρόβλημα της επιτυχημένης ηλεκτρονικής παρουσίας των οργανισμών στο διαδίκτυο. Τα CMS επιτρέπουν στους οργανισμούς να δημιουργούν, αλλά και να εισάγουν έτοιμο πολυμεσικό υλικό. Να πιστοποιούν τους χρήστες του συστήματος και να επιμερίζουν ξεχωριστούς ρόλους στον καθένα στον κύκλο της λειτουργίας τους. Επίσης, επιτρέπουν τον προσδιορισμό εργασιών ροής του περιεχομένου, συχνά σε συνδυασμό με την λειτουργία των ειδοποιήσεων συμβάντων, που επιτρέπουν στους διαχειριστές του περιεχομένου να ειδοποιούνται για οποιαδήποτε αλλαγή. Τα CMS δίνουν ακόμη στους χρήστες την δυνατότητα να εντοπίζουν και να διαχειρίζονται πολλαπλές εκδόσεις ενός μόνο αρχείου περιεχομένου, να το αποθηκεύουν σε μία ξεχωριστή βάση δεδομένων, ενώ ταυτόχρονα προσφέρουν την δυνατότητα ευρετηρίου, διευκολύνοντας τον έλεγχο και την επαναφορά παλαιότερου υλικού της ιστοσελίδας. Το βασικότερο, όμως, χαρακτηριστικό που προσφέρουν είναι η δυνατότητα διαχωρισμού του περιεχομένου από την παρουσίαση της ιστοσελίδας.

12.1.1 Ιστορικά Στοιχειά CMS

Ο όρος Content Management Systems (CMS) αρχικά χρησιμοποιήθηκε για να δηλώσει τα συστήματα δημοσίευσης ιστοσελίδων στο Διαδίκτυο γενικότερα, καθώς επίσης και για τα προγράμματα διαχείρισης περιεχομένου ευρύτερα. Τα πρώτα Συστήματα Διαχείρισης Περιεχομένου αναπτύσσονταν εσωτερικά στους οργανισμούς από το τεχνικό τους τμήμα, καθώς ήταν απαραίτητα για την δημοσιοποίηση ενός μεγάλου όγκου υλικού, από ηλεκτρονικά περιοδικά και εφημερίδες των επιχειρήσεων μέχρι την δημοσίευση και αποστολή των εταιρικών newsletters. Το 1995, η εταιρεία CNET αποφάσισε να επεκτείνει το εσωτερικό σύστημα διαχείρισης περιεχομένου, που χρησιμοποιούσε, για την δημοσίευση ηλεκτρονικού υλικού και να δημιουργήσει την ξεχωριστή εταιρεία Vignette. Στόχος της ήταν να εκμεταλλευτεί εμπορικά τα CMS. Στην διάρκεια της δεκαετίας που ακολούθησε η αγορά εξελίχτηκε και σήμερα υπολογίζεται ότι υπάρχουν περί τις 500 εφαρμογές CMS κάθε είδους. Η αγορά εξελίσσεται συνεχώς αναγκάζοντας τους οργανισμούς να ενημερώνονται συνεχώς για τις εξελίξεις και να μετακινούνται στα συστήματα, που πλέον καλύπτουν ακόμη περισσότερο τις ανάγκες τους.

12.1.2 Τι είναι το σύστημα διαχείρισης περιεχομένου CMS

Το CMS είναι ακρωνύμιο για το Content Management System (Σύστημα διαχείρισης Περιεχομένου). Ο όρος Content Management Systems (CMS, Συστήματα διαχείρισης Περιεχομένου) αναφέρεται στις εφαρμογές που επιτρέπουν στον πελάτη να διαχειρίζεται το δικτυακό του περιεχόμενο, όπως κείμενα, εικόνες, πίνακες κλπ., με εύκολο τρόπο, συνήθως παρόμοιο με αυτόν της χρήσης ενός κειμενογράφου. Οι εφαρμογές διαχείρισης περιεχομένου επιτρέπουν την αλλαγή του περιεχόμενου χωρίς να είναι απαραίτητες ειδικές γνώσεις σχετικές με τη δημιουργία ιστοσελίδων ή γραφικών, καθώς συνήθως τα κείμενα γράφονται μέσω κάποιων online WYSIWYG ("What You See Is What You Get") html editors, ειδικών δηλαδή κειμενογράφων, παρόμοιων με το MS Word, που επιτρέπουν τη μορφοποίηση των κειμένων όποτε υπάρχει ανάγκη. Οι αλλαγές του site μπορούν να γίνουν από οποιονδήποτε υπολογιστή που είναι συνδεδεμένος στο διαδίκτυο, χωρίς να χρειάζεται να έχει εγκατεστημένα ειδικά προγράμματα επεξεργασίας ιστοσελίδων, γραφικών κλπ. Μέσω ενός απλού φυλλομετρητή ιστοσελίδων (browser), ο χρήστης μπορεί να συντάξει ένα κείμενο και να ενημερώσει άμεσα το δικτυακό του τόπο. Με άλλα λόγια είναι ένα «αντικείμενο» υψηλής συμπερίληψης. Υλοποιείται με την λογική του μοντέλου «WYSIWYG»(What You See Is What You Get – Ότι βλέπεις είναι ότι παίρνεις), δηλαδή μια αντικειμενοστραφή λογική όπου το «αντικείμενο» είναι στο υψηλότερο επίπεδο. Και όπου η υλοποίηση είναι μια εικονική λογική. Με ένα CMS, είναι πολύ εύκολη η λειτουργία δημοψηφισμάτων μέσω του website, το στήσιμο ενός forum, η δημιουργία ενός blog, η χρήση news feeds, η δημιουργία βάσης δεδομένων με εικόνες, αρχεία, κτλ. Αν λοιπόν ο χρήστης θέλει να δημιουργήσει ένα δυναμικό, εύκολα αναβαθμίσιμο και μοντέρνο website, η εύκολη και σίγουρη λύση είναι ένα πρόγραμμα CMS.

12.1.3 Τα διαθέσιμα Web CMS

Τα διαθέσιμα Web CMS χωρίζονται σε 3 κατηγορίες:

- CMS κλειστού κώδικα.
- CMS ανοιχτού κώδικα.
- Παραμετροποιημένα CMS βασισμένα σε πλαίσια ανοιχτού κώδικα.

12.1.4 CMS ανοικτού κώδικα

Ένας παράγοντας που έχει συμβάλλει αποφασίστηκα στην άνοδο της δημοτικότητας και της ευχρηστίας των CMS είναι η πρόοδος που σημειώνεται τα τελευταία χρόνια στο κίνημα ανάπτυξης λογισμικού ανοιχτού κώδικα

Το 1998, η εταιρία Netscape αντιτάχθηκε για πρώτη φορά στην καθιερωμένη πρακτική της ανάπτυξης λογισμικού, καθιστώντας τον πηγαίο κώδικα της δικής της εφαρμογής περιήγησης ελεύθερα διαθέσιμο στους πάντες. Αυτή η ενέργεια αποτέλεσε ορόσημο στην εξέλιξη του λογισμικού, δημιουργώντας ένα φιλοσοφικό κίνημα, βάσει του οποίου μια εφαρμογή λογισμικού αναπτύσσεται από μια μεγάλη κοινότητα δημιουργών και διατίθεται ελεύθερα σε όλο τον κόσμο. Το σύνολο των σημαντικότερων λογισμικών ανοιχτού κώδικα αποκαλείται συλλογικά LAPMP ένα ακρωνύμιο που σημαίνει τα εξής :

- Linux
- Apache
- MySQL
- Php

Στις εφαρμογές ανοικτού κώδικα επιτρέπεται η πρόσβαση και η αλλαγή του πηγαίου κώδικα, που σημαίνει ότι μπορούμε να επεξεργαστούμε τον κώδικα και να τον προσαρμόσουμε σύμφωνα με τις ανάγκες μας. Το κόστος της εφαρμογής μειώνεται δραματικά καθώς στις περισσότερες περιπτώσεις ολόκληρη η εφαρμογή καθώς και πρόσθετα (plugins) τα οποία δημιουργεί και προσφέρει η κοινότητα υποστήριξης (που συνήθως υπάρχει για τα συστήματα ανοικτού κώδικα) βρίσκονται στο διαδίκτυο.

12.1.5 Τα πιο δημοφιλή CMS ανοιχτού κώδικα

Το τελευταίο διάστημα διεξάγεται μια μεγάλη "μάχη" ανάμεσα στα συστήματα διαχείρισης περιεχομένου (CMS) ανοιχτού κώδικα, με πολλές αξιόλογες προτάσεις. Η επιλογή δεν είναι εύκολη, κάθε εφαρμογή έχει πλεονεκτήματα και μειονεκτήματα, ενώ μεγάλο ρόλο στην τελική απόφαση παίζουν οι απαιτούμενες προδιαγραφές των υπό κατασκευή ιστότοπων. Σύμφωνα με έρευνες των τελευταίων χρόνων τα τρία συστήματα διαχείρισης περιεχομένου που κυριαρχούν στην αγορά σήμερα είναι το **Drupal,** το WordPress και το Joomla.

Εμείς θα ασχοληθούμε με την ανάλυση με σκοπό τον σχεδιασμό για τη δημιουργία ιστοσελίδας με χρήση του Drupal.

12.1.6 Δυνατότητες και χαρακτηριστικά ενός CMS

- Παρέχει τη δυνατότητα της διαχείρισης – συντήρησης ενός ιστότοπου από απλούς χειριστές χωρίς την απαίτηση για εμπλοκή ειδικού τεχνικού προσωπικού.
- Παρέχει δηλαδή την ευκαιρία ο διαχειριστής του να επικεντρωθεί στο περιεχόμενο και όχι στην τεχνολογία
- Αυτοματοποιεί εργασίες ρουτίνας π.χ. εφαρμόζει την ίδια μορφοποίηση (layout) σε όλες τις ιστοσελίδες. Οι επιλογές (menus) και γενικότερα η πλοήγηση αναπαράγεται επίσης αυτόματα.
- Παρέχει απλά εργαλεία (επεξεργαστές σαν το Word) για τη δημιουργία του περιεχομένου.
- Παρέχει τη δυνατότητα διαχείρισης της δομής του ιστότοπου, της εμφάνισης των δημοσιευμένων σελίδων καθώς και της πλοήγησης σε αυτές.

Τα χαρακτηριστικά των CMS αφορούν στη σύνθεση κάθε τέτοιου τύπου λογισμικού. Υπάρχουν πολλαπλά χαρακτηριστικά στα πιο απλά πακέτα, ενώ έχουν αναδειχτεί ακόμη και πιο πλούσιες σε χαρακτηριστικά λύσεις. Σημαντικό είναι σε αυτό το σημείο να τονιστεί, ότι τα open-source CMS, των

οποίων ο κώδικας προγραμματισμού διατίθεται δωρεάν στο διαδίκτυο, έχουν θεωρητικά άπειρες δυνατότητες βελτίωσης, σε σχέση με τα εμπορικά (commercial), τα οποία έχουν κάποιο κόστος και η βελτίωση των χαρακτηριστικών τους μπορεί να γίνει μόνο από την ίδια την εταιρεία δημιουργίας τους

12.1.7 Πλεονεκτήματα ενός CMS (Content Management System)

- **Αυτοματοποιημένα Πρότυπα:**

Δημιουργία προτύπων (templates) συνήθως σε μορφή HTML ή XML τα οποία μπορούν εύκολα να εφαρμοστούν σε νέα και υπάρχοντα περιεχόμενα, επιτρέποντας στην εμφάνιση όλου του περιεχόμενου να αλλάξει από μια κεντρική θέση.

- **Εύκολα Επεξεργάσιμο Περιεχόμενο**:

Εφόσον το περιεχόμενο είναι διαχωρισμένο από την οπτική παρουσίαση της ιστοσελίδας, συνήθως γίνεται πιο εύκολο να το επεξεργαστείς και να το διαχειριστείς. Τα περισσότερα Συστήματα διαχείρισης Περιεχομένου για sites (WCMS) περιλαμβάνουν εργαλεία επεξεργασίας WYSIWYG (What You See Is What You Get) επιτρέποντας σε μη-τεχνικά άτομα να δημιουργήσουν και να επεξεργαστούν περιεχόμενο.

- **Χαρακτηριστικό κλιμακωτών συνόλων**:

Τα περισσότερα Συστήματα διαχείρισης Περιεχομένου για sites (WCMS) περιλαμβάνουν plug-ins ή modules (μονάδες) τα οποία μπορούν εύκολα να εγκατασταθούν για να επεκτείνουν την λειτουργικότητα της υπάρχουσας ιστοσελίδας.

- **Αναβάθμιση προτύπων μέσω Web**:

Ένα ενεργό Σύστημα διαχείρισης Περιεχομένου για sites(WCMS), συνήθως λαμβάνει ενημερώσεις οι οποίες εμπεριέχουν νέα χαρακτηριστικά και κρατούν το σύστημα ενημερωμένο στα τρέχοντα πρότυπα.

- **Διαχείριση της ροής εργασίας(workflow):**

Workflow είναι η διαδικασία της δημιουργίας κύκλων των αλληλοδιαδοχικών ή παράλληλων έργων που πρέπει να εκπληρωθούν σε ένα Σύστημα διαχείρισης Περιεχομένου (CMS). Για παράδειγμα ένας δημιουργός περιεχομένου μπορεί να υποβάλλει μια «ιστορία»(story), αλλά δεν θα δημοσιευθεί μέχρι ο συντάκτης αντιγράφων να την «καθαρίσει» και ο αρχισυντάκτης να την εγκρίνει.

- **Αντιπροσωπεία:**

Μερικά Συστήματα διαχείρισης Περιεχομένου (CMS) επιτρέπουν σε διάφορες ομάδες χρηστών να έχουν περιορισμένα δικαιώματα πάνω σε συγκεκριμένα περιεχόμενα μιας ιστοσελίδας, εξαπλώνοντας την ευθύνη της διαχείρισης περιεχομένου.

- **Διαχείριση των εγγράφων:**

Ένα Σύστημα διαχείρισης Περιεχομένου μπορεί να παρέχει ένα μέσο για την διαχείριση του κύκλου ζωής ενός εγγράφου από την αρχική στιγμή δημιουργίας του, μέσα από τις αναθεωρήσεις των εκδόσεων, τη δημοσίευση, το αρχείο, και την καταστροφή του εγγράφου.

- **Virtualization (εικονικότητα) του περιεχομένου**:

Ένα Σύστημα διαχείρισης Περιεχομένου μπορεί να αποτελέσει το μέσο που να επιτρέπει σε κάθε χρήστη να εργάζεται μέσα σε ένα εικονικό αντίγραφο μιας πλήρους ιστοσελίδας, σύνολα εγγράφων και/ή μιας βάσης κώδικα. Αυτό επιτρέπει στις αλλαγές σε πολλούς αλληλοεξαρτώμενους πόρους να είναι εμφανείς και/ή να εκτελούνται σε πλαίσιο πριν την υποβολή τους.

13 Γενική περιγραφή του Drupal

13.1 Τι είναι το Drupal

Το Drupal είναι ένα αρθρωτό σύστημα διαχείρισης περιεχομένου (Content Management System, CMS) ανοικτού/ελεύθερου λογισμικού, γραμμένο στη γλώσσα προγραμματισμού PHP. Το Drupal, όπως πολλά σύγχρονα CMS, επιτρέπει στο διαχειριστή συστήματος να οργανώνει το περιεχόμενο, να προσαρμόζει την παρουσίαση, να αυτοματοποιεί διαχειριστικές εργασίες και να διαχειρίζεται τους επισκέπτες του ιστότοπου και αυτούς που συνεισφέρουν. Παρόλο που υπάρχει μια πολύπλοκη προγραμματιστική διεπαφή, οι περισσότερες εργασίες μπορούν να γίνουν με λίγο ή και καθόλου προγραμματισμό. Το Drupal ορισμένες φορές περιγράφεται ως "υποδομή για εφαρμογές ιστού", καθώς οι δυνατότητές του προχωρούν παραπέρα από τη διαχείριση περιεχομένου, επιτρέποντας ένα μεγάλο εύρος υπηρεσιών και συναλλαγών. Το Drupal μπορεί να εκτελεστεί σε διάφορες πλατφόρμες, συμπεριλαμβανομένων των λειτουργικών συστημάτων Windows, Mac OS X, Linux, FreeBSD, ή οποιασδήποτε πλατφόρμα που υποστηρίζει είτε το διακομιστή ιστοσελίδων Apache HTTP Server (έκδοση 1.3+), είτε το Internet Information Services (έκδοση IIS5+), καθώς επίσης και τη γλώσσα προγραμματισμού PHP (έκδοση 4.3.3+). Το Drupal απαιτεί μια βάση δεδομένων όπως η MySQL και η PostgreSQL για την αποθήκευση του περιεχομένου και των ρυθμίσεών του.

13.1.1 Πλεονεκτήματα Drupal

- **Εύκολο στην χρήση και την ενημέρωση:**

Μπορείτε να ενημερώσετε την ιστοσελίδα σας όποτε εσείς το επιθυμήσετε, όπου το επιθυμήσετε, χωρίς ανησυχία. Δεν χρειάζεται να ζητήσετε από τον σχεδιαστή της ιστοσελίδας να το κάνει για εσάς. Χρησιμοποιώντας το κατάλληλα εναρμονισμένο editor (συντάκτη) WYSIWYG (What You See Is What You Get) όπως για παράδειγμα τον TinyMCE, μπορείτε ακόμα και να επικολλήσετε κείμενο από το Word και να αφαιρέσετε όλους εκείνους τους περίεργους χαρακτήρες που συνήθως κολλάνε από το MS Word.

- **Μηδενικό κόστος:**

Το Drupal είναι λογισμικό ανοιχτού κώδικα, έτσι δεν χρειάζεται να καταβάλετε έξοδα ούτε για χορήγηση αδειών αλλά ούτε και για την ανάπτυξη της ιστοσελίδας σας.

- **Αξιόπιστο και ασφαλές:**

Το Drupal έχει μια εκτεταμένη και ενεργή κοινότητα που το υποστηρίζει. Βελτιώνεται συνεχώς και υπόκειται σε εκτεταμένο έλεγχο, έτσι μπορείτε να βασιστείτε πάνω του, είναι στέρεο σαν βράχος!

- **Φιλικό με μηχανές αναζητήσεις**:

Μπορεί να διαμορφωθεί για φιλικές διευθύνσεις (URLs). Το παραγόμενο περιεχόμενο σχεδιάζεται έτσι ώστε να συμμορφώνεται με τους κανονισμούς κάτι που δεν βοηθάει στην ώθηση των ταξινομήσεων της μηχανής αναζήτησης αλλά είναι προσιτό.

- **Βασίζεται σε μονάδες και είναι επεκτάσιμο:**

Μπορείτε να προσθέσετε έξτρα λειτουργίες όπως τα μπλοκ, τα φόρουμ, το ηλεκτρονικό εμπόριο, ακόμα και ημερολόγιο αν το επιθυμείτε. Υπάρχουν πλήθη από μονάδες τρίτων (third party) από τις οποίες μπορείτε να επιλέξετε όσες θέλετε για την επέκταση της Drupal ιστοσελίδας σας.

- **Έλεγχος περιεχομένου :**

Μπορεί κάνεις να ρυθμίσει το Drupal ώστε να αποθηκεύει κάθε αλλαγή που γίνεται στο περιεχόμενο όποτε αυτός το επεξεργάζεται. Αυτό σημαίνει ότι μπορεί να πάει κάνεις βήματα πίσω για να δει ή να επανέλθει σε μια παλιά έκδοση του περιεχομένου αν θέλει.

- **Κατηγοριοποίηση του περιεχομένου :**

Το Drupal έχει ένα ισχυρό σύστημα ταξινόμησης (κατηγοριοποίησης) του περιεχομένου. Κάθε κατηγορία περιεχομένου μπορεί να περιορίζεται σε ορισμένους τύπους περιεχομένου που περιλαμβάνει.

- **Διαχείριση χρηστών:**

Το Drupal έχει σχεδιαστεί για χρήση από κοινότητες χρηστών οπότε και έχει ένα ισχυρό σύστημα κατανομής και διαχείρισης χρηστών και ρόλων σ' αυτούς.

- **Προσαρμοσμένο περιεχόμενο:**

Μπορείτε να χρησιμοποιήσετε το Content Construction Kit (CCK) και τα View Models για τη δημιουργία νέων ειδών περιεχομένου, χωρίς να ξέρετε να γράφετε κώδικα. Μερικά παραδείγματα των "τύπων περιεχομένου" είναι τα Blogs, ειδήσεις, φόρουμ, οδηγοί χρήσεως, μικρές αγγελίες, podcasts.

- **Εξαιρετική βοήθεια και Τεκμηρίωση (documentation):**

Περιλαμβάνει τα επίσημα εγχειρίδια, πολλά tutorials, blogs, videos, και podcasts. Υπάρχει ακόμη και μια Drupal Dojo κοινότητα όπου μπορείτε να μάθετε πώς μπορείτε να γίνετε ninja Drupal.

- **PHP Template:**

Το Drupal χρησιμοποιεί την PHP Template μηχανή, η οποία δεν απαιτεί καμία γνώση PHP.

- **Μεγάλη Κοινότητα χρηστών :**

Με τόσες σημαντικές τοποθεσίες χρηστών του Drupal, δεν είναι εύκολο να φύγεις σύντομα. Το φόρουμ του Drupal είναι υψηλής δραστικότητας και είναι ένα εξαιρετικό μέρος για να πάρετε απαντήσεις στις ερωτήσεις σας σχετικά με το Drupal. Μπορείτε επίσης να βρείτε βοήθεια για το

Drupal σε κανάλια συνομιλίας IRC #drupal-support και #drupaldojo. Τέλος υπάρχουν και τα Drupal Groups.

- Είναι εφαρμογή ανοικτού κώδικα, δηλαδή εφαρμογή ασφαλής, διαρκώς εξελισσόμενη και με μηδενικό κόστος απόκτησης.
- Έχει Ομάδα Ανάπτυξης πολυβραβευμένη για τα προϊόντα που έχει παράξει το προηγούμενο διάστημα.
- Έχει τη δυνατότητα να λειτουργήσει με ελληνικό περιβάλλον διαχείρισης
- Διαθέτει μια πλήρη γκάμα από δωρεάν, αλλά και εμπορικές, πρόσθετες εφαρμογές (addons, components, modules, bots κλπ), που δίνουν τη δυνατότητα να δώσουμε ακριβώς το χαρακτήρα που θέλουμε εμείς στο δικτυακό μας τόπο.
- Προσθήκη περιεχομένου στον ιστότοπο μας από οποιονδήποτε υπολογιστή διαθέτει σύνδεση στο διαδίκτυο.
- Υπάρχει η δυνατότητα της ομαδικής εργασίας αφού κάθε μέλος έχει τα απαραίτητα δικαιώματα για να επεξεργάζεται ή να δημοσιεύει περιεχόμενο στον ιστότοπο.
- Το Drupal μπορεί να εγκατασταθεί σε Windows, Linux, , MacOSX, Solaris κ.α.

13.1.2 Μειονεκτήματα Drupal

- Όχι τόσο φιλικό ως προς τον χρήστη και τον designer. Χρειάζονται προγραμματιστικές γνώσεις για να εκμεταλλευτείς τις εξαιρετικές του δυνατότητες.
- Τα έτοιμα πρότυπα σχεδίασης του Drupal υστερούν από αισθητικής άποψης
- Η δημιουργία ενός Drupal website απαιτεί περισσότερο χρόνο

13.1.3 Τα κύρια χαρακτηριστικά του Drupal:

- **Συνεργατικό βιβλίο:** Αυτό το στοιχείο επιτρέπει στους χρήστες να δημιουργήσουν ένα "βιβλίο" και στη συνέχεια να εξουσιοδοτήσουν άλλα άτομα τα οποία θα συνεισφέρουν στην προσθήκη περιεχομένου
- **Φιλικά URLs:** Το Drupal εκμεταλλεύεται τη λειτουργία mod_rewrite του Apache για τη δημιουργία προσαρμοσμένων URLs τα οποία είναι φιλικά τόσο για το χρήστη όσο και για τις μηχανές αναζήτησης
- **Μονάδες:** Η κοινότητα του Drupal έχει δημιουργήσει πολλές μονάδες (modules), οι οποίες παρέχουν επιπρόσθετη λειτουργικότητα στο σύστημα
- **Online βοήθεια:** Όπως σε πολλά άλλα έργα ανοιχτού λογισμικού, έτσι και στο Drupal δεν μπορεί να υποστηρίξει κανείς ότι η online βοήθεια είναι τέλεια. Ωστόσο υπάρχει ενσωματωμένο στην πλατφόρμα ένα ιδιαίτερα εύχρηστο online σύστημα βοήθειας.
- **Ανοιχτός κώδικας:** Ο πηγαίος κώδικας του Drupal είναι ελεύθερα διαθέσιμος υπό τους όρους της άδειας χρήσης GNU GPL 2. Σε αντίθεση με ιδιόκτητα συστήματα διαχείρισης περιεχομένου (Content Management Systems, CMS) ή blogs, το σύνολο των χαρακτηριστικών του Drupal μπορεί εύκολα να προσαρμοστεί ή να επεκταθεί σύμφωνα με τις ανάγκες των χρηστών.
- **Προσωποποιημένο περιβάλλον:** Βασικό στοιχείο του συστήματος είναι ένα ιδιαίτερα χρήσιμο προσωποποιημένο περιβάλλον. Τόσο το περιεχόμενο όσο και η παρουσίασή του μπορεί να είναι διαφορετικά ανάλογα με τις επιλεγμένες ρυθμίσεις κάθε χρήστη.
- **Σύστημα δικαιωμάτων με βάση το ρόλο:** Οι διαχειριστές του Drupal δεν είναι υποχρεωμένοι να ασχοληθούν με την παροχή ξεχωριστών δικαιωμάτων σε κάθε χρήστη. Αντιθέτως, αναθέτουν δικαιώματα σε ρόλους και στη συνέχεια αναθέτουν τους ρόλους αυτούς σε ομάδες χρηστών.

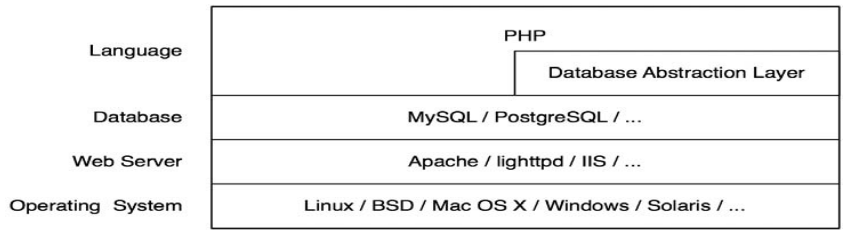
- **Αναζήτηση:** Χρησιμοποιώντας τη μονάδα αναζήτησης του Drupal μπορεί κανείς να διενεργήσει αναζήτηση σε όλο το σύστημα
- **Πιστοποίηση χρήστη:** Οι χρήστες μπορούν να εγγραφούν και να πιστοποιηθούν είτε τοπικά είτε χρησιμοποιώντας κάποια εξωτερική πηγή πιστοποίησης, όπως το Jabber, το Blogger ή το LiveJournal. Αν πρόκειται να χρησιμοποιηθεί εντός ενός ενδοδικτύου, το Drupal μπορεί να συνεργαστεί με εξυπηρετητή LDAP.
- **Δημοσκοπήσεις:** Το σύστημα διαθέτει μονάδα δημοσκοπήσεων η οποία δίνει τη δυνατότητα στους χρήστες και τους διαχειριστές να διενεργούν δημοσκοπήσεις και να τις παρουσιάζουν σε διάφορες σελίδες

- **Δημιουργία προτύπων:** Το σύστημα διαχωρίζει το περιεχόμενο από την παρουσίασή του, επιτρέποντας στο χρήστη να ελέγχει την εξωτερική εμφάνιση του διαδικτυακού χώρου. Τα πρότυπα δημιουργούνται με τη βοήθεια κώδικα HTML και PHP, κάτι που σημαίνει ότι ο χρήστης δε χρειάζεται να μάθει κάποια ιδιόκτητη γλώσσα δημιουργίας προτύπων
- **Έλεγχος εκδόσεων:** Το σύστημα εκδόσεων του Drupal καταγράφει τις λεπτομέρειες των ενημερώσεων του περιεχομένου, λεπτομέρειες όπως το χρήστη που τροποποίησε το περιεχόμενο, την ημερομηνία και την ώρα που οι τροποποιήσεις αυτές πραγματοποιήθηκαν, καθώς και ποια τμήματα του περιεχομένου έχουν τροποποιηθεί. Το σύστημα ελέγχου εκδόσεων παρέχει τη δυνατότητα στο χρήστη να προσθέτει σχόλια, καθώς και να μπορεί να επαναφέρει το περιεχόμενο σε προηγούμενη έκδοσή του
- **Υποστήριξη API για Bloggers:** Η API για Bloggers επιτρέπει στο διαδικτυακό χώρο να ενημερώνεται μέσω πολλών διαφορετικών εργαλείων. Σε αυτά περιλαμβάνονται εργαλεία που δεν είναι web-based, τα οποία παρέχουν πλουσιότερο περιβάλλον επεξεργασίας.

- **Διάθεση περιεχομένου:** Το Drupal έχει τη δυνατότητα να εξάγει το περιεχόμενό του σε μορφή RDF/RSS. Αυτό επιτρέπει σε όποιον διαθέτει ένα συλλέκτη ειδήσεων να έχει πρόσβαση στα RSS νέα ενός διαδικτυακού χώρου Drupal.
- **Συλλέκτης ειδήσεων:** Το Drupal έχει ενσωματωμένο έναν εύχρηστο συλλέκτη ειδήσεων για την ανάγνωση νέων και blogs από άλλους διαδικτυακούς χώρους. Ο συλλέκτης ειδήσεων αποθηκεύει προσωρινά τα άρθρα σε βάση δεδομένων MySQL.
- **Δυνατότητα πολλαπλών γλωσσών:** Το Drupal έχει σχεδιαστεί με τέτοιο τρόπο ώστε να ανταποκρίνεται στις απαιτήσεις των χρηστών του διεθνώς και για το λόγο αυτό παρέχει πλήρες περιβάλλον εργασίας για τη δημιουργία πολυγλωσσικών διαδικτυακών χώρων, blogs και εφαρμογών διαχείρισης περιεχομένου. Όλο το κείμενο μπορεί να μεταφραστεί με τη βοήθεια γραφικού περιβάλλοντος, εισάγοντας υφιστάμενες μεταφράσεις ή ενσωματώνοντας κάποιο άλλο εργαλείο, όπως το GNU gettext.
- **Ανάλυση, καταγραφή και στατιστικά:** Το Drupal μπορεί να εκτυπώσει αναφορές με πληροφορίες που αφορούν τη δημοφιλία του περιεχομένου, καθώς και τον τρόπο με τον οποίο οι επισκέπτες περιηγούνται στο διαδικτυακό χώρο.
- **Web-based διαχείριση:** Η διαχείριση του Drupal πραγματοποιείται εξ ολοκλήρου χρησιμοποιώντας κάποιον φυλλομετρητή και ως εκ τούτου μπορεί να γίνει από οποιοδήποτε σημείο της γης και δεν απαιτείται η εγκατάσταση επιπρόσθετου λογισμικού.
- **Forums συζήτησης:** Στο Drupal υπάρχει πλήρης δυνατότητα ενσωμάτωσης forum συζητήσεων για τη δημιουργία ζωντανών, δυναμικών διαδικτυακών χώρων.

13.1.4 Τεχνολογική υποδομή του Drupal

Οι σχεδιαστικοί στόχοι του Drupal συνδυάζουν την ικανότητα λειτουργίες σε απλές διαδικτυακές εφαρμογές και σε διαδικτυακές πύλες μαζικής διανομής περιεχομένου. Ο πρώτος στόχος ικανοποιείται με την χρήση των πιο διαδεδομένων τεχνολογιών και ο δεύτερος με προσεκτικό και καλογραμμένο κώδικα. Η τεχνολογική υποδομή του Drupal παρουσιάζεται στην εικόνα:



Εικόνα 19: Η τεχνολογική υποδομή του Drupal

Το Drupal μπορεί να "τρέξει" σε οποιοδήποτε λειτουργικό σύστημα και στους περισσότερους διακομιστές (Web Servers) που υποστηρίζουν την PHP, με τον πιο ευρέως χρησιμοποιούμενο να είναι ο Apache.

Ο κώδικας PHP του Drupal επικοινωνεί με το επίπεδο (layer) της βάσης δεδομένων διαμέσω ενός επιπέδου αφαίρεσης βάσης δεδομένων (database abstraction layer), το όποιο είναι υπεύθυνο για αποτροπή επιθέσεων με τεχνική SQL injection, κάνοντας φιλτράρισμα (sanitation) στα δεδομένα που εισάγει ένας χρήστης μέσω φορμών, και επιπλέον προσφέρει ανεξαρτησία του κώδικα από το σύστημα βάσεων δεδομένων, δίνοντας μας τη δυνατότητα να χρησιμοποιούμε ό,τι σύστημα θέλουμε (MySQL, PostgreSQL, Microsoft SQL Server Oracle) χωρίς να χρειάζεται να κάνουμε αλλαγές στον κώδικα.

Το sanitation των SQL ερωτημάτων γίνεται τοποθετώντας placeholders στα queries και αντικαθιστώντας τα placeholders με παραμέτρους.

Π.χ. *$result = db_query('SELECT name FROM {role} WHERE rid = %d', -);* Η ανεξαρτησία από το σύστημα της βάσης δεδομένων επιτυγχάνεται με το Database Api του Drupal το οποίο παρέχει γενικές εντολές για χειρισμό της βάσης, ανεξάρτητες από το σύστημα και έχοντας αναγνωρίσει τι σύστημα και ποια βάση χρησιμοποιεί κατά την αρχική διαδικασία της εγκαθίδρυσης (establishing) της σύνδεσης με τη βάση.

Για παράδειγμα δε χρησιμοποιεί τις εντολές της PHP, *mysql_query() (*για MySQL βάση) ή *pg_query()* (για Postgress), αλλά μια γενική *db_query().*

Ο κώδικας του πυρήνα του Drupal συμμορφώνεται με τα αυστηρά πρότυπα γραφής κώδικα.

14 Υλοποίηση

14.1 Μια μικρή εισαγωγή για το τη και που θα τα μπουν τα αρχεια του Drupal και πιο κάτω θα τα δούμε ποιο αναλυτικά .

Όπως είπαμε, το Drupal είναι ένα σύστημα γραμμένο σε PHP και συνεργάζεται με την MySQL. Άρα ο Web Server που φιλοξενεί το Drupal site πρέπει να είναι εφοδιασμένος με την PHP και επίσης να υπάρχει βάση δεδομένων έτοιμη να δεχθεί τα δεδομένα. Για τη σωστή λειτουργία της έκδοσης 6 του Drupal απαιτούνται ο **Apache 1.3 ή 2.x** ή εναλλακτικά ο Web Server **IIS 5, IIS 6 ή IIS 7** της

Microsoft, η **MySQL 4.1 ή 5** και η **PHP4.4.0** ή μεγαλύτερη (προτεινόμενη η 5.2.x.,καθώς η PHP 5.3 δε συνεργάζεται ακόμα τέλεια με όλα τα modules).

Η έκδοση του Drupal στην οποία υλοποιήθηκε το site είναι η 6.x, που είναι και η τελευταία, καθώς η έκδοση 7 βρίσκεται σε δοκιμαστικό στάδιο. Ως Web Server επιλέχθηκε ο Apache, καθώς η πλειοψηφία των Drupal site έχουν υλοποιηθεί σε αυτόν, με αποτέλεσμα να υπάρχει περισσότερη εμπειρία πάνω στον συγκεκριμένο Web Server στην κοινότητα του Drupal. Επίσης είναι ένας από τους δημοφιλέστερους Web Server, γιατί λειτουργεί σε διάφορες πλατφόρμες όπως Windows, Linux, Unix και Mac OS X και παράγεται και διανέμεται δωρεάν από μια κοινότητα ανοιχτού κώδικα με επιτήρηση από το Ίδρυμα Λογισμικού Apache (Apache Software Foundation).

Το **XAMPP** είναι ένα ελεύθερο και ανοικτό crossplatform Web Server package, που αποτελείται κυρίως από τον Apache HTTP Server, τη MySQL βάση δεδομένων και των διερμηνέων για scripts γραμμένα σε γλώσσες προγραμματισμού PHP και Perl. Η έκδοση Xampp 1.7.1 που χρησιμοποιήσαμε διαθέτει τον Apache HTTPD 2.2.11 ,τη MySQL 5.1.33,την PHP 5.2.9 και το εργαλείο PHPMyAdmin, το οποίο είναι ένα λογισμικό γραμμένο σε PHP που προορίζεται για τη διαχείριση της MySQL μέσω του Παγκόσμιου Ιστού. Εγκαθιστώντας το πακέτο αυτό έχουμε τα απαραίτητα εργαλεία στο μηχάνημά μας για να υποδεχθεί το Drupal. Στη συνέχεια κατεβάζουμε τα αρχεία του Drupal και τα τοποθετούμε στο φάκελο C:\xampp\htdocs. Έτσι, αν τώρα πληκτρολογήσουμε στον browser http://localhost όταν ο web Server βρίσκεται στον υπολογιστή μας ή το domain όνομα του site αν φιλοξενείται σε κάποιον απομακρυσμένο Web Server, ανοίγει το αρχείο install.php και ξεκινάει η διαδικασία της εγκατάστασης. Συνδέουμε τη βάση την οποία έχουμε προηγουμένως δημιουργήσει μέσω του PHPMyAdmin, δημιουργούμε το λογαριασμό για το διαχειριστή του site και ρυθμίζουμε ιδιότητες του site, όπως το όνομα, το email και την ημερομηνία. Κατά τη διάρκεια της εγκατάστασης μας ζητάει να μετονομάσουμε το αρχείο default.settings.php, που βρίσκεται στο C:\xampp\htdocs\sites\default, σε settings.php και να αποθηκεύσει σε αυτό κάποιες ιδιότητες σχετικά με τη σύνδεση της βάσης δεδομένων
($db_url = 'mysql://username:password@localhost/databasename'). Όταν η εγκατάσταση τελειώσει, πληκτρολογώντας http://localhost ή το domain name του site το Drupal βλέπει το αρχείο settings.php και με βάση αυτό μας κατευθύνει στην αρχική σελίδα του site. Το αρχείο settings.php είναι πολύ σημαντικό καθώς είναι υπεύθυνο για τη σύνδεση της βάσης δεδομένων με το Drupal . Μπορούμε να αλλάξουμε το αρχείο αυτό, όπως για παράδειγμα τη ρύθμιση της βάσης δεδομένων της Drupal εγκατάστασης ή να ξανακάνουμε την εγκατάσταση σβήνοντάς το και αφήνοντας μόνο το αρχικό default.settings.php. Τα modules και τα template themes που κατεβάζουμε πρέπει να τοποθετηθούν κάτω από το φάκελο \htdocs\modules και \htdocs\themes είτε κάτω από το φάκελο sites/all/modules και sites/all/themes.

14.1.1 Πολλαπλά site με μία εγκατάσταση Drupal

Ένα μεγάλο πλεονέκτημα του Drupal, είναι η δυνατότητα για πολλαπλά site με μία εγκατάσταση Drupal. Για να γίνει αυτό, δημιουργούμε φακέλους (πχ site1 και site2) κάτω από το φάκελο \htdocs\sites και αντιγράφουμε μέσα στους υποφακέλους το default.settings.php για να γίνει ξεχωριστή εγκατάσταση για το κάθε site. Αν θέλουμε να έχουμε διαφορετικά themes, modules ή αρχεία για το κάθε site τότε τοποθετούμε τα αρχεία αυτά μέσα στον φάκελο του κάθε site και όχι στο sites/all που θα είναι για όλα τα sites. Οι επιπλέον ρυθμίσεις που πρέπει να γίνουν είναι στο αρχείο hosts και πρέπει να δηλώσουμε ότι στη localhost IP address και τα url site1 και site2.

Τώρα πληκτρολογώντας στο browser είτε localhost, είτε site1, είτε site2 θα βλέπει το αρχείο settings.php. Άρα θα πρέπει να πούμε στο Server να κοιτάει στους υποφακέλους site1 και site2 και να τους αντιμετωπίζει σαν ξεχωριστά sites .Αυτό το ρυθμίζουμε στο αρχείο httpdvhosts που βρίσκεται στο φάκελο \xampp\apache\conf\extra και δηλώνουμε πολλούς διαφορετικούς εικονικούς διακομιστές.

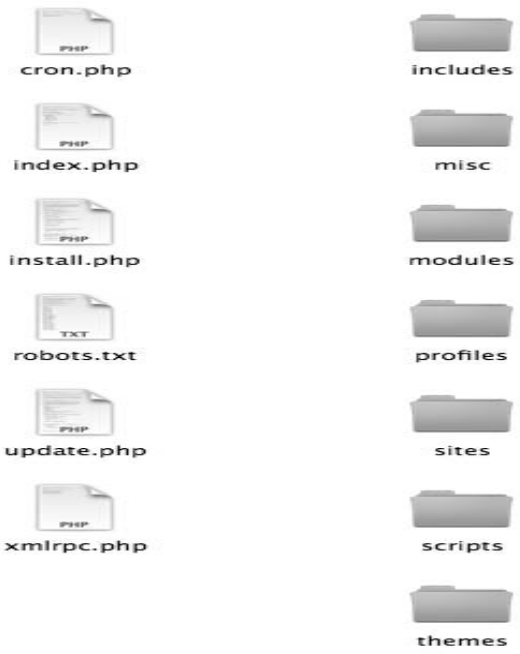
*<VirtualHost *.80>*
DocumentRoot C:/xampp/htdocs
ServerName localhost
</VirtualHost>
<VirtualHost test1>
DocumentRoot C:/xampp/htdocs
ServerName test1
</VirtualHost>
<VirtualHost test2>
DocumentRoot C:/xampp/htdocs
ServerName test2
</VirtualHost>

Στη συνέχεια αρκεί να κάνουμε επανεκκίνηση στον Server. Έτσι τώρα πληκτρολογώντας στο url το http://site.

14.2 Εγκατάσταση του απαραίτητου λογισμικού

Όπως αναφέραμε και ποιο πάνω προκειμένου να είναι δυνατή η εγκατάσταση του Drupal και κατά επέκταση η υλοποίηση αυτής της ιστοσελίδας θα χρειαστεί πρώτα εγκαταστήσουμε κάποιο λογισμικό. Πρώτη και βασική προϋπόθεση είναι η ύπαρξη ενός web server. Μετά να έχουμε εγκαταστήσει τα απαραίτητα εργαλεία όπως **WampServer** και τις ρυθμίσεις που χρίζονται, Και με τη σειρά τους της Αφού εγκατάστασης του **Apache** ,την **PHP** και την **MySQL** με την χρήση του πακέτου **XAMPP**, εφόσον έχουμε κάνει όλες αυτές της παραπάνω εγκαταστήσεις ήρθε η ώρα για την εγκατάσταση του Drupal . Πάμε λοιπόν να δούμε πώς να φτιάξουμε το site μας με drupal γρήγορα και εύκολα. Αυτό που όμως μας ενδιαφέρει είναι να δούμε πως λειτουργεί και ποια είναι τα βήματα που υλοποιείται το drupal. Αφού έχουμε έτοιμο τον server μας πηγαίνουμε στην σελίδα του www.drupal.org και κατεβάζουμε το σύνολο από το file του Drupal στον Υπολογιστή μας την πιο πρόσφατη έκδοση. Στην περίπτωση μας την 6.

Μετά την ολοκλήρωση της διαδικασίας το αποσυμπιέσαμε μέσα στο root του server μας . Έτσι είχαμε αυτά τα αρχεία:



Εικόνα 20: Τα αρχεία του Drupal .

Παρακάτω δίνονται κάποιες πληροφορίες σχετικά με τους πιο σημαντικούς φακέλους και αρχεία:

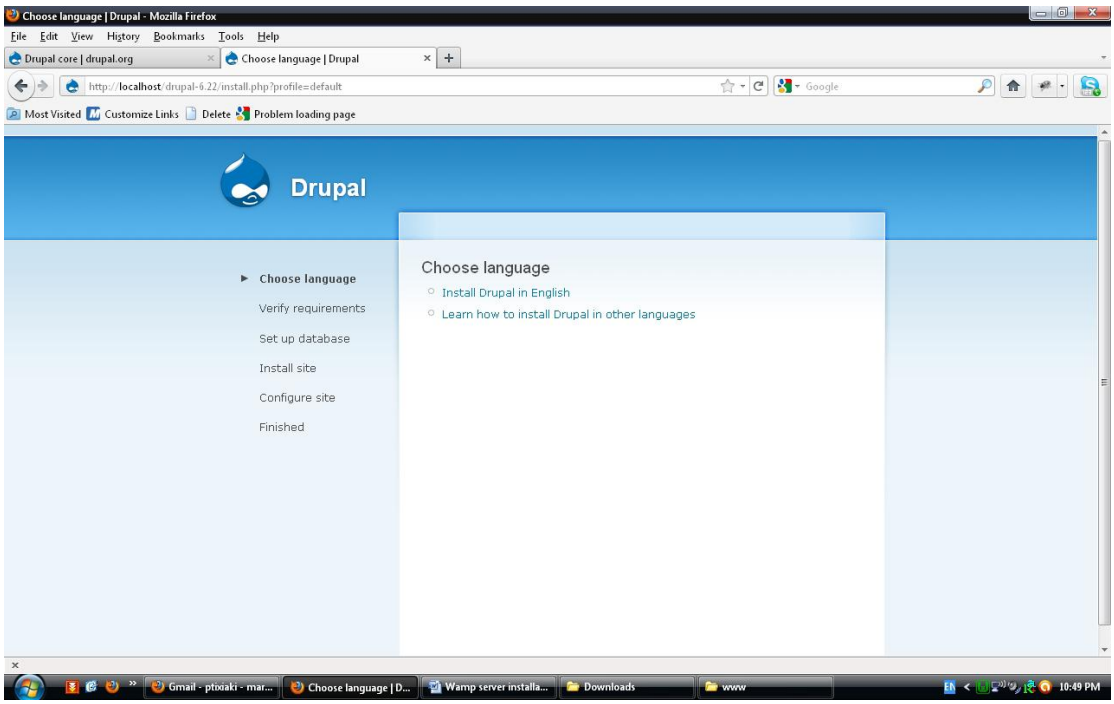
ΦΑΚΕΛΟΙ	ΠΕΡΙΓΡΑΦΗ
Includes:	Περιέχει βιβλιοθήκες συναρτήσεων του Drupal.
Modules:	Περιέχει τα modules του πυρήνα, ενώ τα επιπλέον modules που θα προστεθούν πρέπει να μπουν κάτω από το φάκελο sites/all/modules.
Themes:	Themes περιέχει τα default themes του Drupal και τα template engines, ενώ τα επιπλέον themes που θα προστεθούν πρέπει να μπουν κάτω από το φάκελο sites/all/themes.
Sites:	Περιέχει τις αλλαγές που κάνουμε στο Drupal όσον αφορά ρυθμίσεις (settings), modules και themes. Στον υποφάκελο sites/default βρίσκεται το προκαθορισμένο αρχείο ρυθμίσεων (default configuration file) για το Drupal site, το default.settings.php, το οποίο ο installer του Drupal θα αλλάξει σύμφωνα με τις πληροφορίες που θα δώσουμε κατά την εγκατάσταση και θα τις καταγράψει στο

	setting.php.

ΑΡΧΕΙΟ	ΠΕΡΙΓΡΑΦΗ
cron.php:	Το cron.php χρησιμοποιείται για την εκτέλεση περιοδικών εργασιών, της ξεκαθάρισμα πινάκων της βάσης δεδομένων και συλλογή στατιστικών.
Index.php:	Το index.php είναι το κύριο σημείο όλων των αιτήσεων εξυπηρέτησης.
install.php:	Το install.php είναι το κύριο αρχείο για την εγκατάσταση του Drupal.
Update.php:	Το update.php ανανεώνει το σχήμα της βάσης μετά από μια αναβάθμιση της έκδοσης του Drupal.

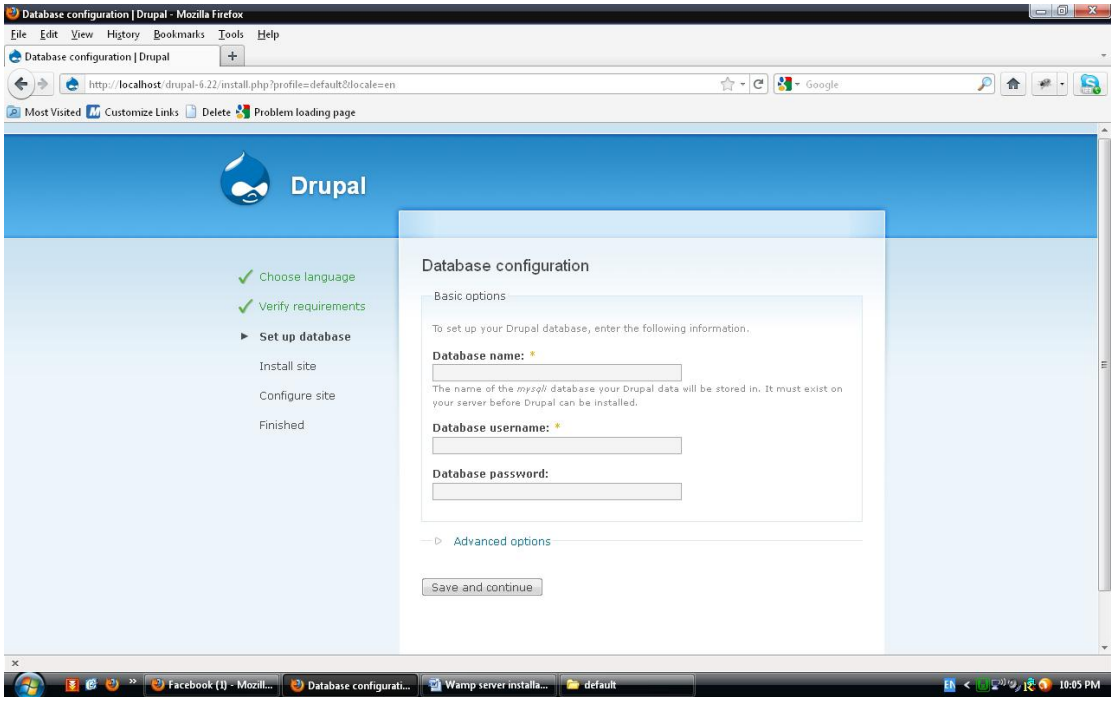
Έπειτα φτιάξαμε την βάση δεδομένων μας. Πήγαμε στον browser και γράψαμε http://localhost/phpmyadmin (με wamp είναι 100% αυτό). Μας ζήτησε κωδικό,δώσαμε τα στοιχεία μας και πατήσαμε εκτέλεση.

Γυρίσαμε πάλι στο localhost /drupal που έχουμε τα αρχεία μας. Είδαμε αυτό:



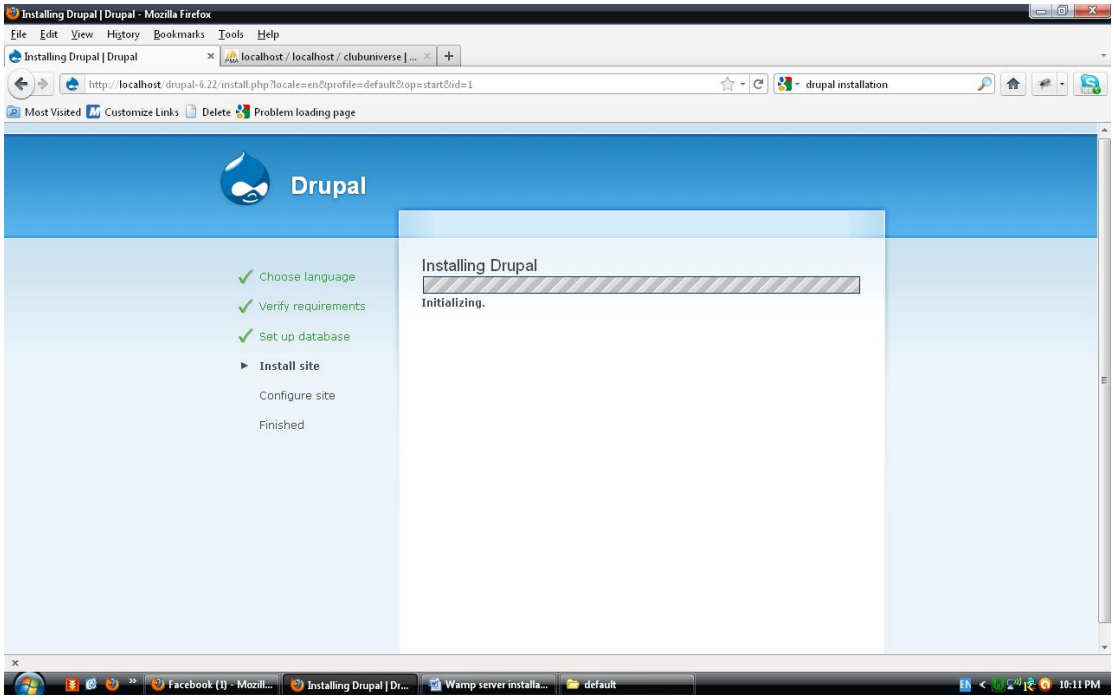
Εικόνα 21: Το localhost/drupal του Drupal .

Διαλέγουμε το install Drupal in English -> πατάμε **try again:** Για να συνεχίσει η εγκατάσταση πήγαμε λοιπόν στον φάκελο με τα αρχεία του drupal και μπήκαμε στον φάκελο sites- default. Εκεί είδαμε ένα αρχείο με όνομα default.settings.php. Αυτό που κάναμε ήταν copy paste στον ίδιο φάκελο, μας έβγαλε ένα αρχείο με όνομα Αντίγραφο από default.settings κάναμε μετονομασία και το κάναμε settings.
Μετά από αυτή την διαδικασία πήγαμε πάλι στον browser και πατήσαμε try again. Και βλέπουμε πως η εγκατάσταση τώρα προχωρούσε.



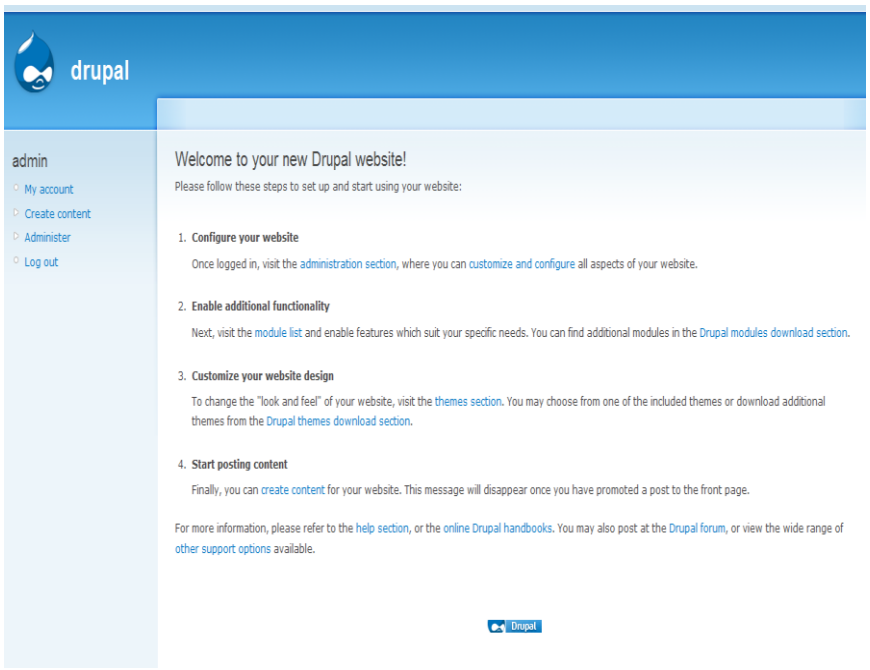
Εικόνα 22: Το database του Drupal .

Δώσαμε το όνομα της βάσης (drupal) το username του phpmyadmin και τον κωδικό.
Μετά πατήσαμε Save and continue.



Εικόνα 23: Γίνετε εγκατάσταση του Drupal.

Μετά από αυτό το στάδιο πρέπει να ορίσουμε κάποιες ρυθμίσεις της ιστοσελίδας μας. Στην συνέχεια εισάγουμε κάποια στοιχεία όσον αφορά τον λογαριασμό τον διαχειριστή της ιστοσελίδας το username, την διεύθυνση ηλεκτρονικού ταχυδρομείου και τον κωδικό πρόσβασης



Εικόνα 24: Η διαχειριστή της ιστοσελίδας του Drupal.

14.3 Ολοκλήρωση της εγκατάστασης του Drupal

Έχουμε πλέον ολοκληρώσει την εγκατάσταση του Drupal. Στο παραπάνω στιγμιότυπο οθόνης μπορούμε να δούμε την αρχική σελίδα του Drupal. Εκ των αριστερών βρίσκεται το βασικό μενού πλοήγησης μέσω του οποίου μπορούμε να πλοηγηθούμε στις διάφορες περιοχές της διεπαφής διαχείρισης.

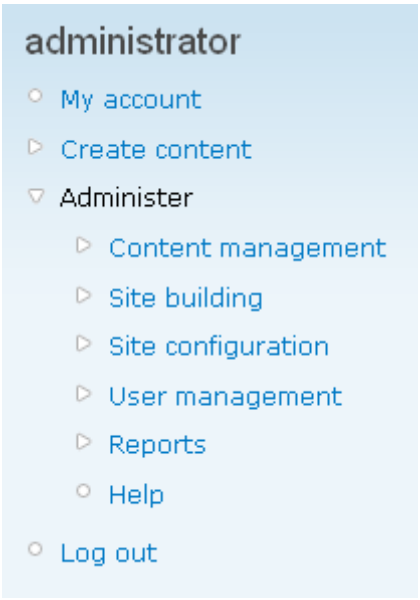
Γνωριμία με το περιβάλλον εργασίας

Εφόσον έχουμε κατεβάσει θα δούμε πρώτα το περιβάλλον εργασίας και τα μενού διαχείρισης του Drupal.

Το βασικό μενού διαχείρισης του Drupal αποτελείται από τέσσερα αντικείμενα:

- **My Account :** Όταν είμαστε μέσα στο My Account μας πηγαίνει στον λογαριασμό του χρήστη που είναι συνδεδεμένος στο περιβάλλον διαχείρισης και στην προκειμένη περίπτωση, στον λογαριασμό του διαχειριστή.
- **Create content :** Το αντικείμενο Create content αφορά την δημιουργία νέου περιεχομένου και επιλέγοντας το μας εμφανίζει μία λίστα με τους διάφορους τύπους περιεχομένου από τους οποίους μπορούμε να επιλέξουμε για την δημιουργία του αντίστοιχου περιεχομένου.
- **Administer και :** Το αντικείμενο Administer αποτελείται από άλλα υπομενού και από εδώ γίνονται όλες οι ρυθμίσεις τις ιστοσελίδας.
- **Log out :** Το Log out χρησιμοποιείται για την έξοδο μας από το περιβάλλον διαχείρισης.

Τα επόμενου της διαχείρισης του Drupal είναι χωρισμένη (από προεπιλογή) σε 5 κύριες κατηγορίες και έναν τομέα βοήθειας. Αυτές οι κατηγορίες είναι:



Εικόνα 25:Administration menu

- **Content management**: Από εδώ γίνεται η διαχείριση του περιεχομένου της ιστοσελίδας.
- **Site building**: Έλεγχος της όψης και δομής της ιστοσελίδας, καθώς και διαχείριση των μενού και των modules.
- **User management**: Διαχείριση των χρηστών, των ομάδων και των δικαιωμάτων πρόσβασης.
- **Site configuration**: Έλεγχος των βασικών ρυθμίσεων της ιστοσελίδας.
- **Reports**: Αναφορές που δημιουργούνται από διάφορες καταγραφές συμβάντων του συστήματος.
- **Help:** το οποίο είναι η βοήθεια

14.3.1 Content management

Τα εργαλεία του τομέα Content management μας επιτρέπουν να διαχειριστούμε το περιεχόμενο της ιστοσελίδας και συνεπώς είναι ένας από τους πιο σημαντικούς τομείς διαχείρισης του Drupal. Τα εργαλεία αυτά μας παρέχουν την δυνατότητα ρυθμίσεων που αφορούν τα:

- **Comments (Σχόλια) :** Οι χρήστες τις ιστοσελίδας έχουν την δυνατότητα να καταχωρούν σχόλια που αφορούν το περιεχόμενο που προβάλλεται. Το μενού comments είναι το μέρος που ο διαχειριστής μπορεί να έχει μια γενική εικόνα για τα σχόλια που υποβάλλονται στην ιστοσελίδα και να προβεί σε διαφορές διαθέσιμες ενέργειες. Τα σχόλια μπορούν να έχουν δύο καταστάσεις. Είτε published, που σημαίνει ότι είναι ορατά στην ιστοσελίδα, είτε unpublished που σημαίνει ότι μπορούν να τα δουν μόνο οι διαχειριστές της ιστοσελίδας.

- **Content (Περιεχόμενο)** : Μπορούμε να δούμε, να τροποποιήσουμε και να διαγράψουμε όποιο περιεχόμενο έχουμε δημιουργήσει εμείς ως διαχειριστής ή κάποιος άλλος χρήστης.

- **Content types** (Τύπους περιεχομένων) :
- **Post settings** (Ρυθμίσεις καταχωρήσεων)
- **RSS publishing:** Αφορά κάποιες ρυθμίσεις σχετικά με την τροφοδοσία δεδομένων από εξωτερικές πηγές. Αφορά κάποιες ρυθμίσεις σχετικά με την τροφοδοσία δεδομένων από εξωτερικές πηγές.
- **Taxonomy** (Ταξινόμηση) : Από εδώ μπορούμε να διαχειριστούμε την ταξινόμηση, την κατηγοριοποίηση και τις ετικέτες του περιεχομένου μας.

Από το μενού **Site building** μπορούμε μπορούμε να ελέγξουμε την διαρρύθμιση και την εμφάνιση της ιστοσελίδας μας.

- **Blocks:** Σε γενικές γραμμές οι ιστοσελίδες χωρίζονται σε διάφορες περιοχές. Από εδώ μπορούμε να ρυθμίσουμε σε ποιες περιοχές θα εμφανίζονται τα διάφορα blocks περιεχομένου
- **Menus:** Μπορούμε να ελέγξουμε τα μενού της ιστοσελίδας όπως το μενού πλοήγησης, πρωτευόντων και δευτερευόντων συνδέσμων, καθώς και άλλων μενού που μπορούμε να δημιουργήσουμε εμείς οι ίδιοι. Επίσης μπορούμε να προσθέσουμε νέα μενού ή να επεξεργαστούμε τα ήδη υπάρχοντα.
- **Modules:** Αφορά την περιοχή από την οποία μπορούμε να ενεργοποιήσουμε ή να απενεργοποιήσουμε τα διάφορα επιπρόσθετα modules (ενότητες).
- **Themes:** Από εδώ μπορούμε να ορίσουμε είτε εμείς είτε οι χρήστες ένα theme (θέμα) της προτίμησης μας.

Σε αυτήν την κατηγορία **Site configuration** προσαρμόζουμε τις βασικές ρυθμίσεις και επιλογές για την ιστοσελίδα μας.

- **Actions:** Τα actions (ενέργειες) είναι μεμονωμένες εργασίες που μπορεί να κάνει το σύστημα όπως ή κατάργηση δημοσίευσης ενός περιεχομένου ή η απαγόρευση ενός χρήστη. Υπάρχουν κάποια modules όπως το trigger module που ενεργοποιούν αυτές τις ενέργειες όταν ορισμένα γεγονότα του συστήματος συμβούν. Υπάρχουν δύο είδη ενεργειών, οι απλές (simple) και οι προχωρημένες (advanced).
- **Administration theme:** Από εδώ μπορούμε να ορίσουμε ποιο theme θα εμφανίζεται στις σελίδες διαχείρισης.
- **Clean URLs:** Μπορούμε να ενεργοποιήσουμε ή να απενεργοποιήσουμε αυτή την επιλογή.
- **Date and time:** Ρυθμίσεις για το πώς εμφανίζεται η ημερομηνία και η ώρα στο Drupal καθώς και για την ζώνη ώρας του συστήματος.
- **Error reporting:** Από εδώ ελέγξουμε πως θα εμφανίζονται η σελίδες λάθους 403/404 (Access denied/not found) καθώς και να ρυθμίσουμε πως θα εμφανίζονται η αναφορές λάθους.
- **File system:** Ορίζουμε που αποθηκεύονται τα αρχεία μας και τον τρόπο πρόσβασης σε αυτά.
- **Image toolkits:** Από εδώ επιλέγουμε ποιο εργαλείο επεξεργασίας εικόνων θα χρησιμοποιήσουμε. Έχουμε την δυνατότητα να προσθέσουμε κάποιο εργαλείο της αρεσκείας μας.
- **Input formats:** Αφορά τις ρυθμίσεις φιλτραρίσματος για το περιεχόμενο που δημιουργούν οι χρήστες συμπεριλαμβάνοντας τα html tags. Επίσης επιτρέπεται η ενεργοποίηση φίλτρων που παρέχονται από άλλα modules.
- **Logging and alerts:** Ρυθμίσεις για modules σχετικά με τις συνδέσεις και τις ειδοποιήσεις.
- **Performance:** Ενεργοποίηση ή απενεργοποίηση της επιλογής cache των σελίδων για τους ανώνυμους χρήστες και ρυθμίσεις για CSS και Javascript που αποσκοπεί στην βελτιστοποίηση χρήσης του εύρους ζώνης.
- **Site information:** Από εδώ μπορούμε να ρυθμίσουμε βασικές πληροφορίες της ιστοσελίδας μας όπως την ονομασία της, το σλόγκαν, την διεύθυνση ηλεκτρονικού ταχυδρομείου,τον σκοπό, την αρχική σελίδα και άλλα.
- **Site maintenance:** Θέτουμε την ιστοσελίδα μας offline για συντήρηση ή την επαναφέρουμε online.

Από το User management Από αυτή την κατηγορία μπορούμε να διαχειριστούμε τους χρήστες και τα γκρουπ της ιστοσελίδας μας. Επίσης έχουμε πρόσβαση σε επιλογές της ιστοσελίδας. Επίσης από της επιλογές αυτές άλλες πληροφορίες κατάστασης.

- **Access rules:** Έχουμε την δυνατότητα να ορίσουμε κανόνες για να μην επιτρέψουμε κάποια usernames, διευθύνσεις ηλεκτρονικού ταχυδρομείου και διευθύνσεις IP.
- **Permissions:** Καθορισμός πρόσβασης σε χαρακτηριστικά της ιστοσελίδας επιλέγοντας δικαιώματα για τους ρόλους χρηστών.
- **Roles:** Από εδώ μπορούμε να δούμε, να επεξεργαστούμε και να προσθέσουμε ρόλους χρηστών.
- **User settings:** Αφορά τις ρυθμίσεις της προεπιλεγμένης «συμπεριφοράς» των χρηστών συμπεριλαμβανομένου των απαιτήσεων εγγραφής, διευθύνσεων ηλεκτρονικού ταχυδρομείου και φωτογραφιών των χρηστών.
- **Users:** από εδώ μπορούμε να δούμε την λίστα με τους χρήστες μας και να προσθέσουμε κάποιους ή να επεξεργαστούμε τους ήδη υπάρχοντες.

Υπάρχει και μια υποκατηγορία που λέγεται **Reports** και από αυτές τις επιλογές μπορούμε να δούμε αναφορές από τα αρχεία καταγραφής του συστήματος και άλλες πληροφορίες κατάστασης:

- **Recent log entries:** Προβολή γεγονότων που πρόσφατα καταγράφηκαν.
- **Top acces denied errors (403s):** Από εδώ μπορούμε να δούμε

τις σχετικές εγγραφές από τα αρχεία καταγραφής του συστήματος.

- **Top page not found errors (404s):** Από εδώ μπορούμε να δούμε τις σχετικές εγγραφές από τα αρχεία καταγραφής του συστήματος.
- **Available updates**: Μας ενημερώνει το σύστημα με μία αναφορά κατάστασης, για τις διαθέσιμες ενημερώσεις των εγκατεστημένων modules, themes καθώς και για τον πυρήνα του Drupal.
- **Status report:** Το σύστημα παράγει μία αναφορά κατάστασης, την οποία μπορούμε να δούμε από εδώ, σχετικά με τις λειτουργίες της ιστοσελίδας και διεγνωσμένων προβλημάτων.

14.4 Ασφάλεια

Στο διαδίκτυο, η ασφάλεια είναι μια ταχέως εξελισσόμενη και πάντα παρούσα πρόκληση. Δεν υπάρχει ένας συγκεκριμένος τρόπος για να εξασφαλίσουμε την ακεραιότητα ενός ιστότοπου, καθώς οι κίνδυνοι είναι πολλοί και οι μέθοδοι ασφαλείας αναθεωρούνται και απαξιώνονται ανά πάσα στιγμή. Κάποιοι από τους κινδύνους που απειλούν ένα site είναι οι επιθέσεις DoS (DenialofService), οι SQL injections, το CrossSite Scripting (XSS) το Phising και το Packet Sniffer. Πιο συγκεκριμένα, όταν μιλάμε για DoS επιθέσεις εννοούμε τεχνικές, με συνηθέστερη την αποστολή συνεχόμενων αιτήσεων επικοινωνίας, με στόχο την κατανάλωση των πόρων του συστήματος θύματος, έτσι ώστε να μην μπορεί να ανταποκριθεί στις απαιτήσεις της νόμιμης κυκλοφορίας ή να ανταποκρίνεται τόσο αργά ώστε να καταστεί αποδοτικά μη διαθέσιμο. Οι SQL Injections από τη μεριά τους είναι επιθέσεις με στόχο την πρόσβαση στην βάση δεδομένων και τη συγκέντρωση προσωπικών πληροφοριών, όπως ονόματα χρηστών και κωδικούς πρόσβασης ή απλά την καταστροφή κάποιων πινάκων της βάσης. Η διαδικασία μίας Injection λειτουργεί με την εισαγωγή κακόβουλων κομματιών κώδικα μέσα σε strings τα οποία θα περάσουν στον Server της SQL για μεταγλώττιση και εκτέλεση με αποτέλεσμα να ολοκληρωθεί πρόωρα μια σειρά κειμένων strings και να επισυναφθεί μια νέα εντολή sql. Ευτυχώς, υπάρχουν πολλές καλά καθιερωμένες αρχές που μπορούν να βοηθήσουν τον ιστότοπο που υλοποιήσαμε να παραμένει ασφαλής, όπως ο περιορισμός των δικαιωμάτων πάνω σε κρίσιμα αρχεία του συστήματος ή η αποτροπή εκτέλεσης PHP κώδικα, εισαγωγή επικίνδυνων html tags και upload επικίνδυνων αρχείων από μη εξουσιοδοτημένους χρήστες.

14.5 Οι Μονάδες

Οι μονάδες που χρησιμοποιήθηκαν καθώς και η περιγραφή της λειτουργικότητας τους εμφανίζονται παρακάτω:

14.5.1 Διαχείριση:

Administration menu:	Παρέχει ένα ανεξάρτητο θέμα διεπαφής της διοίκησης για το Drupal. Είναι ένας βοηθός για τους νέους χρήστες του Drupal που προέρχονται από άλλα CMS, ένα πραγματικό εργαλείο για την προφύλαξη των διαχειριστών της ιστοσελίδας του Drupal και σίγουρα αναγκαίο για τους προγραμματιστές και τους κατασκευαστές του Drupal site.
Administration menu toolbar style:	Μια καλύτερη εργαλειοθήκη.
Administration views:	Αντικαθιστά όλες τις σελίδες διαχείρισης του συστήματος αντικείμενο στο Drupal πυρήνων με πραγματικά views

14.5.2 Content Construction Kit (CCK):

Σε αυτή την κατηγορία έχει γίνει χρήση των προεπιλεγμένων από την εγκατάσταση, modules εκτός από μερικά που απενεργοποιήθηκαν. Πιο συγκεκριμένα τα modules που είναι ενεργοποιημένα αναφέρονται παρακάτω.

Content :	Επιτρέπει την προσωπική επικοινωνία καθώς και την επικοινωνία με χρήση γενικής φόρμας της ιστοσελίδας.
Content Copy :	Επιτρέπει την δυνατότητα εισαγωγής / εξαγωγής του προσδιορισμού των πεδίων περιεχομένου.
Content Permissions:	Ορίζει τα επίπεδα δικαιωμάτων για τα πεδια CCK.
Taxonomy:	Ενεργοποιεί την κατηγοριοποίηση του περιεχομένου
FileField :	Ορίζει τον τύπο αρχείου file.
FileField Meta:	Προσθέτει μεταδεδομένα συλλογής και αποθήκευσης για το FileField.
ImageField:	Ορίζει τον τύπο αρχείου image.
Location CCK:	Ορίζει ένα τύπο πεδίου με τίτλο τοποθεσία.
Node Reference:	Ορίζει ένα τύπο πεδίου για την αναφορά ενός

	κόμβου από έναν άλλο.
Node reference views:	Δείχνει πεδία nodereference χρησιμοποιώντας ένα view.
FileField ImageCache :	Υποστήριξη για τις εικόνες των πεδίων των αρχείων του CCK.
Number:	Καθορίζει τους τύπους αριθμητικών πεδίων.
Option Widgets:	Ορίζει την επιλογή, το check box και το radio button για κείμενο και αριθμητικά πεδία.
Text :	Ορίζει τον τύπο αρχείου απλού κειμένου.
User Reference:	Ορίζει ένα τύπο πεδίου για την αναφορά ενός χρήστη από ένα κόμβο.

14.5.3 Και σε αυτή την κατηγορία έχει γίνει χρήση των προεπιλεγμένων **modules** και σε καμία περίπτωση δεν πρέπει να απενεργοποιηθεί κάποιο από αυτά. Συγκεκριμένα αναφερόμαστε στα modules που ακολουθούν.

Block :	Ελέγχει τα πλαίσια πρόσθετης ύλης που εμφανίζονται γύρω από την κύρια ύλη.
Filter :	Χειρίζεται το φιλτράρισμα της ύλης κατά την προετοιμασία για παρουσίαση.
System :	Χειρίζεται τις γενικές ρυθμίσεις του ιστότοπου για τους διαχειριστές.
User :	Διαχειρίζεται το σύστημα εγγραφής και εισόδου χρηστών.
Block translation :	Επιτρέπει τα πολύγλωσσα μπλοκ και τη μετάφραση των μπλοκ.

14.5.4 Views:

Views:	Δημιουργεί προσαρμοσμένες λίστες και ερωτήματα από τη βάση δεδομένων μας.
Views Bulk Operations:	Εκθέτει νέο στυλ Προβολές «Μαζική εκμετάλλευσης» για την επιλογή πολλαπλούς κόμβους και την εφαρμογή δράσεων σχετικά με αυτές.
Views exporter:	Επιτρέπει την εξαγωγή πολλαπλές εμφανίσεις με τη μία. Views Test: Τεστ

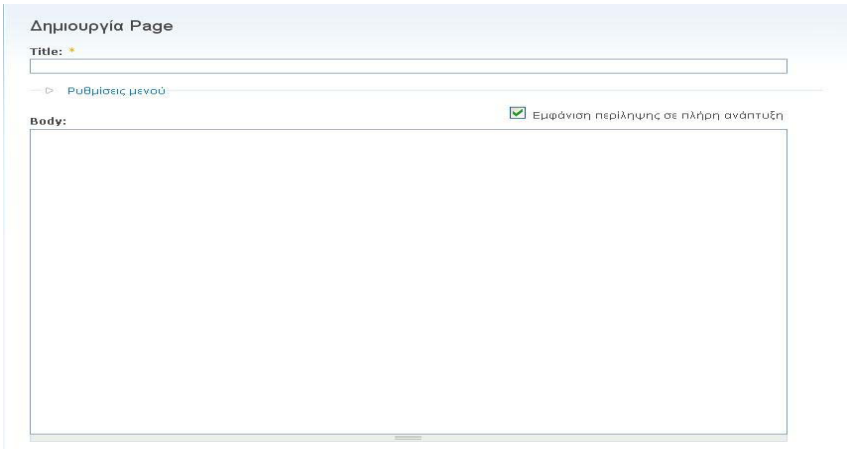
	ενότητας για προβολές.
Views UI:	Διοικητική διασύνδεση στις απόψεις. Χωρίς αυτό το module δεν θα μπορούσαμε να δημιουργήσουμε ή να επεξεργαστούμε τις απόψεις μας.

Άλλες μονάδες:

Blog :	Επιτρέπει την εύκολη και τακτική ενημέρωση ιστοσελίδων ή των ιστολογιών των χρηστών.
Blog API :	Επιτρέπει στους χρήστες να υποβάλλουν ύλη χρησιμοποιώντας εφαρμογές που υποστηρίζουν API ιστολογιών XML-RPC.
Book :	Επιτρέπει στους χρήστες να δομούν τις ιστοσελίδες ιεραρχικά ή περιληπτικά.
Contact :	Επιτρέπει να χρησιμοποιούνται φόρμες επικοινωνίας, προσωπικές ή κεντρικές, για ολόκληρο τον ιστότοπο.
Menu :	Επιτρέπει στους διαχειριστές να προσαρμόζουν το μενού πλοήγησης του ιστότοπου
Update status :	Ελέγχει την κατάσταση λειτουργίας για διαθέσιμες αναβαθμίσεις του Drupal και των εγκατεστημένων μονάδων και θεματικών παραλλαγών.

15 .Το διαχειριστικό περιβάλλον του Drupal:

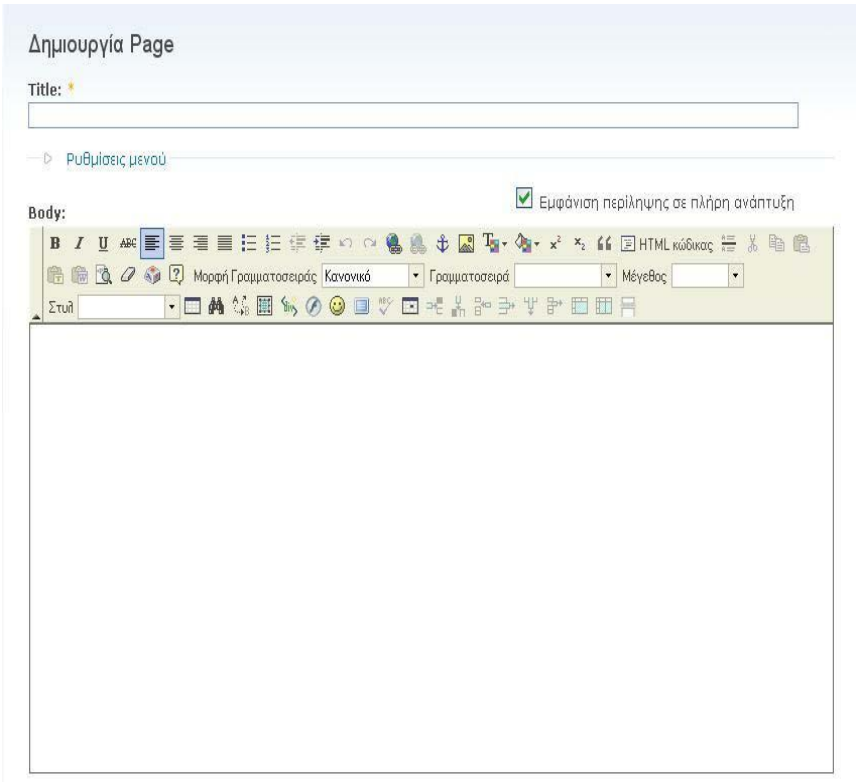
Δημιουργία κόμβου τύπου ύλης Page:



Εικόνα 26 : Δημιουργία κόμβου τύπου ύλης Page.

Παρόλο, που όπως είπαμε, με το Drupal θέλουμε να διευκολύνουμε την εισαγωγή περιεχομένου από έναν ανειδίκευτο χρήστη, με την βασική εγκατάσταση του Drupal, χρειάζεται γνώση html για να εισάγουμε το κυρίως σώμα του περιεχομένου. Για να επιτύχουμε το σκοπό μας, δηλαδή η εισαγωγή περιεχομένου να διευκολυνθεί, προσθέτουμε ένα module κατηγορίας user interface, το wysiwyg (what you see is what you get) και διαλέγοντας έναν client side editor όπως τον Fckeditor ή TinyMce η εισαγωγή περιεχομένου γίνεται σαν να δημιουργούμε έγγραφο του word δίνοντας μας πολλές επιλογές όπως εισαγωγή εικόνας, επιλογή ιδιοτήτων για τα γράμματα (γραμματοσειρά, μέγεθος, χρώμα) κτλ.

Δημιουργία κόμβου τύπου ύλης Page με WYSIWYG editor



Εικόνα 27 : WYSIWYG editor

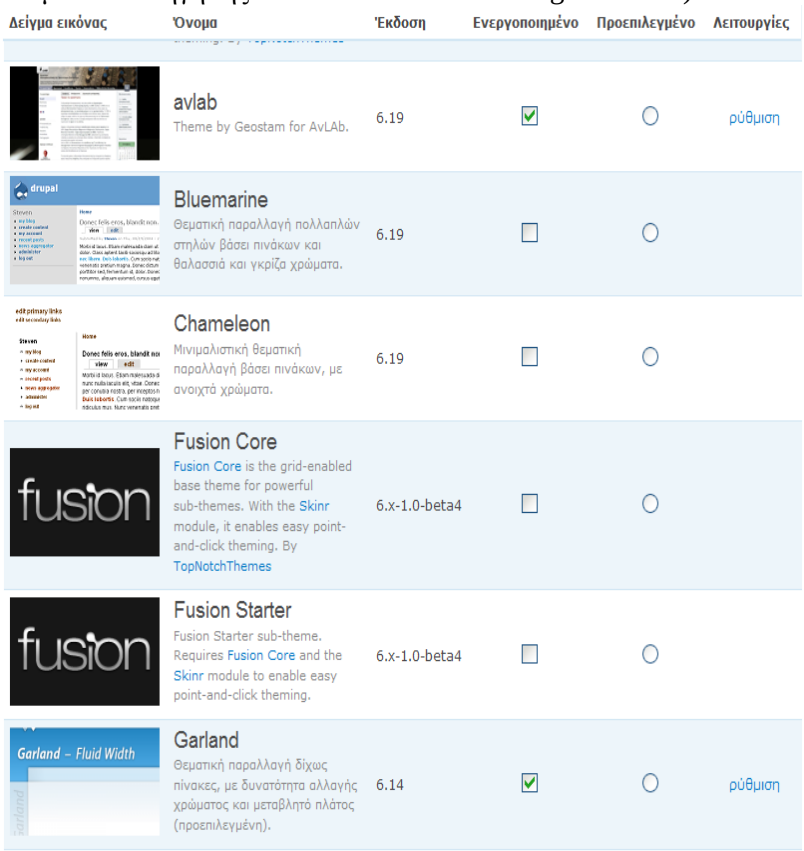
<u>Μέσα στο WYSIWYG editor ο κωδικας που έχει είναι :</u>

```
function hook_ckeditor_plugin() {
  return array(
    'plugin_name' => array(
      // Name of the plugin used to write it.
      'name' => 'plugin_name',
      // Description of the plugin - it would be displayed in the plugins management section of profile settings.
      'desc' => t('Plugin description'),
      // The full path to the CKEditor plugins directory, with the trailing slash.
```

```
    'path' => drupal_get_path('module', 'my_module') . '/plugin_dir/',
    'buttons' => array(
      'button_name' => array(
        'icon' => 'path to button icon',
        'label' => 'Button Label',
      )
    )
  )
 );
}
?>
```

15.1 Το πρότυπο σχεδίασης (Theme template)

Υπάρχουν πολλά έτοιμα θεματικά πρότυπα σχεδιασμένα από μέλη της Drupal κοινότητας που μπορεί κάποιος να χρησιμοποιήσει Αρχικά δημιουργήθηκε το αρχείο info με τις απαραίτητες πληροφορίες για το θεματικό πρότυπο της ιστοσελίδας μας και στη συνέχεια έγιναν Override τα σημαντικότερα templates των module αντιγράφοντας τα στο φάκελο του θεματικού μας προτύπου μας και τροποποιώντας τα ανάλογα με τις επιθυμίες μας. Έτσι, στο φάκελο του προτύπου μας εισήχθησαν αρχικά τα page.tpl, το node.tpl, το block.tpl και το View.tpl.
Μπορούν να επιλέξουν ένα theme για την ιστοσελίδα (μεταβαίνουμε σε αυτή την σελίδα επιλέγοντας από το μενού πλοήγησης Administer->Site building->Themes).



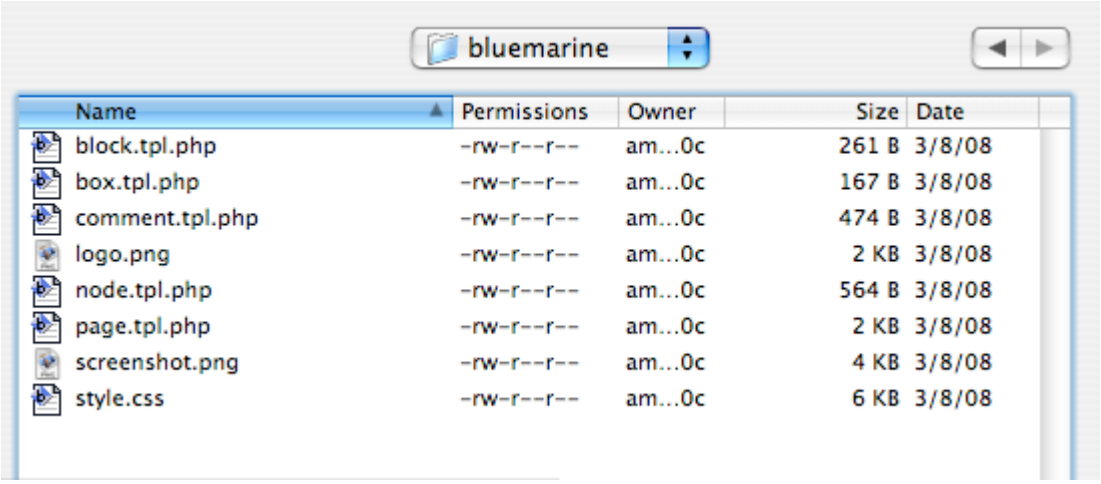
Εικόνα 28: Το πρότυπο σχεδίασης (Theme template)

Με την επιλογή configure δίπλα στο κάθε θέμα, μπορούμε να μεταβάλλουμε κάποια χαρακτηριστικά όπως το χρώμα, τις default εικόνες και logos, καθώς και να καθορίσουμε ποια στοιχεία της ιστοσελίδας (πχ primary links menu) θα είναι ορατά κλπ.

Στο drupal 6 η θεματική παραλλαγή περιέχει τα ακόλουθα αρχεία:

Όνομα.info :	Ένα απαραίτητο αρχείο που είναι νέο στο Drupal 6, το οποίο παρέχει πληροφορίες σχετικά με τη θεματική παραλλαγή.
Page.tpl.php	Το κύριο πρότυπο που καθορίζει το περιεχόμενο στο μεγαλύτερο μέρος της σελίδας.
Page.tpl.php,style:	Το CSS αρχείο που ορίζει κανόνες CSS για το πρότυπο.
Node.tpl.php :	Το αρχείο αυτό καθορίζει το περιεχόμενο των κόμβων.
Page.tpl.php :.	Προσδιορίζει το περιεχόμενο του μπλοκ
Comment.tpl.php :	Προσδιορίζει το περιεχόμενο των σχολίων.
Logo.png :	Το λογότυπό σας, αν χρησιμοποιείτε ένα.
Screenshot.png	Αυτή είναι μια εικόνα με το θέμα σας που χρησιμοποιείται στο admin panel και στις ρυθμίσεις του λογαριασμού χρήστη, αν έχετε ενεργοποιήσει περισσότερους από έναν θέμα έτσι ώστε οι επισκέπτες μπορούν να επιλέξουν ποιο θέμα θέλουν να χρησιμοποιήσουν.

να απαραίτητο αρχείο που είναι νέο στο Drupal 6, το οποίο παρέχει πληροφορίες σχετικά με τη θεματική παραλλαγή, όπου είναι το **bluemarine** και αυτό έχει μέσα τα παρακάτω αρχεία που περιλαμβάνουν τους κώδικες με το όποιο τα Themes μπορούν να λειτουργήσουν σωστά μέσα στο drupal .Ακολουθούν οι κώδικες που περιέχει μέσα τα Themes



Εικόνα 29 : Το bluemarine.

Μέσα στο φάκελο **bluemarine έχουμε** ένα **Screenshot.png** όπου μας παρέχει ένα αρχείο όπου θα είναι το λογότυπο μας (το οποίο χρησιμοποιείται από το Drupal για να προσδιορίσει το θέμα στην σελίδα μας που θα είναι το θέμα επιλογής) με στυλ όπου θα έχουμε πέντε αρχεία προτύπων php.

- **block.tpl.php:** ελέγχει πώς φαίνονται το μπλοκ.
- **comment.tpl.php:** Σχολιάζει.
- **Box.tpl.php:** Ελέγχει τα πλαίσια γύρω από τα στοιχείο, το πιο συνηθισμένο παράδειγμα είναι γύρω από τις μορφές σχόλιον.
- **Node.tpl.php :** Ελέγχει το πραγματικό περιεχόμενο μιας σελίδας, το ίδιο κόμβο.
- **Page.tpl.php:** Το κύριο πρότυπο που καθορίζει το περιεχόμενο στο μεγαλύτερο μέρος της σελίδας.

Αυτός είναι ο σκελετός του site μας

Το κύριο πρότυπο που καθορίζει το περιεχόμενο στο μεγαλύτερο μέρος της σελίδας δηλ.Η βασικη Page.tpl.php ειναι:

```
<?php
// $Id: page.tpl.php,v 1.28.2.1 2009/04/30 00:13:31 goba Exp $
?><!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" lang="<?php print $language->language ?>"
xml:lang="<?php print $language->language ?>" dir="<?php print $language->dir ?>">

<head>
  <?php print $head ?>
  <title><?php print $head_title ?></title>
  <?php print $styles ?>
  <?php print $scripts ?>
```

```
  <script type="text/javascript"><?php
/ ?> </script>
</head>

<body>

<table border="0" cellpadding="0" cellspacing="0" id="header">
  <tr>
    <td id="logo">
      <?php if ($logo) { ?><a href="<?php print $front_page ?>" title="<?php print t('Home') ?>"><img src="<?php print $logo ?>" alt="<?php print t('Home') ?>" /></a><?php } ?>
      <?php if ($site_name) { ?><h1 class='site-name'><a href="<?php print $front_page ?>" title="<?php print t('Home') ?>"><?php print $site_name ?></a></h1><?php } ?>
      <?php if ($site_slogan) { ?><div class='site-slogan'><?php print $site_slogan ?></div><?php } ?>
    </td>
    <td id="menu">
      <?php if (isset($secondary_links)) { ?><?php print theme('links', $secondary_links, array('class' => 'links', 'id' => 'subnavlist')) ?><?php } ?>
      <?php if (isset($primary_links)) { ?><?php print theme('links', $primary_links, array('class' => 'links', 'id' => 'navlist')) ?><?php } ?>
      <?php print $search_box ?>
    </td>
  </tr>
  <tr>
    <td colspan="2"><div><?php print $header ?></div></td>
  </tr>
</table>

<table border="0" cellpadding="0" cellspacing="0" id="content">
  <tr>
    <?php if ($left) { ?><td id="sidebar-left">
      <?php print $left ?>
    </td><?php } ?>
    <td valign="top">
      <?php if ($mission) { ?><div id="mission"><?php print $mission ?></div><?php } ?>
      <div id="main">
        <?php print $breadcrumb ?>
        <h1 class="title"><?php print $title ?></h1>
        <div class="tabs"><?php print $tabs ?></div>
        <?php if ($show_messages) { print $messages; } ?>
        <?php print $help ?>
        <?php print $content; ?>
        <?php print $feed_icons; ?>
      </div>
    </td>
    <?php if ($right) { ?><td id="sidebar-right">
      <?php print $right ?>
```

```
    </td><?php } ?>
  </tr>
</table>

<div id="footer">
  <?php print $footer_message ?>
  <?php print $footer ?>
</div>
<?php print $closure ?>
</body>
</html>
```

Το αρχείο αυτό καθορίζει το περιεχόμενο των κόμβων .Node.tpl.php :.

```
   <?php
   // $Id: node.tpl.php,v 1.7 2007/08/07 08:39:36 goba Exp $
   ?>
     <div class="node<?php if ($sticky) { print " sticky"; } ?><?php if (!$status) { print "
node-unpublished"; } ?>">
       <?php if ($picture) {
         print $picture;
       }?>
       <?php if ($page == 0) { ?><h2 class="title"><a href="<?php print $node_url?>"><?php
print $title?></a></h2><?php }; ?>
       <span class="submitted"><?php print $submitted?></span>
       <div class="taxonomy"><?php print $terms?></div>
       <div class="content"><?php print $content?></div>
       <?php if ($links) { ?><div class="links">&raquo; <?php print $links?></div><?php };
?>
     </div>
```

Προσδιορίζει το περιεχόμενο των σχολίων. comment.tpl.php ειναι:

```
<?php
// $Id: comment.tpl.php,v 1.7 2008/01/04 19:24:23 goba Exp $
?>
  <div class="comment<?php print ' '. $status; ?>">
    <?php if ($picture) {
    print $picture;
  } ?>
  <h3 class="title"><?php print $title; ?></h3><?php if ($new != '') { ?><span
class="new"><?php print $new; ?></span><?php } ?>
      <div class="submitted"><?php print $submitted; ?></div>
      <div class="content">
       <?php print $content; ?>
       <?php if ($signature): ?>
        <div class="clear-block">
         <div>—</div>
         <?php print $signature ?>
        </div>
       <?php endif; ?>
      </div>
      <div class="links">&raquo; <?php print $links; ?></div>
    </div>
```

15.2 Modules

Οι μονάδες του Drupal είναι plugins που μπορούν να εγκατασταθούν στο site της βάσης του Drupal για να βοηθήσουν την επέκταση ή την ενίσχυση της λειτουργικότητας του προγράμματος του πυρήνα του Drupal. Τα modules είναι ελεύθερα να τα κατεβάσει κάποιος και απαιτούν κάποια διαμόρφωση, αλλά είναι άριστα εργαλεία σχεδιασμένα για να μας βοηθήσουν να επιτύχουμε τα αποτελέσματα που θέλουμε από το δικτυακό μας τόπο με χρήση του Drupal. Επίσης η εγκατάσταση των επιπρόσθετων modules πραγματοποιείται ακριβώς με τον ίδιο τρόπο όπως και στα Themes. Εφόσον καταλήξουμε στο module που επιθυμούμε να εγκαταστήσουμε, πραγματοποιούμε λήψη του. Έπειτα αποσυμπιέζουμε το αρχείο και το αντιγράφουμε στον φάκελο public_html/sites/all/modules. τον οποίο έχουμε δημιουργήσει. Έπειτα μεταβαίνουμε στην σελίδα των modules (Asministrator Menu – Site Bulding –Modules – List και το ενεργοποιούμε τσεκάροντας το αντίστοιχο κουτάκι και κάνοντας κλικ στο κουμπί Save configuration.

Ένα Module έχει ένα υπο-φάκελους όπου εκεί βάζει τον κώδικα PHP όπου είναι :

- **php.info :** Το αρχείο του **.info** είναι ένα απλό αρχείο κειμένων που δίνει τις αναγκαίες πληροφορίες στο πυρήνα Drupal για την ενότητα μας.
- **php.module:** Το **.module περιέχει** τον κώδικα php που ο πυρήνας Drupal θα φορτώσει και θα εκτελέσει.
- **Υπάρχει και το Hooks:** Ένα από τα ισχυρά χαρακτηριστικά γνωρίσματα της αρχιτεκτονικής του Drupal είναι οι ειδικά ονομασμένες λειτουργίες που επιτρέπουν στην ενότητα μας για να

αλληλεπιδρά με τον πυρήνα Drupal. Αυτές οι ειδικές λειτουργίες αναφέρονται ως **Hooks** επειδή αφήνουν την ενότητα μας κολλάει στον υπόλοιπο του πυρήνα Drupal.

15.2.1 Τα Drupal Hooks

Drupal Hooks: Τα hooks επιτρέπουν τα modules να αλληλεπιδρούν με τον πυρήνα του Drupal (Drupal core).Το σύστημα των module του Drupal βασίζεται στην γενική ιδέα των "hooks" (άγκιστρα). Το hook είναι μία PHP λειτουργία (function) η οποία ονομάζεται foo_bar(), όπου "foo" είναι το όνομα του module (του οποίου το όνομα αρχείου είναι foo.module) και "bar" είναι το όνομα του hook. Το κάθε hook έχει ένα καθορισμένο σύνολο από παραμέτρους κι ένα καθορισμένο τύπο αποτελεσμάτων.

Για να επεκταθεί το Drupal, θα πρέπει απλά να εφαρμοστεί ένα hook από ένα module. Όταν το Drupal επιθυμεί να επιτρέψει μεσολάβηση από τα modules, καθορίζει το ποια modules εφαρμόζουν κάποιο hook και καλεί αυτό το hook σε όλα τα ενεργοποιημένα modules που το εφαρμόζουν.

Λειτουργίες και Μέθοδοι

Όνομα	Περιγραφή
custom_url_rewrite_inbound:	Δεν είναι hook, είναι ένα function που μπορούμε να προσθέσουμε στο settings.php για να μεταβάλλουμε τα εισερχόμενα αιτήματα έτσι ώστε να δείξουν στο path του Drupal. Αυτό το function καλείται πριν φορτωθούν τα modules και το σύστημα του μενού αρχικοποιείται και αλλάζει.
custom_url_rewrite_outbound:	Δεν είναι hook, είναι μία function που μπορούμε να προσθέσουμε στο settings.php για να μεταβάλλουμε όλους τους συνδέσμους που έχουν παραχθεί από το Drupal. Αυτό το function καλείται από το url().
hook_access:	Ορίζει τους περιορισμούς στην πρόσβαση.
hook_actions_delete:	Εκτελεί κώδικά μετά από τη διαγραφή ενός action.
hook_action_info:	Δηλώνει πληροφορίες σχετικά με τα actions.
hook_action_info_alter:	Τροποποιεί τα actions που δηλώθηκαν από ένα άλλο module.
hook_block:	Δηλώνει ένα μπλοκ ή ένα σύνολο από μπλοκ.
hook_boot:	Εκτελεί εργασίες εγκατάστασης.
hook_comment:	Ανταποκρίνεται στις ενέργειες των χολίων.

hook_cron:	Εκτελεί περιοδικές ενέργειες.
hook_db_rewrite_sql:	Επαναγράφει ερωτήματα της βάσης δεδομένων. Συνήθως για έλεγχο της πρόσβασης.
hook_delete:	Ανταποκρίνεται στη διαγραφή κόμβου.
hook_disable:	Εκτελεί τις απαραίτητες ενέργειες πριν απενεργοποιηθεί το module.
hook_elements:	Επιτρέπει τα modules να δηλώσουν τις δικούς τους τύπους των στοιχείων των Forms API και να καθορίσουν τις προκαθορισμένες τιμές.
Hook_enable:	Διενεργεί τις απαραίτητες ενέργειες μετά από την ενεργοποίηση του module.
hook_exit:	Διενεργεί διεργασίες καθαρισμού.
hook_file_download:	Ελέγχει την πρόσβαση στη λήψη ιδιωτικών αρχείων και καθορίζει τις HTTP κεφαλίδες.
hook_filter:	Ορίζει τα φίλτρα περιεχομένου.
hook_filter_tips:	Παρέχει συμβουλές για τη χρήστη των φίλτρων.
hook_flush_caches:	Προσθέτει μία λίστα από πίνακες που βρίσκονται στη μνήμη cache για να εκκαθαριστούν.
hook_footer:	Εισάγει το κλείσιμο σε HTML.
hook_form:	Προβάλει μία φόρμα επεξεργασίας ενός κόμβου.
hook_forms:	Καθορίζει τις form_ids στις builder functions.
hook_form_alter:	Εκτελεί τροποποιήσεις πριν η φόρμα καταστεί.
hook_form_FORM_ID_alter:	Παρέχει μία συγκεκριμένη τροποποίηση της φόρμας αντί για το globalhook_form_alter().
hook_help:	Παρέχει βοήθεια στο χρήστη σε απευθείας σύνδεση.
hook_hook_info:	Εκθέτει μία λίστα από γεγονότα στα οποία οι χρήστες μπορούν να αναθέσουν ενέργειες.
hook_init:	Εκτελεί εργασίες εγκατάστασης.
hook_insert:	Ανταποκρίνεται στην εισαγωγή κόμβου.

hook_install:	Εγκαθιστά την τρέχουσα έκδοση του σχήματος της βάσης δεδομένων, και ότι άλλες διεργασίες εγκατάστασης υπάρχουν.
hook_link:	Ορίζει τους εσωτερικούς συνδέσμους του Drupal.
hook_link_alter:	Διενεργεί τροποποιήσεις πριν να παρασχεθούν οι σύνδεσμοι σε έναν κόμβο ή τα σχόλια.
hook_load:	Φορτώνει πληροφορίες συγκεκριμένων τύπων κόμβων.
hook_locale:	Επιτρέπει στα modules να ορίσουν τις δικές τους ομάδες κειμένου όπου μπορούν να μεταφραστούν.
hook_mail:	Προετοιμάζει ένα μήνυμα βασισμένο σε παραμέτρους, το οποίο καλείται από το drupal_mail().
hook_mail_alter:	Τροποποιεί οποιαδήποτε όψη των email που αποστέλλονται από το Drupal.
hook_menu:	Ορίζει τα αντικείμενα των μενού και τις επανακλήσεις των σελίδων.
hook_menu_alter:	Τροποποιεί τα δεδομένα που έχουν αποθηκευτεί στον πίνακα {menu_router} αφου επικαλεσθεί το after hook_menu.
hook_menu_link_alter:	Τροποποιεί τα δεδομένα που έχουν αποθηκευτεί στον πίνακα {menu_links} από το menu_link_save().
hook_nodeapi:	Ενεργεί στους κόμβους που έχουν οριστεί από άλλα modules.
hook_node_access_records:	Θέτει τις άδειες που θα γραφτούν στην βάση δεδομένων για έναν κόμβο.
hook_node_grants:	Ενημερώνει το σύστημα πρόσβασης του κόμβου για το τι άδειες έχει ένας χρήστης.
hook_node_info:	Ορίζει τους τύπους κόμβου που παρέχονται από module.
hook_node_operations:	Προσθέτει μαζικές λειτουργίες κόμβου.
hook_node_type:	Ενεργεί στις αλλαγές του τύπου κόμβου.
hook_openid:	Επιτρέπει στα modules να τροποποιήσουν τις παραμέτρους των αιτημάτων OpenID.
hook_perm:	Ορίζει τις άδειες του χρήστη.

hook_ping:	Εκτελεί τη λειτουργία ping σε έναν άλλο διακομιστή.
hook_prepare:	Αυτό είναι ένα hook το οποίο χρησιμοποιείται από modules κόμβων. Καλείται μετά τη φόρτωση αλλά πριν την εμφάνιση του κόμβου στη φόρμα προσθήκης/επεξεργασίας.
hook_preprocess:	Προεπεξεργάζεται τις μεταβλητές του θέματος για τα αρχεία προτύπων.
hook_preprocess_HOOK:	Προεπεξεργάζεται τις μεταβλητές του θέματος για ένα συγκεκριμένο hook θέματος.
hook_profile_alter:	Τροποποιεί τα αντικείμενα προφίλ πριν αυτά αποδοθούν.
hook_requirements:	Ελέγχει τις απαιτήσεις τις εγκατάστασης και κάνει αναφορά κατάστασης.
hook_schema:	Ορίζει την τρέχουσα έκδοση του σχήματος της βάσης δεδομένων.
hook_schema_alter:	Διενεργεί τροποποιήσεις σε υφιστάμενα σχήματα βάσης δεδομένων.
hook_search:	Ορίζει μία προσαρμοσμένη ρουτίνα αναζήτησης.
hook_search_page:	Παρακάμπτει την απόδοση των αποτελεσμάτων αναζήτησης.
hook_search_preprocess:	Προεπεξεργάζεται κείμενο για το ευρετήριο αναζήτησης.
hook_system_info_alter:	Τροποποιεί την πληροφορία που αναλύθηκε από τα αρχεία πληροφορίας του module και του θέματος.
hook_taxonomy:	Ενεργεί στις αλλαγές ταξινόμησης.
hook_term_path:	Επιτρέπει στα modules να παρέχουν ένα εναλλακτικό path για τους όρους τους οποίους διαχειρίζεται.
hook_theme:	Καταχωρεί τις εκτελέσεις του θέματος ενός module ή ενός θέματος.
hook_theme_registry_alter:	Τροποποιεί τις πληροφορίες του μητρώου του θέματος που επιστράφηκαν από το hook_theme().
hook_translated_menu_link_alter:	Τροποποιεί έναν σύνδεσμο μενού αφού μεταφραστεί και πριν αποδοθεί.
hook_translation_link_alter:	Διενεργεί τροποποιήσεις σε συνδέσμους μετάφρασης.

hook_uninstall:	Αφαιρεί οποιαδήποτε πληροφορία θέτει το module.
hook_update:	Ανταποκρίνεται στην ενημέρωση του κόμβου.
hook_update_index:	Ενημερώνει το ευρετήριο του πλήρους κειμένου του Drupal γι αυτό το module.
hook_update_last_removed:	Επιστρέφει έναν αριθμό ο οποίος δεν είναι πλέον διαθέσιμος, όπως το hook_update_N().
hook_update_N:	Διενεργεί μία μόνο ενημέρωση.
hook_update_projects_alter:	Τροποποιεί τη λίστα των project πριν πάρει δεδομένα και συγκρίνει τις εκδόσεις.
hook_update_status_alter:	Τροποποιεί τις πληροφορίες για τις διαθέσιμες ενημερώσεις για τα projects.
hook_user:	Ενεργεί στις ενέργειες του λογαριασμού χρήστη.
hook_user_operations:	Προσθέτει μαζικές λειτουργίες του χρήστη.
hook_validate:	Επαληθεύει τη φόρμα επεξεργασίας ενός κόμβου.
hook_view:	Προβάλει έναν κόμβο.
hook_watchdog:	Καταγράφει ένα μήνυμα συμβάντος.
hook_xmlrpc:	Εγγράφει επανακλήσεις XML-RPC.
module_hook:	Καθορίζει το εάν κάποιο module εφαρμόζει ένα hook.
module_implements:	Καθορίζει το ποια modules εφαρμόζουν ένα hook.
module_invoke:	Επικαλείται ένα hook σε συγκεκριμένο module.
module_invoke_all:	Επικαλείται ένα hook σε όλα τα ενεργοποιημένα modules που το εφαρμόζουν.
page_cache_fastpath:	Εξάγει μία αποθηκευμένη σελίδα.

Ο κώδικας που έχει μέσα ένα blog.module είναι:

```
<?php
// $Id: blog.module,v 1.297.2.4 2009/02/25 12:21:53 goba Exp $
/**
 * @file
 * Enables keeping an easily and regularly updated web page or a blog.
 */
/**
 * Ορίζει τους τύπους κόμβου που παρέχονται από module .hook_node_info().
 */
function blog_node_info() {
  return array(
    'blog' => array(
      'name' => t('Blog entry'),
      'module' => 'blog',
      'description' => t('A <em>blog entry</em> is a single post to an online journal, or
<em>blog</em>.'),
    )
  );
}

 * Ορίζει τις άδειες του χρήστη.  of hook_perm().
 */
function blog_perm() {
  return array('create blog entries', 'delete own blog entries', 'delete any blog entry', 'edit own blog
entries', 'edit any blog entry');
}

/**
 * Implementation of hook_access().
 */
function blog_access($op, $node, $account) {
  switch ($op) {
    case 'create':
      // Anonymous users cannot post even if they have the permission.
      return user_access('create blog entries', $account) && $account->uid ? TRUE : NULL;
    case 'update':
      return user_access('edit any blog entry', $account) || (user_access('edit own blog entries',
$account) && ($node->uid == $account->uid)) ? TRUE : NULL;
    case 'delete':
      return user_access('delete any blog entry', $account) || (user_access('delete own blog entries',
$account) && ($node->uid == $account->uid)) ? TRUE : NULL;
  }
}
```

```
/**
 * Implementation of hook_user().
 */
function blog_user($type, &$edit, &$user) {
  if ($type == 'view' && user_access('create blog entries', $user)) {
    $user->content['summary']['blog'] =  array(
      '#type' => 'user_profile_item',
      '#title' => t('Blog'),
      // l() escapes the attributes, so we should not escape !username here.
      '#value' => l(t('View recent blog entries'), "blog/$user->uid", array('attributes' => array('title' =>
t("Read !username's latest blog entries.", array('!username' => $user->name))))),
      '#attributes' => array('class' => 'blog'),
    );
  }
///....  }
}
Καθορίζει from_ids στις builder functions of hook_form().
 */
function blog_form(&$node) {
  global $nid;
  $iid = isset($_GET['iid']) ? (int)$_GET['iid'] : 0;
  $type = node_get_types('type', $node);


  if (empty($node->body)) {
    // If the user clicked a "blog it" link, we load the data from the
    // database and quote it in the blog.
    if ($nid && $blog = node_load($nid)) {
      $node->body = '<em>'. $blog->body .'</em> ['. l($blog->name, "node/$nid") .']';
    }

    if ($iid && $item = db_fetch_object(db_query('SELECT i.*, f.title as ftitle, f.link as flink FROM
{aggregator_item} i, {aggregator_feed} f WHERE i.iid = %d AND i.fid = f.fid', $iid))) {
      $node->title = $item->title;
      // Note: $item->description has been validated on aggregation.
      $node->body = '<a href="'. check_url($item->link) .'">'. check_plain($item->title) .'</a> - <em>'.
$item->description .'</em> [<a href="'. check_url($item->flink) .'">'. check_plain($item->ftitle)
."</a>]\n";
    }
  }

//...
}
```

```
/**
 * Implementation of hook_view().
 */
function blog_view($node, $teaser = FALSE, $page = FALSE) {
  if ($page) {
    // Breadcrumb navigation. l() escapes the title, so we should not escape !name.
    drupal_set_breadcrumb(array(l(t('Home'), NULL), l(t('Blogs'), 'blog'), l(t("'!name's blog",
array('!name' => $node->name)), 'blog/'. $node->uid)));
  }
  return node_prepare($node, $teaser);
}

/**
 * Implementation of hook_link().
 */
function blog_link($type, $node = NULL, $teaser = FALSE) {
  $links = array();

  if ($type == 'node' && $node->type == 'blog') {
    if (arg(0) != 'blog' || arg(1) != $node->uid) {
      // This goes to l() and therefore escapes !username in both the title and attributes.
      $links['blog_usernames_blog'] = array(
        'title' => t("'!username's blog", array('!username' => $node->name)),
        'href' => "blog/$node->uid",
        'attributes' => array('title' => t("Read !username's latest blog entries.", array('!username' =>
$node->name)))
      );
    }
  }

  return $links;
}

/**
 * Implementation of hook_menu().
 */
function blog_menu() {
  $items['blog'] = array(
    'title' => 'Blogs',
    'page callback' => 'blog_page_last',
    'access arguments' => array('access content'),
    'type' => MENU_SUGGESTED_ITEM,
    'file' => 'blog.pages.inc',
  );
  $items['blog/%user_uid_optional'] = array(
    'title' => 'My blog',
    'page callback' => 'blog_page_user',
    'page arguments' => array(1),
```

```
    'access callback' => 'blog_page_user_access',
    'access arguments' => array(1),
    'file' => 'blog.pages.inc',
  );
  $items['blog/%user/feed'] = array(
    'title' => 'Blogs',
    'page callback' => 'blog_feed_user',
    'page arguments' => array(1),
    'access callback' => 'blog_page_user_access',
    'access arguments' => array(1),
    'type' => MENU_CALLBACK,
    'file' => 'blog.pages.inc',
  );
  $items['blog/feed'] = array(
    'title' => 'Blogs',
    'page callback' => 'blog_feed_last',
    'access arguments' => array('access content'),
    'type' => MENU_CALLBACK,
    'file' => 'blog.pages.inc',
  );

  return $items;
}

/**
 * Access callback for user blog pages.
 */
function blog_page_user_access($account) {
  // The visitor must be able to access the site's content.
  // For a blog to 'exist' the user must either be able to
  // create new blog entries, or it must have existing posts.
  return $account->uid && user_access('access content') && (user_access('create blog entries',
$account) || _blog_post_exists($account));
}

/**
 * Helper function to determine if a user has blog posts already.
 */
function _blog_post_exists($account) {
  return (bool)db_result(db_query_range(db_rewrite_sql("SELECT 1 FROM {node} n WHERE
n.type = 'blog' AND n.uid = %d AND n.status = 1"), $account->uid, 0, 1));
}
```

```
/**
 * Implementation of hook_block().
 *
 * Displays the most recent 10 blog titles.
 */
function blog_block($op = 'list', $delta = 0) {
  global $user;
  if ($op == 'list') {
    $block[0]['info'] = t('Recent blog posts');
    return $block;
  }
  else if ($op == 'view') {
    if (user_access('access content')) {
      $result = db_query_range(db_rewrite_sql("SELECT n.nid, n.title, n.created FROM {node} n
WHERE n.type = 'blog' AND n.status = 1 ORDER BY n.created DESC"), 0, 10);
      if ($node_title_list = node_title_list($result)) {
        $block['content'] = $node_title_list;
        $block['content'] .= theme('more_link', url('blog'), t('Read the latest blog entries.'));
        $block['subject'] = t('Recent blog posts');
        return $block;
      }
    }
  }
}
```

Ο κωδικας που έχει μέσα ένα php.info είναι ο παρακάτω :

```
 ; $Id: php.info,v 1.2 2007/06/08 05:50:55 dries Exp $
 name = PHP filter
 description = Allows embedded PHP code/snippets to be evaluated.
 package = Core - optional
 version = VERSION
 core = 6.x

 ; Information added by drupal.org packaging script on 2010-06-02
 version = "6.17"
 project = "drupal"
 datestamp = "1275505216"
```

Ο κωδικας που έχει μέσα ένα php.install είναι ο παρακάτω:

```
<?php
// $Id: php.install,v 1.1 2007/04/24 10:54:34 dries Exp $

/**
 * Implementation of hook_install().
 */
function php_install() {
  $format_exists = db_result(db_query("SELECT COUNT(*) FROM {filter_formats} WHERE name
= 'PHP code'"));
  // Add a PHP code input format, if it does not exist. Do this only for the
  // first install (or if the format has been manually deleted) as there is no
  // reliable method to identify the format in an uninstall hook or in
  // subsequent clean installs.
  if (!$format_exists) {
    db_query("INSERT INTO {filter_formats} (name, roles, cache) VALUES ('PHP code', '', 0)");
    $format = db_result(db_query("SELECT MAX(format) FROM {filter_formats}"));

    // Enable the PHP evaluator filter.
    db_query("INSERT INTO {filters} (format, module, delta, weight) VALUES (%d, 'php', 0, 0)",
$format);

    drupal_set_message(t('A !php-code input format has been created.', array('!php-code' => l('PHP
code', 'admin/settings/filters/'. $format))));
  }
}
```

```
//…
  }
* Implementation of hook_filter(). Contains a basic PHP evaluator.
 *
 * Executes PHP code. Use with care.
 */
function php_filter($op, $delta = 0, $format = -1, $text = '') {
  switch ($op) {
    case 'list':
      return array(0 => t('PHP evaluator'));
    case 'no cache':
      // No caching for the PHP evaluator.
      return $delta == 0;
    case 'description':
      return t('Executes a piece of PHP code. The usage of this filter
should be restricted to administrators only!');
    case 'process':
      return drupal_eval($text);
    default:
      return $text;
  }
}
```

15.3 Διαδικασία αίτησης σελίδας

Όταν θέλουμε να επισκεφθούμε μια σελίδα που έχει δημιουργηθεί με Drupal, ο browser στέλνει το url στον Web Server. Το κομμάτι του URL μετά το domain name είναι το path από το οποίο το Drupal θα αποφασίσει τι πληροφορίες θα στείλει στον browser μέσω ενός ή περισσοτέρων ερωτημάτων στη βάση δεδομένων (db queries) που δημιουργούνται από τον PHP κώδικα των modules και του Προτύπου σχεδίασης, βάσει των δικαιωμάτων του χρήστη και αντλώντας τα δεδομένα από τη βάση δεδομένων. Έτσι, ο Web Server θα κατασκευάσει μια html σελίδα με τα δεδομένα που έχει συλλέξει και θα την στείλει στον browser. Πιο συγκεκριμένα, αν έχουμε clean url, το module του Web Server που είναι υπεύθυνο για το rewrite θα ξεχωρίσει το base url από το path (mod_rewrite στον apache, ISAP rewrite στον IIS).

Το αποτέλεσμα είναι το http://example.com/index.php?q=test/1 από το οποίο καταλαβαίνουμε ότι όλες οι αιτήσεις περνάνε από το index.php αρχείο με παράμετρο το path. Το **index.php** σε κάθε αίτηση καλεί το bootstrap.inc για να κάνει τις απαραίτητες αρχικοποιήσεις, όπως να φορτώσει το settings.php, να συνδέσει τη βάση δεδομένων και ανάλογα με το σύστημα της βάσης να φορτώσει τις απαραίτητες βιβλιοθήκες που θα χρησιμοποιηθούν στο db.abstraction layer (π.χ αν είναι MySQL φορτώνει το database.mysql.inc) και επίσης να φορτώσει όλες τις βιβλιοθήκες συναρτήσεων του Drupal, όλα τα ενεργοποιημένα modules, την υποστήριξη για το theme system και το callback mapping. Στη φάση αυτή έχουν φορτωθεί όλα τα απαραίτητα στοιχεία και είναι διαθέσιμα ώστε να γίνει το mapping μεταξύ του path και των συναρτήσεων που θα παράξουν το βασικό περιεχόμενο αυτής της σελίδας. Πιο συγκεκριμένα, καλούνται όλα τα modules που έχουν εφαρμόσει, δηλαδή είναι υπεύθυνα για τη δημιουργία μιας σελίδας. Έπειτα, αναζητείται στον πίνακα "menu_router" το item με πρωτεύον κλειδί το συγκεκριμένο path και καλείται η συνάρτηση αυτού του item για να παράγει το αποτέλεσμα, αν βέβαια ο χρήστης έχει τα απαραίτητα δικαιώματα. Το αποτέλεσμα αυτό

(περιεχόμενο χωρίς μορφοποίηση) περνάει στην **index.php** σαν μεταβλητή με όνομα "return" και καλείται η "theme (page,return)" για να το μορφοποιήσει σύμφωνα με το page.tpl αρχείο. Το theme system κάνει μια προδιεργασία για τη διάταξη της σελίδας και τις περιοχές (regions) και καλεί το "theme(block,regions)" το οποίο θα εισάγει τα blocks, που θα πλαισιώσουν το βασικό περιεχόμενο, στα regions και εμφανίζει την τελική σελίδα με το περιεχόμενο και τα blocks μορφοποιημένα σύμφωνα με τις κατάλληλες συναρτήσεις ή αρχεία.

Οι κωδικας για να εκτελεστει το index.php είναι:

```
require_once './includes/bootstrap.inc';
drupal_bootstrap(DRUPAL_BOOTSTRAP_FULL);
$return = menu_execute_active_handler();
// Menu status constants are integers; page content is a string.
if (is_int($return)) {
  switch ($return) {
    case MENU_NOT_FOUND:
      drupal_not_found();
      break;
    case MENU_ACCESS_DENIED:
      drupal_access_denied();
      break;
    case MENU_SITE_OFFLINE:
      drupal_site_offline();
      break;
  }
}
elseif (isset($return)) {
  // Print any value (including an empty string) except NULL or undefined:
  print theme('page', $return);
}
drupal_page_footer();
```

15.4 Database Abstraction Layer

Επιτρέπει τη χρήση διαφορετικών διακομιστών βάσεων δεδομένων (database servers) να χρησιμοποιούν την ίδια βάση κώδικα.

Το Drupal παρέχει ένα λεπτό επίπεδο αφαίρεσης βάσης δεδομένων (database abstraction layer) για να δίνει στους προγραμματιστές τη δυνατότητα να υποστηρίζουν πολλαπλούς διακομιστές βάσεων δεδομένων εύκολα. Η πρόθεση αυτού του επιπέδου είναι να διαφυλάττει τη σύνταξη και τη δύναμη του SQL όσο μπορεί περισσότερο, ενώ ταυτόχρονα να επιτρέπει στο Drupal να ελέγχει τα κομμάτια από ερωτήματα (queries) που χρειάζονται να γραφούν διαφορετικά για διαφορετικούς διακομιστές και να παρέχουν βασικούς ελέγχους ασφαλείας.

Τα περισσότερα ερωτήματα βάσεων δεδομένων του Drupal εκτελούνται από μία κλήση στο db_query() ή στο db_query_range(). Οι συντάκτες των modules θα πρέπει να λάβουν υπ' όψην τη χρήση του pager_query() για τα ερωτήματα που επιστρέφουν αποτελέσματα τα οποία θα εμφανίζονται σε πολλαπλές σελίδες, και του tablesort_sql() για τη δημιουργία κατάλληλων ερωτημάτων για ταξινομημένα tables.

Για παράδειγμα, κάποιος μπορεί να επιθυμεί να λάβει τη λίστα με τους 10 πιο πρόσφατους κόμβους που έχει συντάξει κάποιος χρήστης. Αντί να εκδίδουμε απευθείας το SQL ερώτημα

```
SELECT n.nid,. n.tittle, n.created FROM node n WHERE n,uid = $uid LIMIT 0, 10;
```

θα μπορούσαμε να καλέσουμε τις functions του Drupal:

```
$result = db_query_range('SELECT n.nid, n.title, n.created
  FROM {node} n WHERE n.uid = %d', $uid, 0, 10);
while ($node = db_fetch_object($result)) {
  // Perform operations on $node->body, etc. here.
}
```

Λειτουργίες και Μέθοδοι

Όνομα	Περιγραφή
db_affected_rows:	Καθορίζει τον αριθμό των σειρών που άλλαξαν από το προηγούμενο ερώτημα.
db_check_setup:	Επιβεβαιώνει το εάν η βάση δεδομένων έχει δημιουργηθεί σωστά.
db_column_exists:	Ελέγχει εάν μία στήλη υπάρχει στο δεδομένο πίνακα.
db_connect:	Αρχικοποιεί μία σύνδεση βάσης δεδομένων.
db_decode_blob:	Επιστρέφει κείμενο από μία τιμή Binary Large OBject.
db_distinct_field:	Προσθέτει την σημαία (flag) DISTINCT στο παρεχόμενο ερώτημα και επιστρέφει το τροποποιημένο ερώτημα.
db_encode_blob:	Επιστρέφει μια σωστά διαμορφωμένη τιμή Binary Large OBject.
db_error:	Προσδιορίζει το εάν το προηγούμενο ερώτημα δημιούργησε σφάλμα.
db_escape_string:	Προετοιμάζει τη χρήση εισαγωγής από τον χρήστη σε μία βάση δεδομένων, αποτρέπει

	επιθέσεις τύπου SQL injection.
db_escape_table:	Περιορίζει ένα δυναμικό πίνακα, στήλη, ή περιορισμένο όνομα, σε ασφαλή χαρακτήρες.
db_fetch_array:	Προσκομίζει μία σειρά από αποτελέσματα από προηγούμενο ερώτημα σαν συστοιχία.
db_fetch_object:	Προσκομίζει μία σειρά από αποτελέσματα από προηγούμενο ερώτημα σαν αντικείμενο.
db_is_active:	Επιστρέφει μία τιμή boolean η οποία εξαρτάται από την διαθεσιμότητα της βάσης δεδομένων.
db_last_insert_id:	Επιστρέφει το τελευταίο αναγνωριστικό που εισήχθη.
db_lock_table:	Κλειδώνει έναν πίνακα.
db_placeholders:	Παράγει δείκτες για ένα array από arguments ερωτημάτων ενός μόνο τύπου.
db_prefix_tables:	Επισυνάπτει ένα πρόθεμα βάσης δεδομένων μέσω ερωτήματος σε όλους τους πίνακες.
db_query:	Εκτελεί ένα βασικό ερώτημα στη βάση δεδομένων. active database.
db_query_range:	Εκτελεί ένα ερώτημα περιορισμένου εύρους στην ενεργή βάση δεδομένων.
db_query_temporary:	Εκτελεί ένα ερώτημα SELECT και αποθηκεύει τα αποτελέσματά του σε έναν προσωρινό πίνακα.
db_result:	Επιστρέφει ένα ατομικό πεδίο αποτελεσμάτων από προηγούμενο ερώτημα.
db_rewrite_sql:	Επαναγράφει ερωτήματα κόμβου, ταξινόμησης και σχολίων.
db_set_active:	Ενεργοποιεί μία βάση δεδομένων για μελλοντικά ερωτήματα.
db_status_report:	Αναφέρει την κατάσταση της βάσης δεδομένων.
db_table_exists:	Ελέγχει εάν υπάρχει ο πίνακας.
db_unlock_tables:	Ξεκλειδώνει όλους τους κλειδωμένους πίνακες.
db_version:	Επιστρέφει την έκδοση του διακομιστή της βάσης δεδομένων η οποία χρησιμοποιείται.
pager_query:	Εκτελεί ένα σελιδοποιημένο ερώτημα βάσης δεδομένων.
tablesort_sql:	Δημιουργεί μία SQL ρήτρα του είδους.
update_sql:	Εκτελεί ένα ερώτημα SQL και επιστρέφει το εάν ήταν επιτυχής ή όχι.
_db_error_page:	Βοηθητικό function το οποίο δείχνει μοιραία σφάλματα της βάσης δεδομένων.
_db_query:	Βοηθητικό function για το db_query().
_db_query_callback:	Βοηθητικό function για το db_query().
_db_rewrite_sql:	Βοηθητικό function για το db_rewrite_sql.

Σταθερές

Όνομα	Περιγραφή
DB_QUERY_REGEXP:	Υποδεικνύει το μέρος όπου θα πρέπει να αντικατασταθούν οι holders στο _db_query_callback().

15.5 Υποβολή περιεχομένου

Για να εισάγουμε εύκολα περιεχόμενο στον ιστότοπο χρειαζόμαστε μία φόρμα εισαγωγής στοιχείων και τα στοιχεία αυτά να αποθηκευθούν μέσω ενός ερωτήματος (db query) στη βάση δεδομένων. Στο Drupal αυτό γίνεται με το module του πυρήνα "node", το οποίο παρέχει μια σελίδα διαχείρισης με φόρμα εισαγωγής στοιχείων καθώς και τον απαραίτητο κώδικα για την αποθήκευση των εισαγόμενων στοιχείων σε κατάλληλους πίνακες. Με τη διαδικασία αυτή δημιουργούμε έναν κόμβο (node) και τα στοιχεία αποθηκεύονται στους πίνακες "node" και "node revision". Στον πίνακα "node" τοποθετούνται κάποια βασικά meta data στοιχεία του κόμβου, όπως το nid, το uid, ο τύπος του, η γλώσσα του κλπ., ενώ στον πίνακα "node_revisions" ο τίτλος του κόμβου και το κυρίως σώμα του.

Ο κάθε κόμβος έχει url «node/nid», όπου nid ο μοναδικός αριθμός που αντιστοιχεί σε κάθε κόμβο, με αποτέλεσμα, όταν γίνει μια αίτηση (request) σε path αυτής της μορφής, το menu system να ψάξει να βρει τον κόμβο που αντιστοιχεί στο path της αίτησης (request), να καλέσει το module "node" με όρισμα το nid και εκεί να γίνει ένα ερώτημα για να αντληθούν από τη βάση τα στοιχεία που έχουμε εισάγει. Στη συνέχεια το αποτέλεσμα μορφοποιείται με κλήση της συνάρτησης theme_node από το module node και με βάση το node.tpl.php εμφανίζονται τα επιθυμητά στοιχεία μορφοποιημένα.

Οι έτοιμοι τύποι περιεχομένου (content types) που προϋπάρχουν με την εγκατάσταση είναι το "page" και το "story". Όταν θελήσουμε να δημιουργήσουμε μια σελίδα (page) ή μια ιστορία (story), πηγαίνουμε από το διαχειριστικό περιβάλλον στην επιλογή δημιουργία περιεχομένου (create content) και συμπληρώνουμε κάποια πεδία του συγκεκριμένου τύπου περιεχομένου όπως τον τίτλο, το κυρίως σώμα, αν επιτρέπονται τα σχόλια, τον συγγραφέα και αν θα δημοσιοποιηθεί ή αν θα βρίσκεται στην αρχική σελίδα.

Ο κωδικας που έχει μέσα το node.tpl.php είναι :

```
 ?>
 <div id="node-<?php print $node->nid; ?>" class="node<?php if ($sticky) { print ' sticky'; } ?><?php if
(!$status) { print ' node-unpublished'; } ?> clear-block">

 <?php print $picture ?>

 <?php if (!$page): ?>
  <h2><a href="<?php print $node_url ?>" title="<?php print $title ?>"><?php print $title ?></a></h2>
 <?php endif; ?>

  <div class="meta">
  <?php if ($submitted): ?>
   <span class="submitted"><?php print $submitted ?></span>
  <?php endif; ?>

  <?php if ($terms): ?>
   <div class="terms terms-inline"><?php print $terms ?></div>
  <?php endif;?>
  </div>

  <div class="content">
   <?php print $content ?>
  </div>

  <?php print $links; ?>
 </div>
```

15.6 Ταξινόμηση περιεχομένου

Για να ταξινομηθεί κάθε αντικείμενο κάθε μιας οντότητας χρειάζεται να δημιουργηθεί ένας πίνακας που θα συσχετίζει τους κόμβους με τους όρους ταξινόμησης. Αυτό μπορεί να γίνει εύκολα με το **module taxonomy** το οποίο δημιουργεί αυτόν τον πίνακα **("term_node" table**) και μας δίνει πολλές ακόμα επιλογές μέσα από το διαχειριστικό περιβάλλον. Δημιουργώντας τον πίνακα "vocabulary" μας δίνει τη δυνατότητα να δημιουργήσουμε κατηγορίες (vocabulary) στις οποίες θα ανήκουν οι όροι, με τον πίνακα "vocabulary_node_types" να αντιστοιχίσουμε τις κατηγορίες σε συγκεκριμένους τύπους περιεχομένου και με τον πίνακα "term_data" να προσθέσουμε όρους στην κάθε κατηγορία. Στο διαχειριστικό περιβάλλον στην υποβολή και στην επεξεργασία ενός κόμβου υπάρχει αυτό το επιπλέον στοιχείο που μας δίνει τη δυνατότητα να κατατάξουμε κάθε κόμβο σε έναν όρο (term). Μια ταξινόμηση για τα νέα- ανακοινώσεις με εκπτώσεις που θα κάνει το μαγαζί το σαββατοκύριακο στο οποίο ανήκει και μια ταξινόμηση για τα videos με όρους «εκπτώσεις» . Με τον τρόπο αυτό όταν προσθέτουμε περιεχόμενο το οποίο είναι ταξινομημένο, όπως μια νέα "ανακοίνωση" έχουμε τη δυνατότητα να το εντάξουμε σε κάποια κατηγορία. Το μεγάλο πλεονέκτημα του Drupal είναι, όπως έχουμε πει, η ευκολία στην επέκταση των δυνατοτήτων και λειτουργιών του site και αυτό

αντικατοπτρίζεται και στην ταξινόμηση, καθώς μπορούν πολύ εύκολα να προστεθούν νέοι όροι σε κάθε κατηγορία. Επίσης πολύ εύκολα μπορούν να προστεθούν και ολόκληρες ταξινομήσεις (vocabularies) εκτός από όρους, όπως για παράδειγμα μια κατηγοριοποίηση των έργων σε Projects. Κάθε όρος έχει ένα id (term id) ανάλογα με τη σειρά με την οποία δημιουργήθηκε, ενώ με το url : ../taxonomy/term/term- id βλέπουμε μια λίστα με όλους τους κόμβους που ανήκουν στο συγκεκριμένο όρο με αυτό το term id. Ένα module που επεκτείνει τις δυνατότητες του "taxonomy" είναι το "taxonomy menu" το οποίο δημιουργεί menu, με τους όρους της ταξινόμησης που έχουμε κάνει, αυτόματα. Το μεγάλο πλεονέκτημα της προσέγγισης αυτής είναι ότι αντί να δημιουργούμε μενού για κάθε ταξινόμηση χειροκίνητα αυτό γίνεται αυτόματα. Έτσι, όταν δημιουργήσουμε για παράδειγμα ένα νέο όρο, όπως για παράδειγμα ένα νέο τύπο ανακοινώσεων (πχ ρεμπέτικη μουσική το Σάββατο) το στοιχείο θα προστεθεί αυτόματα στο μενού προσφέροντας μας ακόμα μεγαλύτερη ευκολία και ελευθερία επέκτασης του υλοποιημένου συστήματος.

15.7 Δημιουργία σύνθετων σελίδων

Η δημιουργία, όμως, ενός κόμβου ενός συγκεκριμένου τύπου περιεχομένου έχει ως αποτέλεσμα να εμφανίζεται μόνο αυτός ο συγκεκριμένος κόμβος όταν γίνει request στον Browser με το path το node/nid του κόμβου. Στις απαιτήσεις, όμως, της ιστοσελίδας μας είναι να εμφανίζονται λίστες ανά κατηγορία δηλαδή μια σύνθετη εμφάνιση στοιχείων πολλών κόμβων φιλτραρισμένα βάσει κάποιων παραμέτρων, όπως για παράδειγμα η λίστα της επικοινωνίας Για να επιτευχθεί αυτό θα πρέπει να γίνει query στη βάση που θα αντλεί δεδομένα από ένα συνδυασμό πινάκων με κάποιες συνθήκες.

Ένα module το οποίο διευκολύνει τη δημιουργία ερωτημάτων στη βάση δεδομένων είναι το "views" το οποίο δίνει τη δυνατότητα με επιλογές μέσα από το διαχειριστικό περιβάλλον να συνθέτονται queries. Η σύνθεση του ερωτήματος (query), που περιγράφηκε προηγουμένως για τη λίστα των εκπτώσεων, μπορεί να υλοποιηθεί μέσα από το διαχειριστικό περιβάλλον με μια view. Οι ρυθμίσεις που έχουμε τη δυνατότητα να κάνουμε αφορούν τόσο τη δημιουργία του query στη βάση, όσο και στον τρόπο εμφάνισης των αποτελεσμάτων. Όσον αφορά τη σύνθεση του ερωτήματος, στο "fields" επιλέγουμε (select) τα πεδία που θέλουμε να εκπτωσεις" και τα ταξινομούμε κατά ημερομηνία σειρά στο sort criteria (order by). Οι επιλογές για τον τρόπο εμφάνισης του αποτελέσματος του ερωτήματος, αφορούν την απεικόνιση σε πίνακα, σε σειρές ή χωρίς διαμόρφωση, το πόσα αποτελέσματα θα παρουσιάζονται σε κάθε σελίδα και αν θα υπάρχει pager για αλλαγή σελίδας, αν θα χρησιμοποιηθεί τεχνική Ajax για την εναλλαγή από τη μία σελίδα στην επόμενη ή θα ανανεώνεται ολόκληρη η σελίδα κτλ. Για κάθε View έχουμε τη δυνατότητα να προσθέτουμε παρουσίαση σαν σελίδα (page display), σαν block (block display) σαν rss feed κτλ. Αν και η προσέγγιση αυτή εμφανίζει τα επιθυμητά αποτελέσματα μας δεσμεύει όσον αφορά την επεκτασιμότητα που χρειαζόμαστε και τη δυνατότητα εύκολης και γρήγορης πρόσθεσης νέων όρων στις ταξινομήσεις. Πιο συγκεκριμένα, αν προσθέταμε στο μέλλον μια νέα κατηγορία προσωπικού π.χ. «επικοινωνία » θα έπρεπε να δημιουργηθεί ένα νέο page display αυτής της view με φίλτρο τον νέο αυτό όρο, όπως κάναμε και προηγουμένως. Για να ξεπεράσουμε αυτό το εμπόδιο και να αυτοματοποιηθεί όσο γίνεται περισσότερο η διαδικασία έπρεπε να έχουμε μια ενιαία view για όλους τους όρους μιας κατηγορίας και το φίλτρο να είναι δυναμικό.

Στο Drupal αυτό επιτυγχάνεται σχετικά εύκολα μέσα από το διαχειριστικό περιβάλλον των views με την επιλογή των arguments η οποία κάνει αυτό ακριβώς που χρειαστήκαμε, ορίζοντας ως argument το term του κάθε όρου, το οποίο όπως είδαμε νωρίτερα είναι μοναδικό και περνώντας το argument μέσα από το url . Έτσι στο url : « .../taxonomy/term/1» θα εμφανιστούν όλοι οι κόμβοι που έχουμε επιλέξει κατά τη δημιουργία τους ότι ανήκουν στον όρο "επικοινωνίας", αφού το 1 περνάει σαν όρισμα και φιλτράρει δυναμικά το αποτέλεσμα.

Το Taxonomy module με την εγκατάστασή του δημιουργεί μια ενιαία view για όλους τους όρους όταν το request είναι της μορφής /taxonomy/term/tid, όπου tid ο μοναδικός αριθμός του κάθε όρου, οπότε και μια ενιαία εμφάνιση για όλους τους όρους γενικώς όλων των κατηγοριών (vocabularies).

Μπορούμε αλλάζοντας την View αυτή να αλλάξουμε τον τρόπο παρουσίασης των όρων. Όμως, εμείς χρειαζόμαστε άλλο τρόπο παρουσίασης των ανακοινώσεων (π.χ σε πίνακα με τίτλο και ημερομηνία) Τη δυνατότητα να ορίσουμε ξεχωριστές views σε κάθε όρο (ή γενικότερα σε ένα συγκεκριμένο vocabulary) μας τη δίνει το module "TVI" (Taxonomy Views Integration). Με τον τρόπο αυτό μπορούμε να δημιουργούμε ξεχωριστές view και να τις αντιστοιχίζουμε σε όρους ή κατηγορίες.

15.8 Σύστημα Μενού

Ορίζει τα μενού πλοήγησης, και τα τις αιτήσεις των διαδρομών των σελίδων σε κώδικα με βάση τα URL.Το σύστημα του μενού του Drupal οδηγεί το σύστημα πλοήγησης από την οπτική του χρήστη και το σύστημα επανάκλησης το οποίο χρησιμοποιεί το Drupal για τις αποκρίσεις στα URL που περάστηκαν από τον περιηγητή.Το σύστημα του μενού του Drupal ακολουθεί μία απλή ιεραρχία η οποία ορίζεται από paths. Οι εφαρμογές του hook_menu() ορίζουν τα αντικείμενα του μενού και τα αναθέτουν σε paths. Το σύστημα μενού συγκεντρώνει αυτά τα αντικείμενα και καθορίζει την ιεραρχία των μενού από τα paths. Όταν ανταποκρίνεται σε ένα αίτημα σελίδας, το σύστημα του μενού ψάχνει να βρει εάν το path που ζητήθηκε από τον περιηγητή έχει καταχωρηθεί ως στοιχείο μενού με επανάκληση. Εάν όχι, το σύστημα ψάχνει το δέντρο του μενού για πιο ολοκληρωμένη ταύτιση που μπορεί να βρει με μία επανάκληση.Η ευρεθείσα επανακαλούμενη function καλείται με οποιαδήποτε arguments έχουν οριστεί στο χαρακτηριστικό "page arguments" του αντικειμένου του. Το χαρακτηριστικό θα πρέπει να είναι ένα array. Μετά από αυτά τα arguments, κάθε εναπομείναντα στοιχεία του path προσαρτώνται ως επιπλέον arguments.

Η πρόσβαση στα functions επανάκλησης προστατεύονται επίσης από το σύστημα του μενού. Η "access callback" μαζί με ένα προαιρετικό "access argument" του κάθε αντικειμένου του μενού, καλείται πριν η επανάκληση της σελίδας προχωρήσει. Εάν επιστρέψει TRUE, τότε επιτρέπεται η πρόσβαση, ενώ εάν επιστρέψει FALSE, όχι.

Λειτουργίες και Μέθοδοι

Όνομα	Περιγραφή
drupal_help_arg:	Παράγει στοιχεία για το $arg array στο hook της βοήθειας.
menu_cache_clear:	Καθαρίζει τα αποθηκευμένα δεδομένα για ένα ονομασμένο μενού.
menu_cache_clear_all:	Καθαρίζει όλα τα αποθηκευμένα δεδομένα μενού.
menu_execute_active_handler:	Εκτελεί την επανάκληση σελίδας η οποία συνδέεται με το τρέχον path.
menu_get_active_breadcrumb:	Παίρνει τα υπολείμματα για την τρέχουσα σελίδα, όπως ορίζεται από το ενεργό ίχνος.
menu_get_active_help:	Επιστρέφει τη βοήθεια που συνδέεται με το ενεργό στοιχείο του μενού.
menu_get_active_menu_name:	Παίρνει το ενεργό μενού για την τρέχουσα σελίδα – ορίζει το ενεργό ίχνος.
menu_get_active_title:	Παίρνει τον τίτλο της τρέχουσας σελίδας, όπως έχει οριστεί από το ενεργό ίχνος.
menu_get_active_trail:	Παίρνει το ενεργό ίχνος της τρέχουσας σελίδας.
menu_get_ancestors:	Επιστρέφει τα ancestors για κάθε δεδομένο path.
menu_get_item:	Παίρνει ένα στοιχείο δρομολογητή.
menu_get_names:	Δημιουργεί μία λίστα από ονομασμένα μενού.
menu_get_object:	Παίρνει ένα φορτωμένο αντικείμενο από ένα στοιχείο δρομολογητή.
menu_link_children_relative_depth:	Βρίσκει το βάθος του “παιδιού” ενός αντικειμένου το οποίο συσχετίζεται με το βάθος αυτού.
menu_link_maintain:	Εισάγει, ενημερώνει ή διαγράφει έναν μη προσαρμοσμένο σύνδεσμο μενού που συσχετίζεται με ένα module.
menu_link_save:	Αποθηκεύει έναν σύνδεσμο μενού.
menu_list_system_menus:	Επιστρέφει ένα array το οποίο περιέχει τα ονόματα των μενού που έχουν οριστεί από το σύστημα.
menu_local_tasks:	Συλλέγει τις τοπικές διεργασίες (καρτέλες), τους συνδέσμους ενεργειών, και το root path.

menu_navigation_links:	Επιστρέφει ένα array από συνδέσμους για ένα μενού πλοήγησης.
menu_path_is_external:	Επιστρέφει TRUE εάν ο σύνδεσμος είναι εξωτερικός.
menu_primary_links:	Επιστρέφει ένα array από συνδέσμους για να καθιστούν ως Πρωτεύοντες σύνδεσμοι.
menu_primary_local_tasks:	Επιστρέφει τις τοπικές διεργασίες που έχουν αποδοθεί στο κορυφαίο επίπεδο.
menu_rebuild:	(Επανα) συμπληρώνει τους πίνακες της βάσης δεδομένων οι οποίοι χρησιμοποιούνται από διάφορα functions του μενού.
menu_router_build:	Συλλέγει και μεταβάλλει τους ορισμούς του μενού.
menu_secondary_links:	Επιστρέφει ένα array από συνδέσμους για να καθιστούν ως Δευτερεύοντες σύνδεσμοι.
menu_secondary_local_tasks:	Επιστρέφει τις τοπικές διεργασίες που έχουν αποδοθεί στο δεύτερο επίπεδο.
menu_secondary_menu:	Επιστρέφει ένα array από συνδέσμους για να αποδοθούν ως δευτερεύοντες σύνδεσμοι.
menu_set_active_item:	Θέτει το ενεργό path, το οποίο ορίζει το ποια σελίδα θα φορτώσει.
menu_set_active_menu_name:	Θέτει (ή παίρνει) το ενεργό μενού για την τρέχουσα σελίδα – ορίζει το ενεργό ίχνος.
menu_set_active_trail:	Θέτει το ενεργό ίχνος της τρέχουσας σελίδας.
menu_set_item:	Αντικαθιστά το στοιχείο που έχει αποθηκευτεί στατικά για ένα συγκεκριμένο path.
menu_tab_root_path:	Επιστρέφει το path του δρομολογητή, ή το path της καρτέλας γονέα μίας προεπιλεγμένης τοπικής εργασίας.
menu_tail_to_arg:	Επιστρέφει το path σαν ένα string από το argument που βρισκόμαστε.
menu_tree:	Αποδίδει ένα δέντρο μενού το οποίο βασίζεται στο τρέχον path.
menu_tree_all_data:	Παίρνει τη δομή δεδομένων η οποία παρουσιάζει ένα ονομασμένο δέντρο μενού.
menu_tree_check_access:	Ελέγχει την πρόσβαση και διενεργεί άλλες δυναμικές λειτουργίες για κάθε σύνδεσμο στο δέντρο.
menu_tree_collect_node_links:	Αναδρομική βοηθητική function – συλλέγει συνδέσμους κόμβων.
menu_tree_data:	Δημιουργεί τα δεδομένα που παρουσιάζουν ένα δέντρο μενού.
menu_tree_output:	Επιστρέφει ένα αποδοσμένο δέντρο μενού.
menu_tree_page_data:	Παίρνει τη δομή δεδομένων που παρουσιάζει ένα ονομασμένο δέντρο μενού, βασισμένο στην τρέχουσα σελίδα.
menu_unserialize:	Αποσειριοποιεί δεδομένα μενού, κάνοντας χρήση ενός χάρτη για να αντικαταστήσει τα στοιχεία του path.
menu_valid_path:	Επικυρώνει το path ενός συνδέσμου μενού ο οποίος δημιουργείται ή επεξεργάζεται.
theme_menu_item:	Επιστρέφει HTML για ένα στοιχείο μενού και υπο-

	μενού.
theme_menu_item_link:	Επιστρέφει HTML για έναν σύνδεσμο μενού.
theme_menu_local_task:	Επιστρέφει HTML γία έναν σύνδεσμο τοπικής εργασίας.
theme_menu_local_tasks:	Επιστρέφει HTML για πρωτεύουσες και δευτερεύουσες τοπικές διεργασίες.
theme_menu_tree:	Επιστρέφει HTML για ένα wrapper για υπό-δεντρο μενού.
menu_check_access:	Ελέγχει την πρόσβαση σε ένα στοιχείο μενού χρησιμοποιώντας επανάκληση πρόσβασης.
menu_clear_page_cache:	Βοηθητική function για να καθαρίσει τα αποθηκευμένα δεδομένα των σελίδων και των μπλοκ το περισσότερο δύο φορές ανά φόρτωση σελίδας.
menu_delete_item:	Βοηθητική function για for menu_link_delete. Διαγράφει έναν σύνδεσμο μενού.
menu_find_router_path:	Βρίσκει το path του δρομολογητή που θα εξυπηρετήσει αυτό το path.
menu_item_localize:	Εντοπίζει τον τίτλο του αντικειμένου του δρομολογητή χρησιμοποιώντας το t() ή άλλη επανάκληση.
menu_link_build:	Δημιουργεί έναν σύνδεσμο από ένα στοιχείο δρομολογητή.
menu_link_map_translate:	Αυτό το function μεταφράζει τα στοιχεία path στον χάρτη χρησιμοποιώντας οποιοδήποτε βοηθητικό function to_arg. Αυτά τα functions παίρνουν ένα argument κι επιστρέφουν ένα object.
menu_link_move_children:	Ενημερώνει το παιδί ενός συνδέσμου μενού ότι μετακινείται.
menu_link_parents_set:	Βοηθητικό function που θέτει τις τιμές p1..p9 για έναν σύνδεσμο μενού που αποθηκεύεται.
menu_link_translate:	Αυτό το function είναι παρόμοιο με το _menu_translate() αλλά κάνει προετοιμασία συγκεκριμένου συνδέσμου όπως του να καλεί συνέχεια to_arg functions.
menu_load_objects:	Φορτώνει αντικείμενα στον χάρτη όπως έχουν οριστεί στο $item['load_functions'].
menu_navigation_links_rebuild:	Βοηθητικό function το οποίο δημιουργεί συνδέσμους για τα αντικείμενα στον δρομολογητή μενού.
menu_router_build:	Βοηθητικό function το οποίο δημιουργεί τον πίνακα του δρομολογητή βασισμένο στα δεδομένα του hook_menu.
menu_router_cache:	Βοηθητικό function το οποίο αποθηκεύει το δρομολογητή του μενού εάν υπάρχει στη μνήμη.
menu_set_expanded_menus:	Βοηθητικό function το οποίο ενημερώνει μία λίστα από μενού με διευρυμένα αντικείμενα.
menu_site_is_offline:	Ελέγχει εάν το site βρίσκεται σε κατάσταση συντήρησης.
menu_translate:	Διαχειρίζεται μεταφράσεις δυναμικού path και έλεγχο πρόσβασης μενού.
menu_tree_check_access:	Αναδρομικό βοηθητικό function για το

	menu_tree_check_access()
menu_tree_cid:	Βοηθητικό function για το menu_tree_check_access().
menu_tree_data:	Αναδρομικό βοηθητικό function το οποίο δημιουργεί τα δεδομένα που αντιπροσωπεύουν το δέντρο μενού.
menu_update_parental_status:	Ελέγχει και ενημερώνει την κατάσταση του has_children για τον γονέα του συνδέσμου.

Constants

Όνομα	Περιγραφή
MENU_ACCESS_DENIED:	Εσωτερικός κώδικας κατάστασης μενού – Απαγορεύεται η πρόσβαση στο στοιχείο μενού.
MENU_CALLBACK:	Τύπος μενού – Μία κρυφή, εσωτερική επανάκληση, η οποία συνήθως χρησιμοποιείται για κλήσεις API.
MENU_CREATED_BY_ADMIN:	Εσωτερική flag μενού – το στοιχείο του μενού δημιουργήθηκε από τον διαχειριστή.
MENU_DEFAULT_LOCAL_TASK:	Τύπος μενού – Η προεπιλεγμένη τοπική εργασία, που είναι ενεργή από την αρχή.
MENU_FOUND:	Εσωτερικός κώδικας κατάστασης μενού – Βρέθηκε το στοιχείο του μενού.
MENU_IS_LOCAL_TASK:	Εσωτερική flag μενού – το στοιχείο μενού είναι μία τοπική εργασία.
MENU_IS_ROOT:	Εσωτερική flag μενού – το στοιχείο μενού είναι η ρίζα του δέντρου μενού.
MENU_LINKS_TO_PARENT:	Εσωτερική flag μενού – το στοιχείο μενού δείχνει πίσω στον γονέα του.
MENU_LOCAL_TASK:	Τύπος μενού – Μία εργασία για το στοιχείο γονέα.
MENU_MAX_DEPTH:	Το μέγιστο βάθος ενός δέντρου συνδέσμων μενού - Ταιριάζει τον αριθμό των p στηλών.
MENU_MAX_PARTS:	Ο μέγιστος αριθμός από στοιχεία path για μία επανάκληση μενού.
MENU_MODIFIED_BY_ADMIN:	Εσωτερική flag μενού – Το στοιχείο μενού μπορεί να τροποποιηθεί από τον διαχειριστή.
MENU_NORMAL_ITEM:	Τύπος μενού – Ένα κανονικό στοιχείο μενού το οποίο εμφανίζεται στο μενού και στα υπολείμματα.
MENU_NOT_FOUND:	Εσωτερικός κώδικας κατάστασης μενού –Το στοιχείο του μενού δεν βρέθηκε.
MENU_SITE_OFFLINE:	Εσωτερικός κώδικας κατάστασης μενού – Το στοιχείο του μενού είναι μη προσβάσιμο διότι το site βρίσκεται εκτός σύνδεσης.
MENU_SUGGESTED_ITEM:	Τύπος μενού – Ένα κανονικό στοιχείο μενού, κρυφό μέχρι να ενεργοποιηθεί από τον διαχειριστή.
MENU_VISIBLE_IN_BREADCRUMB:	Εσωτερική flag μενού – Το στοιχείο του μενού είναι ορατό στα υπολείμματα.
MENU_VISIBLE_IN_TREE:	Εσωτερική flag μενού – Το στοιχείο του μενού είναι ορατό στο δέντρο του μενού.

Groups

Name	Description
Menu flags:	Flags για χρήστη στο χαρακτηριστικό "τύπος" των στοιχείων μενού.
Menu item types:	Ορισμοί για διάφορους τύπους στοιχείων μενού.
Menu status codes:	Κωδικοί κατάστασης για τις επανακλήσεις των μενού.
Menu tree parameters:	Παράμετροι για ένα δέντρο μενού.

15.9 Blocks

Τα blocks είναι διακριτά τμήματα των πληροφοριών που εμφανίζονται στις περιοχές των σελίδων του δικτυακού σας τόπου. Τα blocks μπορούν να λάβουν τη μορφή μενού (που ασχολούνται με την πλοήγηση), εξόδου από κάποιο ή δυναμικά και στατικά κομμάτια πληροφοριών που έχετε δημιουργήσει μόνοι σας (για παράδειγμα, μια λίστα με προσεχείς εκδηλώσεις) .Για να εισάγουμε τα block πηγαίνουμε στην υποενότητα "Μπλοκ" της ενότητας "Δημιουργία Ιστοτόπου" και μας παρουσιάζεται μια σελίδα, όπου βλέπουμε τις περιοχές (regions) του συγκεκριμένου θεματικού προτύπου και τα διαθέσιμα block με δυνατότητα τοποθέτησής τους σε κάποια περιοχή. Τα block αυτά δε θέλουμε να εμφανίζονται σε όλες τις σελίδες. Όπως είπαμε πριν, το κάθε υπομενού θα πρέπει να εμφανίζεται όταν έχει επιλεγεί η αντίστοιχη ενότητα, ενώ τα άλλα δε θα πρέπει να εμφανίζονται. Για να επιτευχθεί αυτό πηγαίνουμε στη ρύθμιση του αντίστοιχου block και στην καρτέλα "Page specific visibility settings" μπορούμε να ρυθμίσουμε σε ποιες σελίδες θα εμφανίζεται το Block με βάση το url της σελίδας. Ακόμα έχουμε τη δυνατότητα να γράψουμε κώδικα σε PHP και όταν επιστρέφει true να εμφανίζεται το block.

15.10 Διευθύνσεις URL

Οι διευθύνσεις που προκαθορισμένα δημιουργεί το Drupal είναι της μορφής www.example.com/?q=node/67, η οποία μορφή δεν είναι φιλική ούτε προς τους ανθρώπους ούτε προς τις μηχανές αναζήτησης. Με τη δυνατότητα του Drupal για clean urls κι αν ο web Server το υποστηρίζει, η διεύθυνση αυτή μπορεί να πάρει τη μορφή www.example.com/node/67. Όμως και πάλι δεν είναι user friendly και δε δίνουν τόσο καλά αποτελέσματα στις μηχανές αναζήτησης. Για να αντιμετωπισθούν αυτά τα ζητήματα και οι διευθύνσεις να γίνουν της μορφής «www.example.com/Αρχική σελιδα» χρησιμοποιούμε το core **module "Path"** με το οποίο μπορούμε να δημιουργήσουμε ψευδώνυμα url. Αν και η προσέγγιση αυτή είναι πολύ χρήσιμη, είναι επίπονη και δύσκολα κατανοητή από τον διαχειριστή του site, αφού κάθε φορά θα πρέπει να προσθέτει μαζί με τα απαραίτητα στοιχεία κάθε κόμβου και ένα εναλλακτικό url. Ένα πολύ χρήσιμο Module το οποίο δημιουργεί αυτόματα ψευδώνυμα url ανάλογα με τις ρυθμίσεις μας (το όνομα του περιεχομένου, τον τύπο περιεχομένου, την κατηγορία στην οποία ανήκει κτλ) είναι το "path auto".

Και ο κωδιακς που αποτελειτε το path.module είναι :

```
<?php
// $Id: path.module,v 1.138.2.3
* Implementation of hook_help().
 */
function path_help($path, $arg) {
  switch ($path) {
    case 'admin/help#path':
      $output = '<p>'. t('The path module allows you to specify aliases for Drupal URLs. Such aliases improve readability of URLs for your users and may help internet search engines to index your content more effectively. More than one alias may be created for a given page.') .'</p>';
      $output .= t('<p>Some examples of URL aliases are:</p>
<ul>
<li>user/login =&gt; login</li>
<li>image/tid/16 =&gt; store</li>
<li>taxonomy/term/7+19+20+21 =&gt; store/products/whirlygigs</li>
<li>node/3 =&gt; contact</li>
</ul>
');
///…
      return '<p>'. t('Enter the path you wish to create the alias for, followed by the name of the new alias.') .'</p>';
  }
}

/**
 * Implementation of hook_menu().
 */
function path_menu() {
  $items['admin/build/path'] = array(
    'title' => 'URL aliases',
    'description' => "Change your site's URL paths by aliasing them.",
    'page callback' => 'path_admin_overview',
    'access arguments' => array('administer url aliases'),
    'file' => 'path.admin.inc',
  );
  $items['admin/build/path/edit'] = array(
    'title' => 'Edit alias',
    'page callback' => 'path_admin_edit',
    'access arguments' => array('administer url aliases'),
    'type' => MENU_CALLBACK,
    'file' => 'path.admin.inc',
  );
  $items['admin/build/path/delete'] = array(
    'title' => 'Delete alias',
    'page callback' => 'drupal_get_form',
    'page arguments' => array('path_admin_delete_confirm'),
    'access arguments' => array('administer url aliases'),
```

```
    'type' => MENU_CALLBACK,
    'file' => 'path.admin.inc',
   );
   $items['admin/build/path/list'] = array(
    'title' => 'List',
    'type' => MENU_DEFAULT_LOCAL_TASK,
    'weight' => -10,
   );
   $items['admin/build/path/add'] = array(
    'title' => 'Add alias',
    'page callback' => 'path_admin_edit',
    'access arguments' => array('administer url aliases'),
    'type' => MENU_LOCAL_TASK,
    'file' => 'path.admin.inc',
   );

   return $items;
  }

  /**
   * Post-confirmation; delete an URL alias.
   */
  function path_admin_delete($pid = 0) {
   db_query('DELETE FROM {url_alias} WHERE pid = %d', $pid);
   drupal_set_message(t('The alias has been deleted.'));
  }

  /**
   * Set an aliased path for a given Drupal path, preventing duplicates.
   */
  function path_set_alias($path = NULL, $alias = NULL, $pid = NULL, $language = '') {
   $path = urldecode($path);
   $alias = urldecode($alias);
   // First we check if we deal with an existing alias and delete or modify it based on pid.
   if ($pid) {
    // An existing alias.
    if (!$path || !$alias) {
     // Delete the alias based on pid.
     db_query('DELETE FROM {url_alias} WHERE pid = %d', $pid);
    }
    else {
     // Update the existing alias.
     db_query("UPDATE {url_alias} SET src = '%s', dst = '%s', language = '%s' WHERE pid = %d",
$path, $alias, $language, $pid);
    }
   }
   else if ($path && $alias) {
    // Check for existing aliases.
```

```
      if ($alias == drupal_get_path_alias($path, $language)) {
        // There is already such an alias, neutral or in this language.
        // Update the alias based on alias; setting the language if not yet done.
        db_query("UPDATE {url_alias} SET src = '%s', dst = '%s', language = '%s' WHERE dst =
'%s'", $path, $alias, $language, $alias);
      }
      else {
        // A new alias. Add it to the database.
        db_query("INSERT INTO {url_alias} (src, dst, language) VALUES ('%s', '%s', '%s')", $path,
$alias, $language);
      }
    }
    else {
      // Delete the alias.
      if ($alias) {
        db_query("DELETE FROM {url_alias} WHERE dst = '%s'", $alias);
      }
      else {
        db_query("DELETE FROM {url_alias} WHERE src = '%s'", $path);
      }
    }
    drupal_clear_path_cache();
  }


  /**
   * Implementation of hook_nodeapi().
   *
   * Allows URL aliases for nodes to be specified at node edit time rather
   * than through the administrative interface.
   */
  function path_nodeapi(&$node, $op, $arg) {
    // Permissions are required for everything except node loading.
    if (user_access('create url aliases') || user_access('administer url aliases') || ($op == 'load')) {
      $language = isset($node->language) ? $node->language : '';
      switch ($op) {
        case 'validate':
          if (isset($node->path)) {
            $node->path = trim($node->path);
            if (db_result(db_query("SELECT COUNT(dst) FROM {url_alias} WHERE dst = '%s' AND
src != '%s' AND language = '%s'", $node->path, "node/$node->nid", $language))) {
              form_set_error('path', t('The path is already in use.'));
            }
          }
          break;

        case 'load':
          $path = 'node/'. $node->nid;
```

```
        $alias = drupal_get_path_alias($path, $language);
        if ($path != $alias) {
          $node->path = $alias;
        }
        break;

      case 'insert':
        // Don't try to insert if path is NULL. We may have already set
        // the alias ahead of time.
        if (isset($node->path)) {
          path_set_alias('node/'. $node->nid, $node->path, NULL, $language);
        }
        break;

      case 'update':
        path_set_alias('node/'. $node->nid, isset($node->path) ? $node->path : NULL, isset($node->pid) ? $node->pid : NULL, $language);
        break;

      case 'delete':
        $path = 'node/'. $node->nid;
        if (drupal_get_path_alias($path) != $path) {
          path_set_alias($path);
        }
        break;
    }
  }
}

/**
 * Implementation of hook_form_alter().
 */
function path_form_alter(&$form, $form_state, $form_id) {
  if (isset($form['type']) && isset($form['#node']) && $form['type']['#value'] .'_node_form' == $form_id) {
    $path = isset($form['#node']->path) ? $form['#node']->path : NULL;
    $form['path'] = array(
      '#type' => 'fieldset',
      '#title' => t('URL path settings'),
      '#collapsible' => TRUE,
      '#collapsed' => empty($path),
      '#access' => user_access('create url aliases'),
      '#weight' => 30,
    );
    $form['path']['path'] = array(
      '#type' => 'textfield',
      '#default_value' => $path,
      '#maxlength' => 128,
      '#collapsible' => TRUE,
```

```
        '#collapsed' => TRUE,
        '#description' => t('Optionally specify an alternative URL by which this node can be accessed. For example, type "about" when writing an about page. Use a relative path and don\'t add a trailing slash or the URL alias won\'t work.'),
      );
      if ($path) {
        $form['path']['pid'] = array(
          '#type' => 'value',
          '#value' => db_result(db_query("SELECT pid FROM {url_alias} WHERE dst = '%s' AND language = '%s'", $path, $form['#node']->language))
        );
      }
    }
  }
  * Implementation of hook_perm().
   */
  function path_perm() {
    return array('create url aliases', 'administer url aliases');
  }

  * Fetch a specific URL alias from the database.
   */
  function path_load($pid) {
    return db_fetch_array(db_query('SELECT * FROM {url_alias} WHERE pid = %d', $pid));
  }
```

15.11 Η δομή της ιστοσελίδας μας

Στην αρχική μας σελίδα, στο πάνω μέρος της θα βρούμε το κύριος μενού, που με της κατηγορίες του θα μας βοηθήσει να δημιουργήσουμε μια ολοκληρωμένη εικόνα όσων υπηρεσιών προσφέρονται. Αριστερά στο πάνω μέρος ακριβώς μετά το καλωσόρισμα, βλέπουμε ένα κείμενο της ιστορίας του καταστήματος. Ακολουθούν πληροφορίες για το μουσικό **πρόγραμμα**. Εν συνεχεία έχουμε την παρουσίαση του γαστρονομικού **μενού** και των αναλόγων τιμών. Πριν την ολοκλήρωση της ιστοσελίδας μας παρουσιάσετε ένα **αρχείο** με φωτογραφίες των χώρων του καταστήματος και γίνετε μια αναφορά των σημαντικών δραστηριοτήτων που έχουν λάβει μέρος στο συγκεκριμένο χώρο. Η ιστοσελίδα κλείνει με **Επικοινωνία** ,τηλεφωνικούς αριθμούς, ηλεκτρονική διήθηση (e-mail), ώρες και μέρες λειτουργίες.

15.11 Δημιουργία και επεξεργασία σελίδας

Στη περιοχή Υποβολή ύλης/Σελίδα δημιουργήσαμε τις σελίδες που χρησιμοποιήσαμε στο site μας. Βάζοντας τίτλο μορφότυπο εισόδου και διαδρομή URL που επιθυμούσαμε. Στη συνέχεια με τη χρήση του FCKeditor και των εργαλείων του κάναμε εισαγωγή των κειμένων και τον εικόνων μας. Η επεξεργασία σελίδας γινόταν στη περιοχή Διαχείριση/Περιεχόμενα.

15.12 Δημιουργία μπλοκ

Τα μπλοκ στο Drupal, δημιουργούνται αυτόματα από τις εγκατεστημένες μονάδες (modules).
Είναι μπλοκ πληροφοριών, τα οποία τοποθετούνται σε διάφορες θέσεις της σελίδας μας, οι οποίες εξαρτώνται από τις θεματικές παραλλαγές. Στη περιοχή διαχείριση/Μπλοκ/Προσθήκη μπλοκ δημιουργήσαμε τα μπλοκ όπου χρησιμοποιήσαμε. Βάζοντας περιγραφή του μπλοκ, τίτλος του μπλοκ και το σώμα που θέλουμε να έχει. Μετά από ρυθμίσεις ορατότητας σελίδας ρυθμίσαμε το κάθε μπλοκ σε ποιες σελίδες θα είναι ορατό κ σε ποιες όχι. Μετά από τη περιοχή Διαχείριση/Μπλοκ/Λίστα κατατάξαμε τα μπλοκ και τα τοποθετήσαμε στις περιοχές που επιθυμούσαμε. Υπάρχει επίσης η δυνατότητα ρύθμισης συγκεκριμένων επιλογών σε
αυτή την περιοχή.

Εικόνα 30: Η Αρχική σελίδα του Drupal

15.13 Αρχική σελίδα

Στην Αρχική σελίδα που είναι και οι πρώτη που βλέπει ένας χρήστης μόλις μπει στην ιστοσελίδα μας, βλέπουμε πάνω πάνω Ταβέρνα - Ρεμπετάδικο: 'Καλώς ορίσατε' Ποιο κάτω βλέπουμε μια μικρή εισαγωγή και ενημέρωση για τι θα βρει κάποιος που θα επισκεφθεί το Ρεμπετάδικο.
Οι κατηγορίες είναι οι εξής:
1. Πρόγραμμα
2. Μενού
3. Αρχείο
4. Διοργάνωση εκδηλώσεων
5. Επικοινωνία

16. Πίνακας σύγκρισης του Framework Symfony με το Drupal

Το Symfony καθώς και το Drupal είναι δύο πολύ γνωστά εργαλεία, μηδενικού κόστους, για τον σχεδιασμό και την ανάπτυξη ιστοσελίδων.
Το Drupal είναι ένα Content Management System ή απλά CMS, το οποίο παρέχει έτοιμα script που επιτρέπουν στον χρήστη να δημιουργήσει εύκολα και γρήγορα ένα site. Για να χρησιμοποιήσει κανείς κάποιο CMS, δεν απαιτείται η γνώση κάποιας γλώσσας (php, HTML κλπ), μιας και είναι έτοιμο για να τρέξει, σε αντίθεση με το Symfony, το οποίο είναι ένα PHP 5 Framework, το οποίο παρέχει μια αρχιτεκτονική, στοιχεία και εργαλεία στους προγραμματιστές για την ανάπτυξη σύνθετων εφαρμογών διαδικτύου γρήγορα.

Λεπτομέρειες:		
Όνομα Προϊόντος	**Drupal**	**Symfony**
Εταιρεία / Προγραμματιστές	Dries Buytaert	Symfony Team
Τελευταίες Σταθερές Εκδόσεις	Drupal 7.1	Symfony 2.2.5
Server Εφαρμογής	Apache / IIS	Apache
Κόστος	0€	0€
Λειτουργικό Σύστημα	Ανεξαρτήτων Πλατφορμών	Ανεξαρτήτων Πλατφορμών
Άδεια Χρήσης	GPL	MIT
Γλώσσα Προγραμματισμού	PHP	PHP
Βάση Δεδομένων	MySQL / PosgreSQL / SQLite	MySQL
Ιστοσελίδα	drupal.org	www.symfony-project.org/
Χαρακτηριστικά		
Root Πρόσβαση	Όχι	Ναι
Shell Πρόσβαση	Όχι	Όχι
Τύπος Πιστοποίησης	Ναι, Με Δυνατότητα Σύνδεσης	Ναι, Με Δυνατότητα Σύνδεσης
Page Caching	Ναι	Ναι
Αναπαραγωγή Βάσης Δεδομένων	Περιορισμένη	Περιορισμένη
Υποστήριξη FTP	Περιορισμένη	Περιορισμένη

Συμβατό με XHTML	Ναι	Ναι
Υποστήριξη UTF-8	Ναι	Ναι
Αναφορές Βάσης Δεδομένων	Ναι	Δωρεάν Πρόσθετο
Metadata	Ναι	Ναι
Φιλικά URL	Ναι	Ναι
Επανεγγραφή URL	Ναι	Ναι
Άλλα Χαρακτηριστικά		
Εγγριση Περιεχομένου	Ναι	Δωρεάν Πρόσθετο
Ιστορικό Εκδόσεων	Ναι	Δωρεάν Πρόσθετο
Εμπορική Υποστήριξη / Εκπαίδευση	Ναι	Ναι
Μεταφορά και Απόθεση Περιεχομένου	Δωρεάν Πρόσθετο	Όχι
Συνδρομές	Δωρεάν Πρόσθετο	Όχι
Εξισορρόπηση Φορτίου	Ναι	Όχι
Στατιστικά Web	Ναι	Όχι
Υπηρεσίες Web Προσκηνίου	Περιορισμένες	Περιορισμένες
RSS	Ναι	Ναι

ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] www.symfony-project.org/

[2] http://symfony.com/about

[3] www.symfony-project.org/jobeet/1_2/Propel/en/

[4] www.youtube.com/ symphony

[5] http://en.wikipedia.org/wiki/Symfony

[6] Drupal - http://drupal.org/

[7] Drupal Gr - http://mydrupal.gr/

[8] Drupal - http://www.drupalove.com/

[9] Edutorials - http://edutorials.gr/node/216

[10] Wikipedia Gr - http://el.wikipedia.org

[11] Youtube - http://www.youtube.com/

[12] Google - http://www.google.gr/

[13]Go-Online - http://www.go-online.gr/ebusiness/specials/article.html?article_id=1042

[14] GNU - http://www.gnu.org/philosophy/free-sw.el.html

[15] WAMP - http://www.wampserver.com/en/presentation.php

[16] Drupal - http://drupal.org/node/

[17] Google - http://www.google.com

[18] Wikipedia - http://www.wikipedia.com

[19] Edutorials - http://www.edutorials.gr/book/export/html