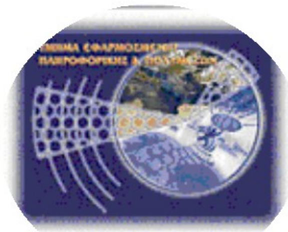
Τεχνολογικό Εκπαιδευτικό Ίδρυμα Κρήτης

Σχόλη Τεχνολογικών Εφαρμογών

Τμήμα Εφαρμοσμένης Πληροφορικής & Πολυμέσων



Πτυχιακή εργασία

Ανάπτυξη Συστήματος Διαδικτυακής Ενημέρωσης και Διαχείρισης Εργασιών

Επιβλέπων καθηγητής : Βασιλάκης Κώστας

Επιτροπή Αξιολόγησης :

Ημερομηνία παρουσίασης:

Επιμέλεια:Ξηρογιάννη Μαριέλενα

Α.Μ. 1340

Ηράκλειο – 2012

Ευχαριστίες

Ευχαριστώ όλους αυτούς τους ανθρώπους που έχουν ασχοληθεί με τη στηρίξει και διαδώσει το Open Source και μπόρεσα να πάρω αρκετή γνώση που ίσος να χρειαζόμουν περισσότερο από μια ζωή να τα γνωρίσω. Τον επιβλέποντα καθηγητή για τη θεματολογία της πτυχιακής και το ελεύθερο που μου έδωσε στην αρχή να επιλέξω ελεύθερα το εργαλείο ανάπτυξης της εφαρμογής αυξάνοντας το ενδιαφέρον μου . Ευχαριστώ ιδιαίτερος το φίλο Γιάννη Τσαγκατάκη για τη βοήθεια του στη Ruby on rails και τους φίλους που με στήριξαν

Abstract

The objective of the thesis is to develop and implement a usefull web application. That provides ability about organization of services and maintainabledamange control on network infrastructure and telematics services. The application developed according to system requirements in trouble ticket. Guided example is the Centre and Network Operations Control (CNOC) the TEI of Crete. To development our application we following methology on framework Ruby on Rails.

Σκοπός

Ο βασικός στόχος της πτυχιακής εργασίας είναι η ανάπτυξη και υλοποίηση μιας εύχρηστης διαδικτυακής εφαρμογής η οποία παρέχει δυνατότητες οργάνωσης για την παροχή υπηρεσιών συντήρησης και αντιμετώπισης βλαβών σε θέματα δικτυακών υποδομών και υπηρεσιών τηλεματικής. Η εφαρμογή αναπτύσσετε με βάση τις προϋποθέσεις σε σύστημα truble ticket με τελική κατευθυντήρια τις ανάγκες του Κέντρου Ελέγχου και Διαχείρισης Δικτύων (ΚΕΔΔ) του ΤΕΙ Κρήτης ακολουθούμενη τη μεθοδολογία ανάπτυξης εφαρμογών Ruby on Rails.

Περιεχόμενα

|Εισαγωγή Βασικές Έννοιες| 1

1.1 ΕΙΣΑΓΩΓΗ 1

1.2 ΤΙ ΕΙΝΑΙ TRUBLE TICKET 2

1.3 ΤΙ ΕΙΝΑΙ RUBY 2

1.4 ΤΙ ΕΙΝΑΙ Η RUBY ON RAILS 4
1.4.1 Αρχιτεκτονική Model – View – Controller (MVC) 5
1.4.2 Χαρακτηριστικά της Ruby on Rails 5

1.5 Η ΠΛΑΤΦΟΡΜΑ ΒΑΣΗΣ ΔΕΔΟΜΕΝΩΝ POSTGRES 6

1.6 ΜΗΧΑΝΗ ΠΕΠΕΡΑΣΜΕΝΩΝ ΚΑΤΑΣΤΑΣΕΩΝ FSM 7

|Λειτουργίες & Εργαλεία Ανάπτυξης| 9

2.1 ΤΑ ΛΕΙΤΟΥΡΓΙΚΑ ΧΑΡΑΚΤΗΡΙΣΤΗΚΑ ΤΗΣ RUBY ON RAILS 9
2.1.1 Η δρομολόγηση REST των αιτήσεων σε εφαρμογή της RoR 9
2.1.2 Η λειτουργία του MVC στη RoR 10

2.2 ΜΕΘΟΔΟΙ ΑΝΑΠΤΥΞΗΣ ΕΦΑΡΜΟΓΩΝ ΣΕ RoR 13
2.2.1 Testing στη RoR 13
2.2.2 Μεθοδολογία Agile ανάπτυξης κώδικα 14
2.2.3 Αποφυγή επαναλήψεων κώδικα DRY 15
2.2.4 Asset Pipeline 15

2.3 ΕΡΓΑΛΕΙΑ ΠΟΥ ΧΡΗΣΙΜΟΠΟΙΗΘΗΚΑΝ 16
2.3.1 Βιβλιοθήκες της RoR 16
2.3.2 Έλεγχος πηγαίου κώδικα με τη χρήση Git 17
2.3.3 Πλατφόρμα εφαρμογών Heroku 19

|Ανάλυση & Υλοποίηση Εφαρμογής| 21

3.1 ΜΕΘΟΔΟΛΟΓΙΑ ΑΝΑΠΤΥΞΗΣ ΛΟΓΙΣΜΙΚΟΥ 21
3.1.1 Βασικές ανάγκες συστήματος 22
3.1.2Δημιουργία πινάκων μέσα από τις απαιτήσεις 22

3.2 ΥΛΟΠΟΙΗΣΗ ΤΟΥ ΕΡΓΟΥ Α' ΣΤΑΔΙΟ 24
3.2.1Εγκατάσταση Ruby on rails 25
3.2.2 Δημιουργία της Εφαρμογής μας 25
3.2.3 Περιβάλλον Εργασίας της Ανάπτυξης 27
3.2.4 Υλοποίηση των αρχικών στόχων 27
3.2.5 Υλοποίηση της συνεδρίας των χρηστών στο σύστημα 30
3.2.6 Υλοποίηση διαχείριση tickets από τους χρήστες 31
3.2.7 Εγκατάσταση σε Server και έλεγχος καλής λειτουργίας 37

3.3 ΥΛΟΠΟΙΗΣΗ ΤΟΥ ΕΡΓΟΥ ΣΕ Β' ΣΤΑΔΙΟ 38
3.3.1 Υυλοποίηση νέων διεργασιών ιστορικού ticket 39
3.3.2 Διαμόρφωση διεπαφής σύμφωνα με τα σενάρια χρήσης 39
3.3.3 Ενημέρωση συστήματος 41

|Θέματα Ασφάλειας| 43

4.1 ΘΕΜΑΤΑ ΑΣΦΑΛΕΙΑΣ ΣΤΗ RUBY ON RAILS 43
4.1.1 Συνεδρίες χρηστών 43

4.1.2Επίθεση συνεδρίας από το μέσο μετάδοσης 43
4.1.3 Επίθεση μέσο συνεδρίας από τον υπολογιστή μας 44
4.1.4 Επιθεση μέσο συνεδρίας σε παράλληλο χρόνο χρήσης. 44
4.1.5 Ανακατεύθυνση ιστοσελίδων και αρχείων 44
4.1.6 Ανακατεύθυνση ιστοσελίδας 45
4.1.5 Ανακατεύθυνση αρχείων 45

4.2 ΕΠΙΘΕΣΕΙΣ ΣΕ ΛΟΓΑΡΙΑΣΜΟΥΣ ΔΙΑΧΕΙΡΗΣΤΗ 45
4.2.1 Ενδοδικτυακή45
4.2.1 Μαζική αποστολή στοιχείων 45

|Άλλα frameworks - Συμπεράσματα| 47

5.1 FRAMEWORK ΚΑΙ Η RAILS 47
5.1.1 Django 47
5.1.2 Symfony 47
5.1.3 Cake php 47

5.2 ΚΡΗΤΗΡΙΑ ΚΑΙ ΠΡΟΫΠΟΘΕΣΕΙΣ ΠΛΑΤΦΟΡΜΩΝ ΑΝΑΠΤΗΞΗΣ 48

5.3 ΣΥΜΠΕΡΑΣΜΑΤΑ 50

5.4 ΜΕΛΛΟΝΤΙΚΗ ΕΡΓΑΣΙΑ ΚΑΙ ΕΠΕΚΤΑΣΕΙΣ 50

References 53

Βασικές ‘Εννοιες 55

Σχήματα 56

|Κώδικας| 60

Ο ΚΩΔΙΚΑΣ ΤΩΝ ΑΡΧΕΙΩΝ 60

|Περίληψη| 84

Πίνακας Εικόνων

Εικόνα 1: Η καθημερινή κίνηση μηνυμάτων για ruby 3
Εικόνα 2 :Διάγραμμα νέων εφαρμογών σε Open Source 3
Εικόνα 3: Χαρακτηριστικά PHP, Python, Ruby 4
Εικόνα 4: Ζήτηση θέσεων εργασίας για Postgres,Mysql,Oracle 7
Εικόνα 5: Πίνακας Στόχων Μεθοδολογίας Agile 22
Εικόνα 6: Περιβάλλον Εργασίας 27
Εικόνα 7: Links Εισόδου Εικόνα 8: Links Εξόδου 31
Εικόνα 9: Εμφάνιση Ιστορικού 36
Εικόνα 10: Γραφικό Περιβάλλον Εκδόσεων Α΄ Στάδιο 38
Εικόνα 11: Περιβάλλον profile 39
Εικόνα 12: Περιβάλλον Εργασίας Workers40
Εικόνα 13: Περιβάλλον Εργασίας Ticket και Admin 40
Εικόνα 14: Γραφική Αναφορά Των Εκδόσεων της Εφαρμογής στο Β΄ Στάδιο 41
Εικόνα 15: Γράφημα ζήτησης θέσεων εργασίας για τηντην ανάπτυξη web εφαρμογών 49

Πίνακας Σχημάτων

Σχήμα 1 :Model - View - Controller 8

Σχήμα 2: Μηχανή Πεπερασμένων Καταστάσεων 8

Σχήμα 3: Rails και το MVC 10

Σχήμα 4:FSM καταστάσεις ticket 23

Σχήμα 5: Οι Βασικοί Πίνακες και οι Σχέσεις τους 24

ΚΕΦΑΛΑΙΟ 1

Εισαγωγή Βασικές Έννοιες

1.1 ΕΙΣΑΓΩΓΗ

Η ανάλυση εφαρμογής λογισμικού είναι το κομμάτι που πρέπει να γίνεται έστω ένα βήμα πριν την υλοποίηση. Οι προγραμματιστές έχουν εισάγει αυτή την ανάγκη στην δουλεία τους. Είναι πολύ πιο εύκολο να ξέρει κανείς που θέλει να φτάσει, να μπορεί να προνοεί για προβλήματα που έρχονται μελλοντικά και να τα αποφύγει ανώδυνα, πόσο μάλλον σε μια διαδικασία όπως ο προγραμματισμός που καταναλώνει πολύ χρόνο σκέψης και ενέργειας. Ειδικότερα όταν ανήκει στη παραγωγική διαδικασία και ακολουθεί τα πρότυπα του μάρκετινγκ, να γίνεται όσο πιο παραγωγικός μπορεί με λιγότερο κόστος και χρόνο.

Το web είναι το "πρόσωπο" που μας μιλά για τη μεταφορά δεδομένων και των διάφορων συσχετισμών μεταξύ αυτών για την παραγωγή νέων αποτελεσμάτων δεδομένων που εμφανίζονται στους χρήστες (ανθρώπους ή όχι)ενδιάμεσων και τελικών κόμβων πάνω σε αυτό .Από τη πλευρά των απλών χρηστών-ανθρώπων απλά δημιουργεί νέες ανάγκες και απαιτήσεις, ως αποτέλεσμα στους web developers να τους οδηγεί σε ένα συνεχές κυνήγι εύρεσης λύσεων σε νέα προβλήματα. Όσο διογκώνονται οι ανάγκες τόσο απομακρύνεται ο στόχος και οι προφανής λύση προβλημάτων. Καταλήγοντας να σπάμε τα προβλήματα σε μικρότερα τμήματα και κατηγορίες με λογικό επακόλουθο οι web εφαρμογές να είναι πολυδιάστατες και να αναζητούνται πολυδιάστατες ανάγκες σε επίπεδο λογισμικού και υλικού.

Η ανάγκη αυτοματισμού γέννησε την ιδέα να δημιουργήσουμε μηχανές σαν τους ηλεκτρονικούς υπολογιστές να δουλεύουν για τους ανθρώπους και να κερδίσουμε χρόνο . Λύνουμε ένα πρόβλημα που οδηγεί στη λύση άλλων προβλημάτων. Το ίδιο γίνετε και με τις εφαρμογές διαδικτύου. Παράγουμε συστήματα και εφαρμογές που χρησιμοποιούμε μέσο του διαδικτύου για τη μεταφορά δεδομένων και διεργασιών σε διαφορετικούς τόπους.

Συχνά καταλήγουμε σε αυτοματοποιημένες λύσεις μικρών προβλημάτων που στο σύνολο τους δίνουν τη τελική λύση. Αυτή τη λογική ακολούθησε το framework ruby on rails. Προσφέρει λύσεις για τα κομμάτια προβλημάτων που αναπόφευκτα εμφανίζονται σε μια εφαρμογή web αφήνοντας το προγραμματιστή να τα χειριστεί και να εφαρμόσει τις ιδιαιτερότητες των αναγκών των χρηστών που πρέπει να επιλύσει. Ακόμα μετατρέπει τη διαδικασία ανάπτυξης σε ένα παιχνίδι δημιουργίας.

Σε αυτή την αναφορά πτυχιακής θα δούμε πως λειτουργεί αυτή η διαδικασία από την αρχή ως το τέλος μιας εφαρμογής για τη δήλωση προβλημάτων (truble ticket). Στο πρώτο κεφάλαιο θα παρουσιάσουμε τις βασικές έννοιες. Στο Δεύτερο πως λειτουργεί και οργανώνει τα μέλη η RoR και τις υπηρεσίες που προσφέρει. Στο Τρίτο κεφάλαιο παρουσιάζεται η υλοποίηση της εφαρμογής με τις τεχνικές που παρουσιάσαμε στο δεύτερο κεφάλαιο. Στο Τέταρτο κεφάλαιο θα αναφερθούμε σε θέματα ασφάλειας για διαδικτυακές εφαρμογές και πως η RoR προτείνει λύσεις και στο Πέμπτο κεφάλαιο θα παραθέσουμε τα συμπεράσματα από την εμπειρία μας αφού πρώτα κάνουμε μια μικρή παρουσίαση άλλων frameworks.

1.2 ΤΙ ΕΙΝΑΙ TRUBLE TICKET

Truble ticket είναι ο μηχανισμός που χρησιμοποιείται από ένα οργανισμό για να ανιχνεύει αναφορές, αιτήσεις ή ανάλυση σε κάποιου είδους προβλήματος. Οι ανάγκες του οργανισμού σχηματίζουν το τύπο και τη πολιτική διαχείρισης των tickets και καθορίζονται από τις απαιτήσεις του. Ο τύπος περιέχει τη κατηγορία που ορίζετε από το κέντρο – οργανισμό ,τις πληροφορίες και τα στοιχεία που συμβάλουν στην άμεση επίλυση του προβλήματος . Η διαχείριση συνήθως έχει άμεση σχέση με το πλήθος των χρηστών και με το καταστατικό που χρησιμοποιούμε για το ticket.

Το Κέντρο Ελέγχου και Διαχείρισης Δικτύων του ΤΕΙ Κρήτης είναι οργανισμός υπεύθυνος για τη δημιουργία και διατήρηση λειτουργίας ποικίλων υπηρεσιών τηλεματικής, η δικτυακή επικοινωνίας μεταξύ των τμημάτων του και τη πρόσβαση στο Διαδίκτυο. Το truble ticket σύστημα είναι αναγκαίο να έχει διαδικτυακή υπόσταση και ηλεκτρονική δημιουργία αιτήσεων (tickets). Οι αιτήσεις δημιουργούνται και δημοσιεύονται από χρήστες μέλη του ΤΕΙ και τεχνικούς υποστήριξης. Η διαχείριση των αιτήσεων απαιτεί και διαχείριση των χρηστών. Ο τύπος των αιτήσεων διακρίνεται σε αιτήσεις τηλεφωνίας, παροχής δικτύου και υπηρεσιών (eclass, blogs κτλ), δηλώσεις βλαβών σε υπάρχουσες υπηρεσίες (πρόσβαση στο διαδίκτυο/eclass) και συσκευές χρήσης (τηλέφωνο, καλωδίωση, κτλ). Δηλώσεις βλαβών του τεχνικού εξοπλισμού (racks, switches, hubs, servers, ups κτλ) και εισαγωγή πληροφοριών για ενημέρωση μεταξύ του προσωπικού τεχνικής υποστήριξης για την ενημέρωση των βάσεων δεδομένων.

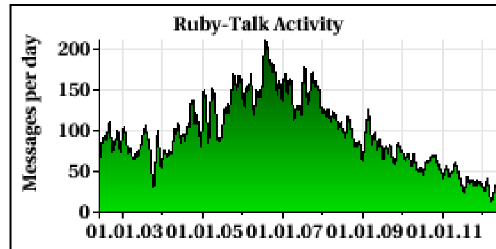
1.3 ΤΙ ΕΙΝΑΙ RUBY

Η Ruby είναι δυναμική γλώσσα προγραμματισμού ανοικτού κώδικα, δημιουργήθηκε από τον Ιάπωνα Yukihiro Matsumoto “Matz” το 1993 και άρχισε να κυκλοφορεί το 1995. Είναι επηρεασμένη από Perl στη σύνταξη και υιοθετεί χαρακτηριστικά από τη Smaltalk όπως το να δίνει μεθόδους και μεταβλητές σε όλους τους τύπους της, ενώ βασικές επιρροές έχει και από τις γλώσσες Eiffel και Lisp. Είναι ευέλικτη δεδομένου ότι αφήνει τους χρήστες της να τροποποιήσουν ελεύθερα τα μέρη της και χαρακτηρίζεται ως ένα απλό και παραγωγικό εργαλείο δημιουργίας ευανάγνωστου και καλαίσθητου στη σύνταξη κώδικα.

Πρόκειται για μια δυναμική ,ανακλαστική και αντικειμενοστραφής γλώσσα. Τα πάντα στη ruby αντιμετωπίζονται ως αντικείμενα και τα αποτελέσματα αυτών τα αντιλαμβάνεται πάλι ως αντικείμενα. Τα αντικείμενα δημιουργούνται με την κλήση ενός κατασκευαστή (constructor) και ένα ειδικό αντικείμενο (κλάση) που σχετίζεται με μια μέθοδο όπως η new().

Ο Matz έδωσε τον εξής χαρακτηρισμό στη ruby ”Η ruby είναι πολύ απλή στην εμφάνιση αλλά πολύ πολύπλοκη στο εσωτερικό της όπως το ανθρώπινο σώμα”. Το κίνητρο που οδήγησε το Matz να δημιουργήσει αυτή τη γλώσσα ήταν η ύπαρξη μιας γλώσσας μεταξύ συναρτησιακού και προστατικού προγραμματισμού όπως έχει δηλώσει: "Ήθελα μια γλώσσα σεναρίων πιο ισχυρή από την Perl και πιο αντικειμενοστραφή από την Python. Για αυτό αποφάσισα να σχεδιάσω τη δική μου γλώσσα". Το Μάρτιο του 2012 στο Παγκόσμιο συνέδριο του Ανοικτού Λογισμικού τιμήθηκε με το βραβείο FSM για την συνεισφορά του στο έργο κοινωνικής ωφέλειας από τον οργανισμό GNU Health.

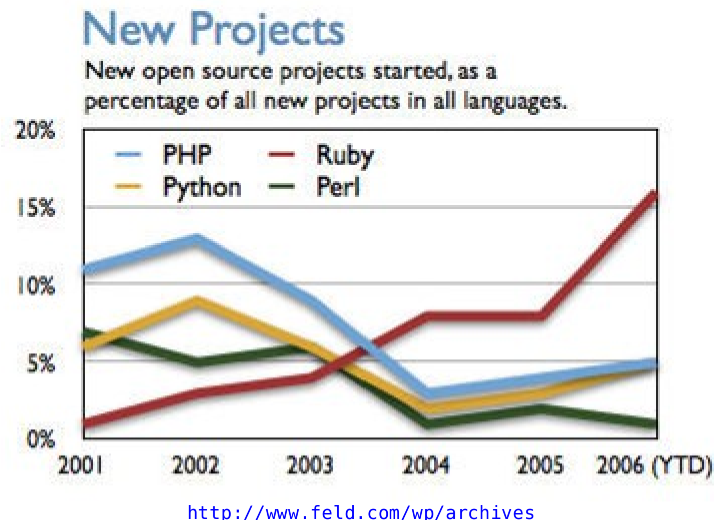
Η κοινότητα της ruby αναπτύχθηκε γρήγορα και συνέβαλε στην επέκταση των δυνατοτήτων της γλώσσας, έτσι πολύ γρήγορα αναδείχθηκαν οι ικανότητες της και σήμερα ανταγωνίζεται άλλες διάσημες γλώσσες προγραμματισμού με γερές βάσεις .Στο παρακάτω διάγραμμα βλέπουμε την κίνηση μηνυμάτων για την ruby σε καθημερινή βάση.



http://www.ruby-lang.org

Εικόνα 1: Η καθημερινή κίνηση μηνυμάτων για ruby

Η ruby είναι μια πλήρες γλώσσα με ανθρωποκεντρικό χαρακτήρα γενικού τύπου και καλύπτει όλα τα φάσματα του προγραμματισμού. Είναι συνδυασμός των απαραίτητων συστατικών για τη δημιουργία εφαρμογών. Τα χαρακτηριστικά της ruby σε σύγκριση με γλώσσες όπως PHP (γλώσσα για web development) και Java είναι περισσότερα. Η απότομη στροφή προς τη ruby δηλώνει τις δυνατότητες της. Δεδομένου ότι χρειάζεται χρόνος να αλλάξει κανείς γλώσσα γραφής προγραμμάτων δικαιώνει την επιλογή της ruby ως καλά ανερχόμενη γλώσσα προγραμματισμού αφού όλο και περισσότερα άτομα ενδιαφέρονται να μάθουν να την χρησιμοποιούν.

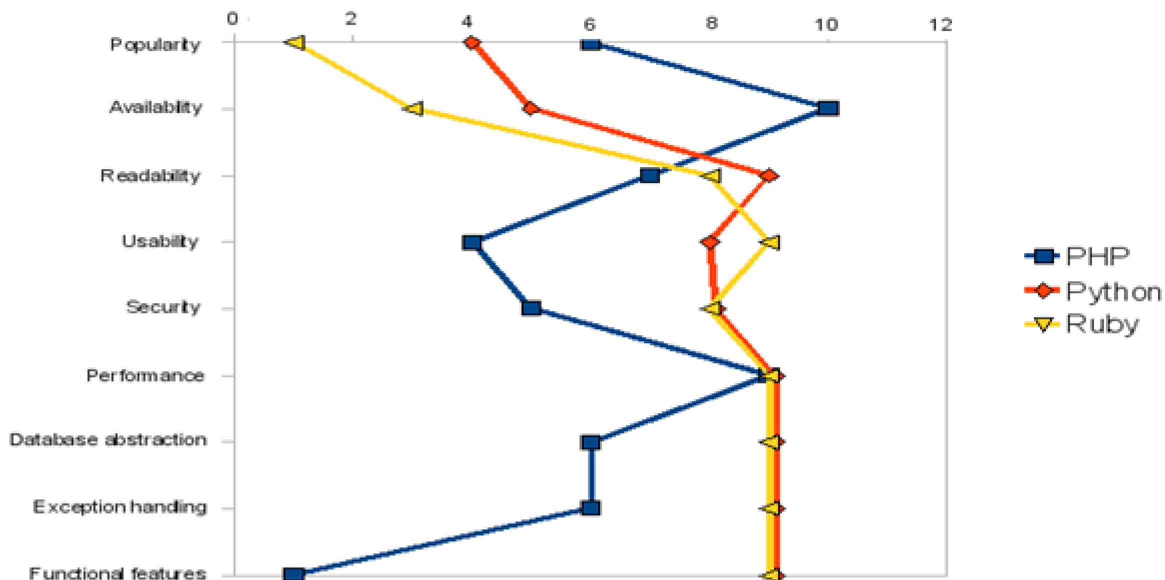


http://www.feld.com/wp/archives

Εικόνα 2 :Διάγραμμα νέων εφαρμογών σε Open Source

Παραπάνω βλέπουμε ένα διάγραμμα ανάπτυξης νέων εφαρμογών και τις γλώσσες προγραμματισμού που χρησιμοποιούνται για την παραγωγή αυτών, από το 2001 έως το 2006 στο ξεκίνημα της ruby. Ως γλώσσα προγραμματισμού έχει διάφορες από γλώσσες όπως η PHP, Python.

Τα χαρακτηριστικά που κάνουν μια γλώσσα προγραμματισμού χρήσιμη δεν είναι μόνο να είναι εύχρηστη αλλά να είναι εξελιγμένη εύκαμπτη και ασφαλής και κατανοητή. Σημαντικό είναι να έχει μια κοινότητα που να την στηρίζει και την εξελίσσει και φυσικά να την χρησιμοποιεί σε εφαρμογές. Ο σύντομος χρόνος ζωής της ruby σε σχέση με άλλες γλώσσες της παρέχει προοπτική ανάπτυξης.



PHP vs. Python vs. Ruby – The web scripting language shootout

Εικόνα 3: Χαρακτηριστικά PHP, Python, Ruby

Ο κόσμος που ασχολείται γίνετε όλο και περισσότερος ο τρόπος που έχει δομηθεί η ruby είναι ευέλικτος σε αναβάθμισες και ο ανθρωποκεντρικός της χαρακτήρας τη καθιστά με μεγάλη επιτάχυνση προς τη κορυφή.

1.4 ΤΙ ΕΙΝΑΙ Η RUBY ON RAILS

Η Ruby o Rails (RoR) είναι ένα framework γραμμένο στη γλώσσα προγραμματισμού ruby με σκοπό τη δημιουργία web-site. Φτιάχτηκε το 2004 από τον David Heinemeier Hansson κατά τη διάρκεια της ανάπτυξης του έργου 37signals' flagship. Όταν το rails χρησιμοποιήθηκε και για άλλες εφαρμογές η ομάδα της αποφάσισε να αποσπάσει το κώδικα κορμό φτιάχνοντας την αρχή του framework και τον κυκλοφόρησε ως ανοικτό κώδικα. Η ruby on rails επιτρέπει τη ταχεία ανάπτυξη εφαρμογών, αποτελείτε από κομμάτια κώδικα και όλα μαζί αλληλεπιδρούν μεταξύ τους με συγκεκριμένο τρόπο. Μερικά από τα ποιο βασικά κομμάτια που δομούν το rails είναι τα παρακάτω :

Action Pack: είναι μια βιβλιοθήκη (gem) που περιέχει Action Controller και Action View και την επικοινωνία τους (Action Dispatch) . Υλοποιεί το κομμάτι View – Controller.

Action Mailer: Είναι ένα κομμάτι του framework που δίνει τη δυνατότητα e-mail υπηρεσιών. Μπορούμε να στέλνουμε και να λαμβάνουμε e-mail προς στην εφαρμογή αλλά και να στέλνουμε μέσα από την εφαρμογή.

Active Model: Ορίζει τη διεπαφή μεταξύ του Action Pack και του Object Relationship Mapping (ORM).

Active Record: Είναι το κύριο συστατικό για τη δημιουργία του model στην εφαρμογή, δίνει ανεξαρτησία στα δεδομένα, τη δυνατότητα εύρεσης δεδομένων και δημιουργεί τις σχέσεις μεταξύ των μοντέλων

Active Resource: Φιλτράρει την επικοινωνία μεταξύ των αντικειμένων για να διευκολύνει τα web services.

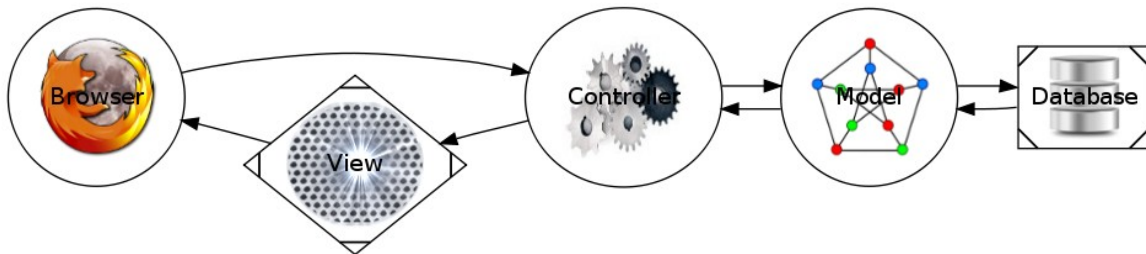
Active support : Είναι μια βιβλιοθήκη από classes από τη ruby, οι οποίες γίνονται extened στο πηγαίο κώδικα και μέσα στην εφαρμογή .

Railties :Είναι ο πηγαίος κώδικας της rails από τον οποίο δημιουργήθηκε το πρώτο στάδιο της εφαρμογής και συνδέει τα framework και plugins μέσα σε αυτήν.

1.4.1 Αρχιτεκτονική Model – View – Controller (MVC)

Η ruby on rails βασίζεται στην αρχιτεκτονική MVC (Model – View - Controller) για την κατασκευή της. Το Model View Controller είναι μια από τις αρχιτεκτονικές ανάπτυξης διαδραστικών εφαρμογών. Η δομή του επιτρέπει το διαχωρισμό της εφαρμογής σε τρία επίπεδα - κορμούς το model, το controller και το view. Καθιστά την εργασία μας πιο εύκολη και ξεκαθαρίζουμε καλύτερα τη κάθε λειτουργία του κώδικα μας.

Στο παρακάτω σχήμα βλέπουμε την επικοινωνία μεταξύ των στοιχείων της αρχιτεκτονικής η φιλοσοφία του είναι ο διαχωρισμός των διεργασιών του κώδικα ανάλογα με την ανταπόκριση του. Για το μοντέλο των αντικειμένων, την εικόνα τους και τη διαχείριση ανάμεσα σε αυτά.



Σχήμα 1 :Model - View - Controller

Το model είναι το κομμάτι που μορφοποιεί την εφαρμογή βάζει δηλαδή κανόνες και περιορισμούς και εδώ βρίσκονται όλες οι “ιδιαιτερότητες” της εφαρμογής όσο αναφορά τη αλληλεπίδραση της εφαρμογής με τους πίνακες της Βάσης Δεδομένων.

Το view είναι το μέρος όπου χτίζουμε τη διεπαφή χρήστη – υπολογιστή, το πως η εφαρμογή θα αλληλεπιδράσει με το χρήστη(συνήθως το view χτίζετε με βάση το model). Για ένα model φτιάχνουμε παραπάνω από ένα views, για παράδειγμα μπορούμε να έχουμε διαφορετικό view για του απλούς χρήστες και άλλο για ένα χρήστη admin. Στη rails συνήθως είναι αρχεία html κώδικα και ενσωματώνουμε ruby μέσα σε <% ruby κώδικας %>.

Ο controller είναι ο συντονιστής της εφαρμογής ανάμεσα στ model και στο view καθώς και με την επικοινωνία της εφαρμογής με τον έξω κόσμο(χρήστη ή άλλη εφαρμογή). Υποδέχεται τις ενέργειες ενός χρήστη ή άλλης εφαρμογής – αλληλεπιδρά με το model και δίνει στο view τα αποτελέσματα για να δράσει ανάλογα. Στη rails ο controller αναλαμβάνει την επεξεργασία των εισερχόμενων αιτημάτων από τον web brower , επεξεργάζεται τα δεδομένα από το model και περνά τα αποτελέσματα στο view.

1.4.2 Χαρακτηριστικά της Ruby on Rails

Δουλεύοντας ένα project σε ruby on rails δίνουμε έμφαση στις αρχές Σύμβαση έναντι Ρύθμισης (CoC) και την αποφυγήν επαναλήψεώς (DRY) και μέθοδος ανάπτυξης Agile.

CoC

Convention over configuration (CoC) :είναι η διαδικασία όπου ο προγραμματιστής πρέπει να αποφασίσει τι πρέπει να κάνει ώστε να γράφει λιγότερα και ευανάγνωστο κώδικα, για παράδειγμα στη ruby on rails πολύ σημασία έχουν τα ονόματα των αντικειμένων(πχ. Classes στο model και πίνακας στη Βάση Δεδομένων)

DRY

Dont Repeat Your Self (DRY) : Είναι η διαδικασία με την οποία δεν επαναλαμβάνουμε τα ίδια πράγματα πάνω από μια φορά. Η διαδικασία εφαρμόζεται σε πολλά επίπεδα στο κώδικά μέσα στο αρχείο, στο κώδικα που εμφανίζετε ξανά σε διαφορετικά αρχεία, στα layouts μια εφαρμογής ακόμα και στη δρομολόγηση RESTful. Οτιδήποτε χρειαστούμε πάνω από μία φορά η RoR μας δίνει τη δυνατότητα να το καλέσουμε χωρίς να το γράψουμε για δεύτερη φορά.

Agile

Agile είναι μεθοδολογία ανάπτυξης λογισμικού, δίνει τη δυνατότητα κατά τη διάρκεια της υλοποίησης να επαναπροσδιορίζουμε τις απαιτήσεις παραδίδοντας κάθε φορά ένα κομμάτι του έργου. Η συνολική ομάδα αποτελείται από μικρότερες δια τμηματικές ομάδες οι οποίες αυτοί-οργανώνουν το τρόπο δουλειάς τους. Βασική αρχή έχει ότι τα πρόσωπα αλληλεπιδρούν με τα εργαλεία και τη διαδικασία ανάπτυξης και το λογισμικό αναπτύσσεται σύμφωνα με τις απαιτήσεις και είναι έγκυρο σε όλα τα στάδια παράδοσης.

Ο πελάτης έχει άμεση συνεργασία και επαφή αφού κατά τη διάρκεια της ανάπτυξης μπορεί να δια πραγματευτεί τις απαιτήσεις της εφαρμογής ,συνεργάζεται όταν χρειαστεί να αλλάξει ή να εισαχθεί μια απαίτηση στο σύστημα. ΄Επίσης το παραδοτέο έργο είναι πιο έγκυρο και τεχνολογικά εξελιγμένο γιατί στη διάρκεια ανάπτυξης να καθορίζουμε καινούργιους στόχους και μεθοδολογίες ανάπτυξης προσαρμοσμένες στις καινούργιες απαιτήσεις που έχουν δημιουργηθεί στο διάστημα αυτό, σε αντίθεση με άλλες μεθόδους ανάλυσης που δεν μπορούν να το υποστηρίξουν.

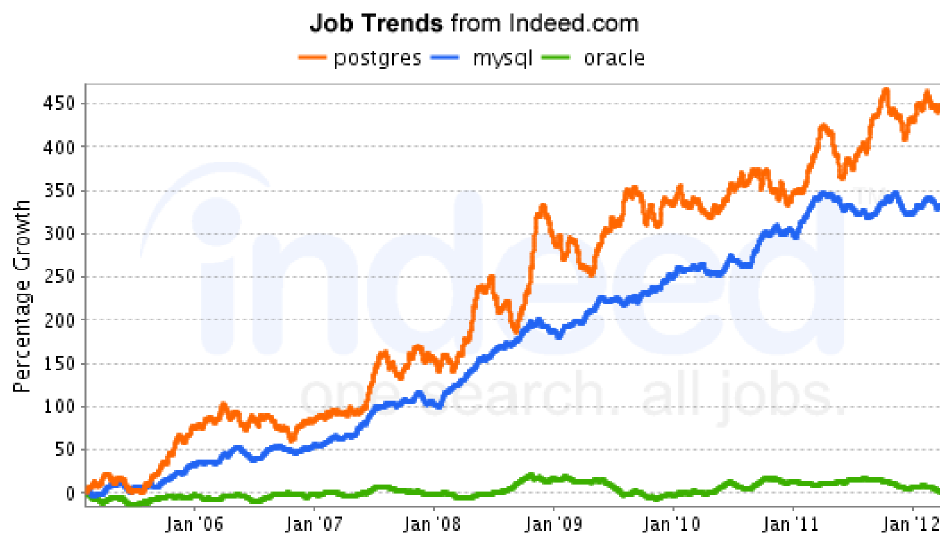
- Η μεθοδολογία Agile χαρακτηρίζετε από δώδεκα αρχές:
- Ικανοποίηση των πελατών με την ταχεία παράδοση λογισμικού
- Αποδοχή στις απαιτήσεις του λογισμικού ακόμα και στο τέλος της ανάπτυξης
- Παράδοση του λογισμικού σε τακτά χρονικά διαστήματα (ένα δύο εβδομάδες)
- Η πρόοδος ορίζετε από την εξέλιξη του λογισμικού
- Σταθερός ρυθμός ανάπτυξης
- Καθημερινή συνεργασία μεταξύ των ανθρώπων ανάπτυξης λογισμικού και της εταιρείας.
- Η πρόσωπο με πρόσωπο επικοινωνία συνιστά τε ως μέθοδο επικοινωνίας
- Η συνεργασία των ατόμων για την υλοποίηση της εργασίας στηρίζετε στην εμπιστοσύνη μεταξύ των ανθρώπων
- Εστίαση στη τεχνική ανάπτυξης και στο σχεδιασμό
- Απλότητα
- Αυτοί-οργάνωση ομάδων
- Δυνατότητα-ευελιξία στις μεταβαλλόμενες συνθήκες

1.5 Η ΠΛΑΤΦΟΡΜΑ ΒΑΣΗΣ ΔΕΔΟΜΕΝΩΝ POSTGRES

Η PostgresSQL είναι μια σχεσιακή βάση δεδομένων ανοικτού κώδικα με πολλές δυνατότητες. Η ανάπτυξη της είναι πάνω από δεκαπέντε χρόνια, προήλθε από ένα έργο του πανεπιστημίου Berkley στη Καλιφόρνια με δοκιμασμένη αρχιτεκτονική που έχει κερδίσει μια ισχυρή φήμη για την αξιοπιστία της. Η PostgresSQL ως βάση δεδομένων επιχειρηματικού επιπέδου διαθέτει εξελιγμένα χαρακτηριστικά.

- Είναι συμβατή με ACID το οποίο εγγυάται ότι οι συναλλαγές στη βάση δεδομένων λειτουργούν αξιόπιστα.
- Δυνατότητα δημιουργίας νέων τύπων δεδομένων από τους προγραμματιστές, ενώ η ίδια παρέχει τους βασικότερους τύπους όπως numeric, decimal, smallint, integer, real, double, serial, char, text, date, time, timestamp, interval, boolean.

- Αποθήκευση BLOBs μεγάλα δυαδικά αντικείμενα συμπεριλαμβανομένου αρχείων κειμένων, φωτογραφιών κ.α.
- Υποστήριξη συναρτήσεων συγκεντρωτικών αποτελεσμάτων και δυνατότητας δημιουργίας νέων από το προγραμματιστή.
- Υποστήριξη όλων των τύπων ενώσεων.
- Περιβάλλον ανάπτυξης γλωσσών προγραμματισμού όπως ruby,Perl, Python
- Υποστηρίζει τις συναρτήσεις που έχει ορίσει ο προγραμματιστής στην εφαρμογή του από πολλές γλώσσες προγραμματισμού.
- Παρέχει βιβλιοθήκη συναρτήσεων και τελεστών με ορισμένες εγκατεστημένες συναρτήσεις προσωρινούς πίνακες οι οποίοι σβήνουν μετά το τέλος της συνόδου.
- Το μέγεθος του πίνακα και της βάσης δεδομένων είναι σχεδόν απεριόριστο με απεριόριστες καταχωρίσεις και ευρετήρια ανά πίνακα
- Χρησιμοποιεί το μοντέλο ασφάλειας ομάδας/χρήστη.
- Στο παρακάτω Γράφημα βλέπουμε την ζήτηση θέσεων εργασίας σε τρεις πλατφόρμες βάσεων δεδομένων, την Postgres, την Mysql και την Oracle.



http://www.indeed.com/jobtrends?q=postgres%2C+mysql%2C+oracle&relative=1&relative=1
Εικόνα 4: Ζήτηση θέσεων εργασίας για Postgres,Mysql,Oracle

Η Postgres θεωρείται από τις πιο ποιοτικές βάσεις δεδομένων και ανταγωνίζεται άλλες βάσεις δεδομένων όπως τη MySQL και την Oracle είναι εμφανές από το παραπάνω σχήμα η αυξανόμενη ζήτηση της. Τρέχει σε όλα τα σημαντικά λειτουργικά συστήματα συμπεριλαμβανομένου των Linux, Unix και Windows.

1.6 ΜΗΧΑΝΗ ΠΕΠΕΡΑΣΜΕΝΩΝ ΚΑΤΑΣΤΑΣΕΩΝ FSM

Η Μηχανή Πεπερασμένων καταστάσεων(Finite State Mchine - FSMs) είναι είδος διαγράμματος ροής με το οποίο περιγράφουμε συστήματα που μπορούν να βρεθούν σε ένα πεπερασμένο σύνολο καταστάσεων και μεταβαίνουν από τη μια κατάσταση στη άλλη ως συνέπειας ενός γεγονότος.

Μια FSMs αποτελείτε από πέντε στοιχεία

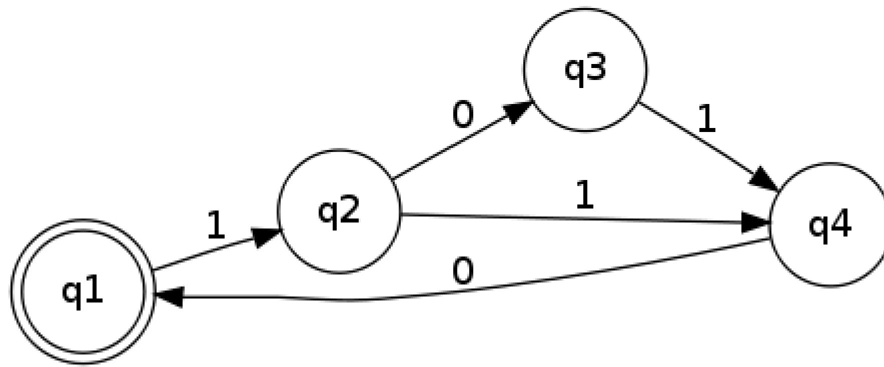
Ένα πεπερασμένο σύνολο καταστάσεων Q

Ένα πεπερασμένο σύνολο εισόδων Σ

Μία συνάρτηση μετάβασης δ: $Qx\Sigma -> Q$

Μία αρχική κατάσταση qo όπου $qo \in Q$

Ένα σύνολο F καταστάσεων αποδοχής όπου $F \subseteq Q$



Σχήμα 2: Μηχανή Πεπερασμένων Καταστάσεων

Μια Μηχανή πεπερασμένων καταστάσεων μπορεί να αναπτυχθεί γραφικά με κόμβους να συμβολίζουν τις καταστάσεις και ακμές μεταξύ των κόμβων που συμβολίζουν τις μεταβάσεις και επαληθεύουν τη συνάρτηση $\delta{:}\,Qx\Sigma -> Q$ όπως φαίνεται στο παραπάνω σχήμα. Στη πραγματικότητα είναι ένα πεπερασμένο αυτόματο και είναι μεγάλο κεφάλαιο στο τομέα των ηλεκτρονικών υπολογιστών και του λογισμικού που δημιουργούμε σε αυτούς. Μπορούμε να περιγράψουμε καταστάσεις σε ένα πηνίο, στη μνήμη ενός υπολογιστή, στη οθόνη ενός ηλεκτρονικού υπολογιστή αλλά και σε άυλα όπως σενάρια διεπαφής, μοντέλα βάσεων δεδομένων ακόμα και παραδείγματα της καθημερινής ζωής του ανθρώπου.

Αν παρατηρήσουμε τη διαδικασία προγραμματισμού για την παραγωγή λογισμικού δεν απέχουμε πολύ από τη δημιουργία μιας μεγάλης πολύπλοκης μηχανής πεπερασμένων καταστάσεων που αποτελείτε από πολλές μικρότερες. Η μηχανή πεπερασμένων καταστάσεων δεν είναι απλά γραφήματα αλλά και κομμάτι προγράμματος όχι υλοποιημένο με κάποια γλώσσα προγραμματισμού αλλά ο μηχανισμός του.

ΚΕΦΑΛΑΙΟ 2

Λειτουργίες & Εργαλεία Ανάπτυξης

2.1 ΤΑ ΛΕΙΤΟΥΡΓΙΚΑ ΧΑΡΑΚΤΗΡΙΣΤΗΚΑ ΤΗΣ RUBY ON RAILS

Όπως προαναφέραμε το framework RoR αποτελείτε από μικρότερα κομμάτια οι λειτουργίες αυτών συνθέτουν την εφαρμογή. Σε αυτό το κεφάλαιο θα εξηγήσουμε την λειτουργία των βασικών χαρακτηριστικών και τη μεταξύ τους επικοινωνία ώστε να είναι πιο κατανοητός ο τρόπος γραφής κώδικα .

2.1.1 Η δρομολόγηση REST των αιτήσεων σε εφαρμογή της RoR

Το MVC στη rails λειτουργεί με τη βοήθεια της Αντιπροσωπευτικής Κατάστασης Μεταφοράς (Representational State Transfer REST), ως οδηγό δρομολόγησης. Το REST είναι η σύμβαση για το routing στην εφαρμογή και όταν κάτι τηρεί αυτή τη σύμβαση λέγετε δρομολογούμενο (RESTful). Με τον όρο δρομολόγηση στη rails αναφερόμαστε στο πόσο τα αιτήματα μπορούν να δρομολογηθούν μέσα στην εφαρμογή. Η ικανότητα δρομολόγησης μιας αίτησης προσφέρει ωφέλιμες λειτουργίες στη rails όπως τον καθορισμό της αιτήσεις αν αυτό μπορεί να σταλεί ή όχι.

Το REST όσο αναφορά τη rails συνοψίζει δυο βασικές αρχές χρησιμοποιεί αναγνωριστικά για να βρει τους πόρους και κάνει αναπαράσταση αυτών σύμφωνα με την εφαρμογή. Τα ερωτήματα που θέτονται στον εξυπηρετητή (server) έχουν ένα ρήμα (verbs) τα οποία είναι GET,POST,PUT και DELETE.

Η αίτηση GET χρησιμοποιείται για να ζητήσουμε από τον server να επιστρέψει την αναπαράσταση μια πληροφορίας. Για παράδειγμα όταν κάνουμε περιήγηση σε μια ιστοσελίδα ο ο brower δημιουργεί GET ερωτήματα. Ο εξυπηρετητής θα ανταποκριθεί για να δώσε στον brower τον απαραίτητο κώδικα της ιστοσελίδας, οι πρόσθετοι πόροι που θα αποσταλούν (εικόνες stylesheets, κείμενα, κτλ.) και τα ζήτησε ο brower χαρακτηρίζονται ως GETs.

Το POST χρησιμοποιείται για την καταχώρηση δεδομένων σε ένα web brower. Για παράδειγμα έχουμε μια σύνδεση μεταξύ πελάτη (client) και διακομιστής (server) όπου ο χρήστης πρέπει να συμπληρώσει μια φόρμα με δεδομένα ' κάνοντας κλικ για να τα στείλει, υποβάλει αυτόματα μια αίτηση POST προς τον εξυπηρετητή.

Το PUT χρησιμοποιείται για δημιουργήσει ή να ενημερώσει μια αναπαράσταση ενός πόρου σε ένα διακομιστής . Αν για παράδειγμα θεωρήσουμε ως πόρο μια αλλαγή εγγραφής χρήστη και θέλουμε να την ανανεώσουμε (update) το PUT παίρνει όλη τη πληροφορία του πόρου και την τοποθετεί στη μοναδική διεύθυνση URL που είναι διαθέσιμη για την συγκεκριμένη εγγραφή χρήστη στον διακομιστής .

Η διαφορά μεταξύ το POST και PUT είναι στο πως ο διακομιστής θα πρέπει να χειριστεί το ωφέλιμο πόρο. Αν το αίτημα είναι POST τότε το συγκεκριμένο url είναι ο πόρος που δοθεί στην εφαρμογή (κίνηση προς τα έξω)ενώ αν το αίτημα είναι PUT το url οδηγεί στο πόρο που θα επεξεργαζόμαστε (κίνηση προς τα μέσα).

Το DELETE χρησιμοποιείται για να καταστρέψει ένα πόρο.

Το PATCH θα υποστηρίζετε από την έκδοση Rails 4 και να προσθέσει νέες δυνατότητες μέσο του πρωτοκόλλου HTTP (Hypertext Transfer Protocol).Η πρόταση HTTP PATCH έχει την ικανότητα να ενημερώνει ένα τμήμα κάνει δηλαδή μερική τροποποίηση του υπάρχον πόρου σε αντίθεση με τη PUT που κάνει αντικατάσταση ολόκληρου του πόρου βάζοντας στο παλιό πόρο τον νέο.

Ρήμα	Path	Δράση	Χρησιμότητα
GET	/toys	index	Εμφανίζει μια λίστα με όλα τα toys
GET	/toys/new	new	Επιστρέφει μια φόρμα HTML για τη δημιουργία νέου toy
POST	/toys	create	Δημιουργία στοιχείου toy
GET	/toy/:id	show	Επιστρέφει το toy που καθορίζει το id
GET	/toys/:id/edit	edit	Επιστρέφει μια φόρμα HTML για τη επεξεργασία του toy με που καθορίζει το id
PUT	/toys/:id	update	Κάνει αναβάθμιση στο συγκεκριμένο στοιχείο toy
DELETE	/toys/:id	destroy	Διαγραφή του στοιχείου toy που καθορίζει η τιμη του id

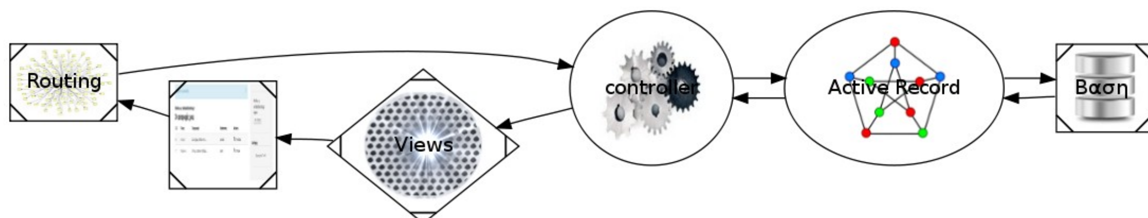
Πίνακας 1: Πίνακας routes

Στο παραπάνω πίνακα βλέπουμε μερικά παραδείγματα και τις αντιστοιχίες των αιτήσεων POST,DELETE,PUT και GET με τα url και τις ενέργειες προς μια εφαρμογή με ένα πίνακα ονόματος Toy και τα δεδομένα .

2.1.2 Η λειτουργία του MVC στη RoR

Η RoR έχει επιλέξει τη δομή του MVC ώστε να αναπτύξουμε το μοντέλο, τον ελεγκτή και την εικόνα ανεξάρτητα μέσα στην εφαρμογή μας. Μια από τις αρετές της RoR είναι να δημιουργεί αυτόματα την επικοινωνία μεταξύ των στοιχείων του MVC και δεν χρειάζεται από εμάς να κάνουμε κάποια ρύθμιση πάνω σε αυτό.

Η Λειτουργία είναι ως εξής, δεχόμαστε μια εξωτερική αίτηση προς την εφαρμογή αυτή παραλαμβάνεται αρχικά από ένα router ο οποίος δουλεύει έξω από την εφαρμογή και είναι αυτός που αποφασίζει που θα σταλεί την αίτηση μέσα στην εφαρμογή (σε ποιο controller αντιστοιχεί) και πως θα αναλυθεί το ίδιο το αίτημα. Η διαδικασία αυτή προσδιορίζει μια συγκεκριμένη μέθοδο από τις υπάρχουσες μέσα στο κώδικα του controller, αυτή τη μέθοδο στη RoR την ονομάζουμε action (δράση) .



Σχήμα 3: Rails και το MVC

Το action θα εξετάσει τα δεδομένα που υπάρχουν για την αίτηση στο controller θα μπορεί ακόμα να επιδράσει με το Active Record (model) και να εκτελέσει και άλλα actions που επικαλούνται. Τέλος θα στείλει τα αποτελέσματα στο view που με τη σειρά του θα δώσει τα αποτελέσματα στον έξω κόσμο.

Η εισερχόμενη αίτηση έχει μια συνιστώσα που την ξεχωρίζει, περιέχει μια διαδρομή π.χ /toys/3 url και μια μέθοδο από τις POST, GET, PUT, DELETE. Έτσι η RoR παίρνει το πρώτο μέρος της διαδρομής και το id του συγκεκριμένου toy και ο controller συνδέεται με το action μέσα στη κλάση του και ενεργεί ανάλογα με το κώδικα της κλάσης. Ζητά από το μοντέλο να βρει τις πληροφορίες που έχει για το πεδίο με το συγκεκριμένο id , το μοντέλο παίρνει το ερώτημα του controller και γνωρίζει τι ακριβώς πρέπει να κάνει για να βρει την απάντηση και έχει δυνατότητα να επικοινωνήσει με τη βάση δεδομένων αν χρειαστεί να πάρει περισσότερα στοιχεία ύστερα επιστρέφει την απάντηση στο controller. Ο controller επικαλείτο τον αντίστοιχο κώδικα στο view που με τη σειρά του δίνει τη κατάλληλη διεπαφή με τα αποτελέσματα στο χρήστη.

Οι ενέργειες μέσα στη RoR για την επικοινωνία του MVC είναι τα κομμάτια όπως το Active Record, το Active Model και το Action Pack. Παρακάτω θα δούμε σε πιο μέρος ακριβώς δουλεύουν και τι αρμοδιότητες έχουν εκεί.

Δομή Model

Στις διαδικτυακές εφαρμογές έχουμε την ανάγκη διατήρηση δεδομένων σε μια σχεσιακή Βάση Δεδομένων. Το **Active Model** είναι το κομμάτι της RoR που αναλαμβάνει να συνδυάσει τη σχεσιακή Β.Δ. με την αντικειμενοστραφή γλώσσα ruby.

Τα αντικείμενα στο μοντέλο είναι όλα σχετικά με τα δεδομένα και τις λειτουργίες των πινάκων που τα αντιστοιχούμε και οι τιμές των πεδίων αντιστοιχούνε στη Β.Δ. .Για να πετύχουμε την αντιστοιχία η RoR χαρτογραφεί τους πίνακες της βάσης σε κλάσεις(**ORM Object Relation Mapping**). Αν για παράδειγμα η βάση έχει ένα πίνακα με το όνομα *toy* η εφαρμογή έχει μια κλάση με το ίδιο όνομα(Toy). Τα στοιχεία μιας γραμμής μέσα στο πίνακα αντιστοιχούν σε αντικείμενα της κλάσης του πίνακα. Στα αντικείμενα αυτά δίνουμε χαρακτηριστικά για να ορίσουμε τα πεδία στον αντίστοιχο πίνακα(για παράδειγμα στη κλάση Toy έχουμε τα αντικείμενα material και demotion).
Η αντιμετώπιση που έχει η RoR προς τη βάση μας δίνει την ικανότητα να χειριζόμαστε σε επίπεδο μεθόδων τις διεργασίες στους πίνακες της εφαρμογής. Για παράδειγμα όταν θα χρειαστούμε να βρούμε το τρίτο στοιχείο του πίνακα toy το ζητάμε με τη μέθοδο της κλάσης και μας επιστέφει το αντίστοιχο αντικείμενο.

```
toy= Toy.find(3)

   puts “το παιχνίδι σου είναι#{toy.name}, και είναι χρήσιμο
για#{toy.demotation}”
```

Για τα αντικείμενα μέσα στο Toy που αντιστοιχούν σε μεμονωμένα στοιχεία-πεδία του πίνακα χρησιμοποιούμε πάλι μεθόδους. Ως παράδειγμα υποθέτουμε ότι στο πίνακα μας στα στοιχεία που έχουν κατασκευή (material) από ξύλο θέλουμε το αντίστοιχο πεδίο ιδιότητα (demotation) να πάρει τη τιμή “feel nature”.

```
Toy.where(material:”wood”).each do |t|

   t.demotation = “feel nature”

   t.save

end
```

Οι μέθοδοι της κλάσης Toy χρησιμοποιούνταν για την παράσταση εν θέατρο του πίνακα, στο επίπεδο της εφαρμογής και οι μέθοδοι μέσα σε αυτές (toy.demotation και toy.save) εκτελούν λειτουργίες στις γραμμές και τα πεδία του πίνακα. Με αυτό το τρόπο η ORM χαρτογραφεί τους πίνακες, τις γραμμές των πινάκων και τα διαφορετικά στοιχεία αυτών .

Το **Active Record** είναι το κομμάτι που αναλαμβάνει να κάνει τη διαδικασία ORM στη RoR. Είναι αυτό που θα επιλέξει πιο μοντέλο αντιστοιχεί στη Βάση, ακολουθεί τη λογική ότι μετατρέπει το πίνακα σε κλάση, τη γραμμή του πίνακα σε αντικείμενο και το πεδίο της γραμμής σε γνώρισμα του αντικειμένου. Πέρα από αυτό είναι σχεδιασμένα για μία επιμέρους λειτουργία. Το Active Record ενσωματώνετε στο REST της RoR έτσι αν κάποιο αίτημα που έρχεται από το brower (μέσο του controller) και απευθύνετε σε ενεργό αντικείμενο φιλτράρεται από το model η εγκυρότητα του. Αν για παράδειγμα έχουμε βάλει περιορισμούς τιμών και το αίτημα έχει λάθος τιμή μπορεί να εντοπιστεί στο model και να καταλήξει στο view ένα μήνυμα λάθους πάνω στη φόρμα.

Δομή view

Η αλληλεπίδραση του view είναι άμεση με το controller το πρώτο λαμβάνει δεδομένα από το δεύτερο για δώσει τη κατάλληλη έξοδο και ο controller τροφοδοτείται με αιτήσεις που λαμβάνει από τις εξόδους του view, γι' αυτό τις ομαδοποιούμε στο ίδιο κομμάτι στη RoR το **Action Pack**.

Το view στο framework είναι υπεύθυνο για το συνολική ή μέρους της εξόδου που έχουμε στην ιστοσελίδα μας ή σε ένα email που στέλνει η εφαρμογή. Αποτελείτε κυρίως από κώδικα HTML που περιέχει κείμενα αλλά και διαδραστικά στοιχεία με το χρήστη. Αν θέλουμε να περιέχει κάτι δυναμικό όπως το να εμφανίζει τα πεδία ενός πίνακα πρέπει να υπάρχει η αντίστοιχη μέθοδος στο controller. Ο τρόπος για να εισάγουμε κώδικα ruby σε αρχεία view γίνετε με τη χρήση του εργαλείου προτυποποίησης **Embedded Ruby (ERB)** . Το ERB μπορεί να εισάγει javascript για τον sever που θα καταλήξει στο brower.

Δομή controller

Ο controller στη RoR είναι το λογικό κέντρο της εφαρμογής, είναι ο εγκέφαλος που σκέπτεται και δίνει εντολές στα μέλη (montel και view) του πως θα κινηθούν. Σε αυτό το μέρος ο κώδικα μας επικεντρώνετε στη λειτουργία της εφαρμογής αφήνοντας στο παρασκήνιο τις ενδιάμεσες λειτουργίες προσθέτει ένα ακόμα πλεονέκτημα στην ανάπτυξη κώδικα. Ο controller παρέχει σημαντικές υπηρεσίες όπως:

- Είναι υπεύθυνος για την δρομολόγηση των εξωτερικών αιτήσεων και τις κατευθύνει σε αντίστοιχο εσωτερικό action, διαχειρίζεται τα ανθρωποκεντρικά URLs πολύ καλά.
- Διαχειρίζεται τη προσωρινή μνήμη προσθέτοντας στη εφαρμογή απόδοση.
- Διαχειρίζεται τα helpers που επεκτείνουν την εμφάνιση της εφαρμογής χωρίς να χρειάζεται να διογκώνουμε το κώδικα μας για αυτό το θέμα.
- Διαχειρίζεται τις συνεδρίες των χρηστών και δίνει την εντύπωση στους χρήστες τη συνεχή αλληλεπίδραση με την εφαρμογή.

Η RoR χτίζει την εφαρμογή πάνω στην αρχιτεκτονική MVC , όμως δίνει την ελευθερία να παραβούμε το κανόνα αν αυτό κριθεί από το προγραμματιστή και είναι πλεονέκτημα της. Το view από μόνο του προσφέρει αυτή τη παρατυπία στην αρχιτεκτονική, κώδικας που θα ανήκε στο controller η στο model εισάγετε στο view. Επίσης ένας κώδικας μπορεί να είναι γραμμένος στο controller και να ανήκει στο model είναι εξίσου λειτουργικός. Η κατάχρηση της αρχιτεκτονικής μπορεί να αποδεχτεί δυσλειτουργική σε μια εφαρμογή και η RoR μας καθοδηγεί να ακολουθήσουμε την αρχέτυπη λογική δημιουργίας δυναμικών ιστοσελίδων δίνοντας την ελευθερία να παραβούμε το κανόνα όταν αυτό κριθεί απαραίτητο σε ειδική περίπτωση.

2.2 ΜΕΘΟΔΟΙ ΑΝΑΠΤΥΞΗΣ ΕΦΑΡΜΟΓΩΝ ΣΕ RoR

Στη διαδικασία ανάπτυξης κώδικα της εφαρμογής η rails εκτός από το μοντέλο γραφής έχει να μας προτείνει τρόπους εκπόνησης κώδικα. Αν ακολουθήσουμε τη διαδικασία παράγουμε σενάρια χρήσης της εφαρμογής , αυτό έχει σαν αποτέλεσμα την εύρεση νέων πρωτότυπων λύσεων σε προβλήματα υλοποίησης εφαρμογών διευρύνοντας το πεδίο προγραμματιστικών λύσεων.

2.2.1 Testing στη RoR

Στο κόσμο της ruby on rails μεγάλη σημασία δίνουμε στα testing, ειδικότερα στα Test Drive Development(TDD) και Test Behaviour Development(BDD). Η χρήση των καλών τεχνικών testing είναι ένας έμπιστος τρόπος για να βεβαιωθούμε ότι τίποτα δε πάει στραβά στο κώδικα της εφαρμογής μας και όλα δουλεύουν σύμφωνα με τις επιλογές μας και ακολουθίες των σεναρίων μας. Η δημιουργία test δηλώνει ταυτόχρονα και την ύπαρξη κάποιου μηνύματος που θα οδηγήσει στο λάθος αν κάτι πάει στραβά με τον κώδικα, με αυτό τρόπο κρατάμε καλύτερα τον έλεγχο σε μικρές και σε πολύπλοκες εφαρμογές χωρίς να χανόμαστε όταν κάτι χτυπήσει κόκκινο.

TDD

Η TDD μεθοδολογία αποτελείται από τη σύνταξη ενός test που περιέχει την μια υποτυπώδες περιγραφή – σενάριο το τι θέλουμε να κάνουμε, γράφουμε τον αντίστοιχο κώδικα που υλοποιήσει το test και αφού τρέξει και λειτουργεί σωστά κάνουμε refactory. Έτσι έχουμε το πλεονέκτημα με το testing ότι βάζει το προγραμματιστή στη διαδικασία να σκεφτεί πως μπορεί να τρέχει ο κώδικας του τρωτού γραφτεί και ενώ έχει ένα αυτοματοποιημένο test και μπορεί να το τρέχει οποιαδήποτε στιγμή χρειαστεί για να δοκιμάζει αν ο κώδικας λειτουργεί με το τρόπο που σχεδίασε. Το παρακάνω είναι ένα παράδειγμα TDD για την αρχική σελίδα σε μια ιστοσελίδα.

```
require 'test_helper'

class HomeControllerTest < ActionController::TestCase

   test "should get index" do

      get :index

            assert_response :success

      end

   end

end
```

BDD

Το BDD είναι μεθοδολογία που βασίζεται πάνω στο TDD με τη διαφορά ότι δημιουργεί αυτοματοποιημένα test για τον ελέγχουμε την αλληλεπίδραση μεταξύ των διάφορων τμημάτων του βασικού κώδικα και όχι πως το κάθε τμήμα λειτουργεί ανεξάρτητα σε αντίθεση με το TDD. Με τη χρήση testing είμαστε σίγουροι για την εφαρμογή μας. Ακόμα όταν συνδέουμε μικρότερα κομμάτια στο σύνολο γνωρίζουμε πως αυτό πιθανά μπορεί να αντιδράσει.

```
require 'test_helper'
```

```
require 'rails/performance_test_help'


class BrowsingTest < ActionDispatch::PerformanceTest
  self.profile_options = { :runs => 5, :formats => [:flat] }
  def test_homepage
    get '/'
  end
end
```

Το παραπάνω είναι παράδειγμα BDD τεστ για την εμφάνιση σελίδων στο brower και περιέχει μικρότερα τμήματα της εφαρμογής.

2.2.2 Μεθοδολογία Agile ανάπτυξης κώδικα

Στη RoR είναι Agile, δεν είναι απλά ένας τρόπος ανάλυσης λογισμικού είναι μεθοδολογία για τη δημιουργία κώδικα και μπορεί να λειτουργεί ανεξάρτητα από τη μεθοδολογία που έχουμε αναλύσει το έργο πριν την υλοποίηση. Για να δημιουργήσουμε κώδικα φτιάχνουμε σενάρια χρήσης και θέτουμε στόχους ή σκοπούς που αφορούν την εφαρμογή. Τα προβλήματα που έχουμε να λύσουμε σπάνε σε μικρότερα έτσι δημιουργούμε ανεξαρτησία μεταξύ τους και είναι δυνατόν να δουλεύουν παραπάνω από μια ομάδα παράλληλα χωρίς αλληλεξάρτηση στο ίδιο έργο. Έτσι η κάθε ομάδα αναδιοργανώνει την εργασία της και την υλοποίηση που έχει να κάνει εις πέρας και βελτιστοποιείται το έργο χρονικά.

Βέβαια τίποτα δεν είναι πιο γρήγορο και αποτελεσματικό από το χρησιμοποιήσουμε ταυτόχρονα τη μεθοδολογία Agile στην υλοποίηση και στην ανάλυση της εφαρμογής μας. Υπάρχει ιδιαίτερο ενδιαφέρον για την μέθοδο ως ανάπτυξη κώδικα και ανάλυσης.

Σε πρώτο στάδιο όσο αναφορά την ανάλυση μια εφαρμογής το γεγονός ότι οι προγραμματιστές έρχονται σε επαφή με το πελάτη κατανοούν καλύτερα τις ανάγκες του και είναι ικανότερος να παρέχει αποτελεσματικότερες υπηρεσίες. Η διαπροσωπική επαφή αυξάνει το επίπεδο επικοινωνίας καταστέλλει τις ασάφειες που δίνουν συχνά οι πελάτες που ακόμα κι αν υπάρξουν λόγο ότι οι διαδικασίες είναι επαναληπτικές και τακτικές τις ξεκαθαρίζουμε. Ακόμα ο προγραμματιστής κερδίζει την εμπιστοσύνη του πελάτη γιατί έχει να του παρουσιάσει σε μικρό χρονικό διάστημα αποτελέσματα. Σε επαγγελματικό επίπεδο αυτό είναι σημαντική προϋπόθεση επιτυχίας.

Όσο αναφορά την εργασία του προγραμματιστή πέρα από δημιουργική γίνετε και διασκεδαστική. Η ομαδική εργασία καταστέλλει τους διαχωρισμούς ατόμων ο ένας μαθαίνει από τον άλλον και συνεργάζονται, η συλλογική εργασία δημιουργεί ρυθμό και όταν μια ομάδα δουλεύει υπό αυτές τις συνθήκες έχει εκθετική ανάπτυξη. Μια ομάδα έχει ως ιδανικό αριθμό ατόμων πέντε έως επτά. Ο προγραμματιστής ως εργαζόμενος γίνετε παραγωγικός και όχι ανταγωνιστικός προς τους συναδέλφους του. Το Agile δημιουργεί καλό κλίμα εργασίας.

Στο δεύτερο επίπεδο εκείνο του πελάτη έχει αντίκτυπο η εργασία των προγραμματιστών, ενημερώνετε συνεχώς με νέο παραδοτέο έργο. Αυτό δημιουργεί την αίσθηση συνέπειας, το έργο παίρνει τη μορφή που έχει συμβάλει και ο ίδιος με τις προτάσεις του .Αυτό που παραλαμβάνει είναι ένα προϊόν που κατευθύνετε πλήρως στις απαιτήσεις του και η διαπροσωπική επαφή έχει γεφυρώσει μια σχέση επικοινωνίας με το δημιουργό του. Σε επιχειρησιακό επίπεδο οι πελάτες αισθάνονται ωραία όταν πριν την παράδοση βλέπουν κάτι που ανταποκρίνεται στη πραγματικότητα που αυτοί το είχαν φανταστεί ακόμα και αν το τελικό έργο δεν καλύπτει πια ακριβώς στις αρχικές τους προσδοκίες η όλη διαδικασία τους πείθει για να αποδεχτούν άλλες προτάσεις και να συνεργαστούν, ακόμα και είναι προτάσεις που είχαν απορρίψει κατηγορηματικά.

2.2.3 Αποφυγή επαναλήψεων κώδικα DRY

Μερικές από τη δυνατότητες που μας παρέχει η RoR να ακολουθούμε αυτή τη μέθοδο DRY στην ανάπτυξη είναι η δημιουργία partial. Τα partial είναι αρχεία στο view που το όνομα τους ξεκινά με “_” και αποτελούν μικρότερα κομμάτια της συνολικής ιστοσελίδας και καλούνται στο layout με τη εντολή render :partial => "όνομα_partial".

Όταν παρατηρούμε ότι χειριζόμαστε ορισμένες μεθόδους κλήσεις για τα αντικείμενα ή μοντέλα παραπάνω από μια φορά τότε δημιουργούμε **scopes.** Με αυτό το τρόπο φτιάχνουμε στο μοντέλο συντομεύσεις και ανακαλούμε μόνο τη συντόμευση κάθε φορά που θέλουμε να την χρησιμοποιήσουμε χωρίς να επαναλαμβάνουμε το κώδικα.

```
class Toy < ActiveRecord::Base

   scope :material, where(:material => 'wood').joins(:category)

end
```

Το μεγαλύτερο κομμάτι του κώδικα συνήθως είναι στο μοντέλο view για να εξομαλυνθεί η κατάσταση για actions μεταξύ του controller και των διαφορετικών καταστάσεων εμφάνισης χρησιμοποιούμε τους helpers για να αποφύγουμε εντολές επιλογών όπως η if μέσα στο view.

2.2.4 Asset Pipeline

Η RoR προσφέρει εξτρά υπηρεσίες όπως για τη καλύτερη διαχείριση των αρχείων javascript και CSS που αναπόφευκτα χρειάζονται για τη καλύτερη λειτουργία της διεπαφής και τις ανάγκες των πελατών σε μια ιστοσελίδα.

Το Asset Pipeline είναι προεπιλεγμένο εργαλείο στο framework και το χρησιμοποιούμε για να ενώσουμε και να συμπιέσουμε στοιχεία javascript CSS, ενώ μας επιτρέπει να γράφουμε παρουσιαστικά στοιχεία σε γλώσσες coofescript , Sass και ERR αν το προτιμήσουμε. Η σμίκρυνση των assets είναι σημαντική γιατί μειώνουμε τις αιτήσεις στο brower και αυξάνουμε τη ταχύτητα όταν φορτώνουμε την εφαρμογή μας.

Στη Rails 3 έχουμε τη δυνατότητα να συγκεντρώνουμε όλα τα στοιχεία της javascrript σε ένα αρχείο .js και όλα τα αρχεία CSS σε ένα αρχείο .cs . Από τη συμπίεση παράγετε ένα αρχείο με MD5 κωδικό και αποθηκεύεται στ web-brower. Όταν τα assets τις εφαρμογής αλλάξουν ακυρώνετε το αρχείο από την μνήμη του brower και το αντικαθιστούμε με το νέο. Το όνομα του αρχείου διαμορφώνετε με παράμετρο τα περιεχόμενα και ο brower καταλαβαίνει αν ενημερώθηκαν τα aseets από το όνομα του συμπιεσμένου αρχείου.

Το Pipeline κάνει συμπίεση στα CSS και Javascript αφαιρεί κενά και σχόλια ή με άλλες διεργασίες. Αν ένα αντικείμενο στο brower περνά από μια σειρά καταστάσεων επιλογής κρατάει μόνο τη τελευταία. Τέλος επιτρέπει την κωδικοποίηση των αρχικών αρχείων assets απο μια υψηλότερη γλώσσα με τη διαδικασία precompilation πρώτου ανεβάσουμε την εφαρμογή στο server.

2.3 ΕΡΓΑΛΕΙΑ ΠΟΥ ΧΡΗΣΙΜΟΠΟΙΗΘΗΚΑΝ

Για την συνολική ανάπτυξη μια δικτυακής εφαρμογής χρειαζόμαστε εργαλεία διαφορετικών στρωμάτων. Εργαλεία που μας βοηθάνε στην γραφή κώδικα όπως οι βιβλιοθήκες της ruby οι οποίες κρατάνε τη λύση προβλημάτων που περιέχονται στην εφαρμογή, εργαλεία που προσφέρουν ευκολότερη διαχείριση της εφαρμογής και των εκδόσεων της και εργαλεία που οδηγούν την εφαρμογή από την ανάπτυξη στη παραγωγή.

2.3.1 Βιβλιοθήκες της RoR

Βιβλιοθήκη (gem) είναι μια μικρή εφαρμογή ή plugins που εύκολα μπορούμε να την εγκαταστήσουμε στο project μας και να χρησιμοποιήσουμε τις μεθόδους της στη εφαρμογή μας. Ένα gem χαρακτηρίζεται από το όνομα και την έκδοση του. Η εγκατάσταση gem απαιτεί το ένα κομμάτι της ruby το Ruby Version Manager (RVM).Παρακάτω παρουσιάζουμε τρεις βιβλιοθήκες και τα χαρακτηριστικά τους.

BOOTSTRAP

Το Bootstrap είναι ένα εργαλείο για web-desing σχεδιασμένο από το Twitter, σκοπός του είναι να δώσει το έναυσμα για τη δημιουργία ιστοσελίδων. Περιλαμβάνει βιβλιοθήκες για CSS, HTML, Javascript, για τη δημιουργία κομψών και λειτουργικών διαδραστικών αντικειμένων όπως φόρμες κουμπιά, εικονίδια, πίνακες, labels, layouts, sidebars κ.α. προσφέροντας εύκολη παρέμβαση στο view ενός διαδικτυακού συστήματος. Ένα gem ενσωματώνει τα assets του bootstrap στο asset pipeline.

DEVISE

Το devise είναι βιβλιοθήκη της rails για τη διαχείριση των χρηστών είναι γραμμένη από τον Daniel Neighman που την έκδωσε ώς Open Source. Το devise αποτελείτε από δώδεκα ενότητες (modules).

- Database Authenticatable η οποία κρυπτογραφεί και αποθηκεύει έναν κωδικό στη Β.Δ. Για την επικύρωση του χρήστη. Η εξακρίβωση γίνεται μέσο POST request ή HTTP Basic Authentication.
- Token Authenticatable: “Πιστοποίηση ενός χρήστη η οποία μπορεί να δοθεί μέσο ενός ερωτήματος ή HTTP Basic Authentication
- Omniauthable Υποστηρίζει τη βιβλιοθήκη omniauth η οποία τυποποιεί την αυθεντικότητα των χρηστών από διαφορετικούς φορείς.
- Confirmable στέλνει email για να μπορούμε να επιβεβαιώσουμε την εγγραφή μας μέσω του email μας.
- Recoverable μας δίνει τη δυνατότητα ανάκτησης του κωδικού
- Registrable είναι το μοντέλο το οποίο χειρίζεστε την εγγραφή χρήστη και την επεξεργασία, διαγραφή λογαριασμού.
- Rememberable διαχειρίζεται τη δυνατότητα να κρατάμε cookies ώστε να “θυμόμαστε τον χρήστη”.
- Trackable
- Timeoutable λήγει την συνεδρία (session) όταν δεν υπάρχει καμία δραστηριότητα για κάποιο χρονικό διάστημα

- Validatable: παρέχει την επικύρωση του ηλεκτρονικού ταχυδρομείου και τον κωδικό πρόσβασης
- Lockable κλειδώνει το λογαριασμό μετά από συνεχόμενες αποτυχημένες προσπάθειες για login
- Encryptable: προσθέτει υποστήριξη των άλλων μηχανισμών πιστοποίησης εκτός από το ενσωματωμένο Bcrypt

Ο προγραμματιστής έχει τη δυνατότητα να επιλέξει ποια μοντέλα θέλει να χρησιμοποιήσει ενώ έχει διαθέσιμες πολύ σημαντικές μεθόδους ασφάλειας και χρηστικότητας όσο αναφορά τις συνεδρίες των χρηστών όπως:
μοντέλο_signed_in?
current_μοντέλο
μοντέλο_session
AASM
Το AASM είναι μια βιβλιοθήκη για την δημιουργία μηχανής πεπερασμένων καταστάσεων, ξεκίνησε ως plugin για πράξεις σε καταστάσεις μηχανής και εξελίχθηκε σε μια ποιο γενική βιβλιοθήκη. Τα χαρακτηριστικά που προσφέρει είναι κατάσταση, μηχανή, γεγονότα και μεταβάσεις. Μερικές από τις μεθόδους που θα χρησιμοποιήσουμε και τις δυνατότητες που μας επιτρέπουν.
aasm_event αναφορά στα γεγονότα
aasm_enter_initial_state δίνουμε αρχική τιμή
aasm_permissible_events επιτρεπτές καταστάσεις σύμφωνα με τη μηχανή

2.3.2 Έλεγχος πηγαίου κώδικα με τη χρήση Git

Το Git είναι ένα ελεύθερο και ανοικτό λογισμικό ελέγχου του πηγαίου κώδικα μιας εφαρμογής. Σχεδιάστηκε για να διαχειρίζεται τις εκδόσεις του λογισμικού καθώς και το πλήρες ιστορικό κρατώντας πληροφορίες για οποιαδήποτε αλλαγή έχει σχέση με τα αρχεία και το συγγραφέα σε αυτά και λειτουργεί το ίδιο αποδοτικά και γρήγορα τόσο σε μικρές όσο και πολύ μεγάλες εργασίες.

Το Git είναι ένα σύνολο από μικρότερα εργαλεία που είναι γραμμένα σε C, οι υπηρεσίες που προσφέρει είναι εύκολο να διαχειριστούν από τον άνθρωπο γιατί προσφέρει μια εύχρηστη διεπαφή που αποδίδει ικανοποιητικά τις υπηρεσίες του. Δίνει μια όμορφη δομή και πέρα από τη διαδικασία ανάγκης να έχουμε την έκδοσης του λογισμικού μας προσφέρει την αίσθηση να καταλάβουμε πως έχει αναπτυχθεί το λογισμικό μας και να είναι και ευανάγνωστο από κάποιο νέο προγραμματιστή που ίσος ασχοληθεί με τη δικιά μας εφαρμογή στο μέλλον.

Τρεις από τις βασικές υπηρεσίες του Git είναι η δημιουργία Branch, Commit και Merge. Με το branch δημιουργούμε έκδοση-κόμβο στο λογισμικό μας, το commit δημιουργεί μικρές αποθήκες ιστορικού όπου οι εισάγει και προσθέτει τις αλλαγές του λογισμικού στο κεφαλικό αντίστοιχο branch, το merge κάνει σύνδεση μεταξύ δυο διαφορετικών κόμβων προσθέτει τις διαφορές τους σε ένα branch και δημιουργεί μια νέα έκδοση με αυτές.

Πέρα ότι είναι πολύ γρήγορο και αποτελεσματικό προσφέρει επιπρόσθετα πλεονεκτήματα. Κάθε κλώνος του λογισμικού από το git έχει μια πλήρες αναφορά του ιστορικού με δυνατότητα παρακολούθησης και αναθεώρησης της διαδικασίας ανάπτυξης. Δεν εξαρτάστε από την πρόσβαση στο διαδίκτυο ή κάποιο κεντρικό server. Όπως και τα περισσότερα συστήματα ελέγχου λογισμικού ο προγραμματιστής παίρνει ένα τοπικό αντίγραφο ολόκληρου του ιστορικού ανάπτυξης και τις αλλαγές που έχουν δημιουργηθεί από τις διαφορετικές εκδόσεις του το οποίο βρίσκετε μέσα στο κρυφό αρχείο .git στο φάκελο του project.

Το ιστορικό αποθηκεύεται με τέτοιο τρόπο ώστε το όνομα της συγκεκριμένης έκδοσης “commit” να εξαρτηθεί από το υπάρχον ιστορικό. Δεν είναι δυνατόν να αλλάξουν οι υπάρχουσες εκδόσεις χωρίς να φαίνεται και επίσης δίνει τη δυνατότητα στο όνομα του commit να υπογράφεται κρυπτογραφημένο . Το git προσφέρει ασφάλεια γιατί πιστοποιεί μέσο ssh ή ενός http κλειδιού τη δημοσίευση της νέας έκδοσης. Έχει τη δυνατότητα να χειριστεί μεγάλα Projects, δημιουργεί πολλά επίπεδα κατά διαδικασία της ανάπτυξης και κρατά στη κορυφή την αναβάθμιση. Η δομή της σχεδίασης, του δίνει ευελιξία ανάπτυξης και πρόσθεσης νέων εργαλείων και υπηρεσιών σε αυτό κερδίζει έτσι την ανταγωνιστικότητα με άλλα αντίστοιχα με αυτό.

Βασικές εντολές του Git

Κατά τη διάρκεια ανάπτυξης μπορούμε να κάνουμε πολλά πράγματα δημιουργία εκδόσεων ακόμα και να διαμορφώσουμε το υπάρχον ιστορικό οι όποιες ενέργειες καταγράφονται. Οι πιο σημαντικές εντολές -ενέργειες είναι οι παρακάτω.

Εγκατάσταση του λογισμικού Git

```
$ apt-get install git-core
```

Ενημέρωση του Git για τα στοιχεία του προγραμματιστή. Από το git θα μας ζητηθεί να ορίσουμε τα στοιχεία μας ονοματεπώνυμο και email ώστε να τα χρησιμοποιεί και να πιστοποιεί το κάθε χρήστη για τις ενέργειες που κάνει στα αρχεία και στις εκδόσεις.

```
$ git config --global user.name "Όνομα Επώνυμο"

$ git config --global user.email user@example.com
```

Για τη διαχείριση branch χρησιμοποιούμε τη παρακάτω εντολή για να καλέσουμε ή να δημιουργήσουμε και να μεταφερθούμε (αν δεν υπάρχει) στο branch με το αντίστοιχο όνομα.

```
$ git branch όνομα_branch
```

Πρόσθεση καταλόγου ώστε να καταγράφομε οι κινήσεις του από το Git

```
$ git add .
```

Για την κλήση - μετακίνηση μεταξύ των branch

```
$ git checkout όνομα_branch
```

Δημιουργία commit μεταξύ των branch και εισαγωγή υποχρεωτικού σχόλιου.

```
$ git commit -m 'Σχόλια'
```

Εμφάνιση των διαφορών των brach ή των commit με το αντίστοιχο όνομα

```
$ git diff όνομα
```

Σύνδεση δύο ή περισσότερων ιστορικών σε ένα branch

```
$ git merge
```

Ενημέρωση του λογισμικού για τις νέες αλλαγές (update)

```
$ git push  όνομα_branch
```

Η επιλογή της λογισμικού Git έγινε μεταξύ άλλων ελέγχου ανάπτυξης όπως το γνωστό Mercurial. Τα πλεονεκτήματά που προσφέρει το Git δεν μπορούμε να τα βρούμε ταυτόχρονα σε κάποιο αντίστοιχο. Το Mercurial είναι μονολιθικό και άκαμπτο σε αντίθεση με το Git που είναι χτισμένο σε επιμέρους κομμάτια όπως το git – apply, git – hash – object και το git – merge έτσι μπορούμε να κάνουμε οτιδήποτε. Το Git προσφέρει μεγαλύτερη ευκολία στη συνεργασία προγραμματιστών που έχουν να μοιραστούν κώδικα, βλέπουμε ξεκάθαρα που έχει επέμβει ο κάθε ένας ακόμα και σε μια γραμμή κώδικα χωρίς να χρειάζεται να βρίσκονται στο ίδιο σημείο.

Αν και δεν προτείνετε προσφέρει τη δυνατότητα να επέμβουμε σε παλιότερες εκδόσεις κρατώντας πλήρες ιστορικό ακόμα και για αυτή την ενέργεια σε περίπτωση που αυτό χρειάζεται να συμβεί στο λογισμικό μας. Το Mercurial είναι σαφώς πιο αργό και δυσλειτουργικό σε θέματα όπως διακλαδώσεις, συγχωνεύσεις και συστήματα σήμανσης δηλαδή κύριες διεργασίες που χρησιμοποιούμε σε ένα σύστημα ελέγχου λογισμικού ειδικότερα όταν εφαρμόζουμε μεθοδολογίες προγραμματισμού όπως Agile και extrem programming.

2.3.3 Πλατφόρμα εφαρμογών Heroku

Το heroku είναι μια πολυγλωσσική πλατφόρμα εφαρμογών και υπηρεσιών. Επιτρέπει στους προγραμματιστές να απαντήσουν, ιεραρχούν και να διαχειρίζονται τις εφαρμογές τους χωρίς να πρέπει να κάνουν κάτι για τον εξυπηρετητή ή το σύστημα διαχείρισης. Χρησιμοποιεί τις υπηρεσίες βάσης δεδομένων Postgres δωρεάν και υποστηρίζει ruby εφαρμογές.

Μέσα από τη πλατφόρμα heroku μπορούμε να παρακολουθήσουμε και να διαχειριστούμε πλήρως την εφαρμογή μας από το τοπικό υπολογιστή μας ανοίγοντας επικοινωνία μέσα από το τερματικό μας. Για την επικοινωνία χρειαζόμαστε να έχουμε δημιουργήσει λογαριασμό. Οι βασικότερες εντολές για την διαχείριση της εφαρμογή είναι:

Σύνδεση μέσο τερματικού από το τοπικό υπολογιστή

```
$ heroku open
```

Δημιουργία μιας θέση να τοποθετήσουμε την εφαρμογή

```
$ heroku create –stack
```

Για να βλέπουμε τη κατάσταση της εφαρμογής μας

```
$ heroku ps
```

Μπορούμε να παρακολουθούμε το logs του εξυπηρετητή ως προς την εφαρμογή μας με την εντολή.

```
$heroku logs
```

Συνδεόμαστε στο κέλυφος ruby για να πειραματιστούμε πω στην εφαρμογή μας ενώ βρίσκετε στο server.

```
$heroku run console
```

Να ανεβάσουμε την νέα έκδοση της εφαρμογής ή ένα από τις εκδόσεις μας

```
$git push heroku όνομα_brance
```

ΚΕΦΑΛΑΙΟ 3

Ανάλυση & Υλοποίηση Εφαρμογής

3.1 ΜΕΘΟΔΟΛΟΓΙΑ ΑΝΑΠΤΥΞΗΣ ΛΟΓΙΣΜΙΚΟΥ

Σε αυτό το κεφάλαιο θα αναπτύξουμε την εφαρμογή με βάση τις μεθοδολογίες και τις τεχνικές που παρουσιάσαμε στο κεφάλαιο 2. και το δεύτερο στάδιο περιέχει την εξέλιξη της εφαρμογής από την επιρροή του τρόπου εργασίας των χρηστών.

Στην περίπτωση της πτυχιακής δε χρειάστηκε να συνεργαστούν διαφορετικές ομάδες προγραμματιστών, όμως καθορίστηκαν ανεξάρτητες πτυχές του προβλήματος σε μικρότερα από την ανάλυση, την υλοποίηση και τη προσαρμογή των απαιτήσεων. Το Κέντρο Διαχείρισης Δικτύων(ΚΕΔΔ) του Τ.Ε.Ι. Κρήτης λειτουργεί πάνω από μια δεκαετία, τα μέλη του συμμετείχαν όλα αυτά τα χρόνια στην τεχνολογική ανάπτυξη και συνέβαλαν ενεργά στην επέκταση του. Αυτό έχει σαν αποτέλεσμα τα βασικά μέλη να είναι πλήρες γνώστες του υλικού εξοπλισμού και των υπηρεσιών που παρέχουν, ενώ χρησιμοποιούν ήδη ένα σύστημα truble ticket. Αυτό σε πρώτη φάση δείχνει να κάνει τη δουλειά μας πιο εύκολη ενώ στη πορεία αποδεικνύεται το αντίθετο.

Οι άνθρωποι έχουν προσαρμοστεί στο τρόπο εργασίας τους είναι γνώστες του περιβάλλοντος και είναι δύσκολο από μέρους μας να προτείνουμε ένα σύστημα που έχουμε σκεφτεί εμείς για αυτούς. Οδηγούμαστε σχεδόν αυτόματα στη μεθοδολογία Agile όχι μόνο σε προγραμματιστικό επίπεδο αλλά και ως μέθοδο ανάλυσης. Σκοπός είναι όχι μόνο να ικανοποιήσουμε τις ανάγκες των πελατών μας αλλά και να προτείνουμε τις λύσεις που εμείς μπορούμε να υλοποιήσουμε όσο το δυνατόν καλύτερα και να καταλήξουμε στο δυνατότερο εύχρηστο, υλοποιήσιμο και έγκυρο σύστημα που μπορούμε να πραγματώσουμε για την κάλυψη των αναγκών τους.

Το πρώτο βήμα είναι να έρθουμε σε επαφή με τα μέλη του Κέντρου Ελέγχου και Διαχείρισης Δικτύων και έτσι και έγινε. Οι δικές μας γνώσεις περιορίζονταν γύρω από το τρόπο διαχείρισης και έπρεπε να μάθουμε και αφού μάθουμε' τα μέλη να δουν τις ιδέες μας για τη λύση του προβλήματος να τις συζητήσουμε, να λάβουμε τις απόψεις τους και να προσδιορίσουμε τους νέους στόχους μας, να τις υλοποιήσουμε και να τις παρουσιάσουμε.

Ακολουθώντας τη πιστά τη μεθοδολογία , οργανώσαμε το περιβάλλον εργασίας και τρόπο εργασίας. Οτιδήποτε θέλουμε ή χρειαζόμαστε να φτιάξουμε το χαρακτηρίζουμε με την αγγλική φράση TODO (= να πράξω). Βοηθητικό εργαλείο ανάπτυξης και λύσεις προβλημάτων είναι τα διαγράμματα ή υλοποίηση μηχανών πεπερασμένων καταστάσεων .

Για τη οργάνωση αυτών χρησιμοποιήθηκε ο πίνακας που βλέπουμε ένα στιγμιότυπο στη παρακάτω εικόνα.

Εικόνα 5: Πίνακας Στόχων Μεθοδολογίας Agile

Κάθε χαρτάκι περιγράφει ένα μικρό στόχο-απαίτηση για λύση του προβλήματος θεωρώντας ως πελάτη τα μέλη τεχνικής υποστήριξης του κέντρου δικτύων του ΤΕΙ Κρήτης. Οι στόχοι της ανάπτυξης καθορίστηκαν σύμφωνα με τις απαιτήσεις των πελατών και τη διαπραγμάτευση αυτών ή όχι και με παράθεση προτάσεων σύμφωνα με τις γνώσεις για την ικανότητα υλοποίησης .

3.1.1 Βασικές ανάγκες συστήματος

Θα ξεκινήσουμε με τις βασικές ανάγκες που δημιουργούνται σε ένα σύστημα truble ticket που θα δημιουργήσουμε με τους πρώτους στόχους για το πως θα εξελιχθεί η εφαρμογή μας. Ανάγκη είναι να έχουμε χρήστες που δημιουργούν αιτήσεις-αναφορές του λεγομένου (ticket).

Στόχοι λοιπόν είναι να μπορούμε να κρατάμε τα στοιχεία των χρηστών (οπότε θα έχουμε τα κατάλληλα πεδία προσδιορισμού χρήστη, να υπάρχουν διαφορετικοί τύπου χρήστη ανάλογα με τις αρμοδιότητες του.

Οι αναφορές – αιτήσεις (tickets) να περιγράφουν το λόγο ύπαρξης τους, να δηλώνουν κατάσταση, να γνωρίζουμε το δημιουργό τους και να υπάρχει ένας χρήστη ως παραλήπτης. Τα tickets δημιουργούν την ανάγκη να κρατάμε ένα ιστορικό όπου καταγράφονται όλες οι ενέργειές που το αφορούν.

3.1.2Δημιουργία πινάκων μέσα από τις απαιτήσεις

Όλοι οι παραπάνω στόχοι οδηγούν στη δημιουργία των τριών πρώτων μοντέλων-πινάκων του συστήματος που περιγράφονται παρακάτω.

Ο πίνακας **User** (χρήστης) ο οποίος κρατά όλα τα απαραίτητα στοιχεία-πεδία των χρηστών για την εγγραφή τους στο σύστημα δημιουργώντας προσωπικό λογαριασμό σε αυτό, την δυνατότητα διαχείρισης του λογαριασμού (επανάκτηση κωδικού), στοιχεία επικοινωνίας με τον χρήστη (τηλέφωνο και email), πληροφορίες της οποίες χρειάζεται το σύστημα για την διαχείριση των χρηστών ως "οντότητες" σε αυτό.

Ο πίνακας **Tickets** (δελτίο) περιέχει όλα τα πεδία τα οποία σε ανάλογο συνδυασμό μπορούν να περιγράψουν ένα truble - ticket. Ανεξάρτητα από το τύπο - "είδος" ενός Truble Ticket έχει κάποιες συγκεκριμένες απαιτήσεις για τα πεδία του.

Το πεδίο name παίρνει ως τιμή το όνομα του ticket και στην ουσία δηλώνει την κατηγορία του.

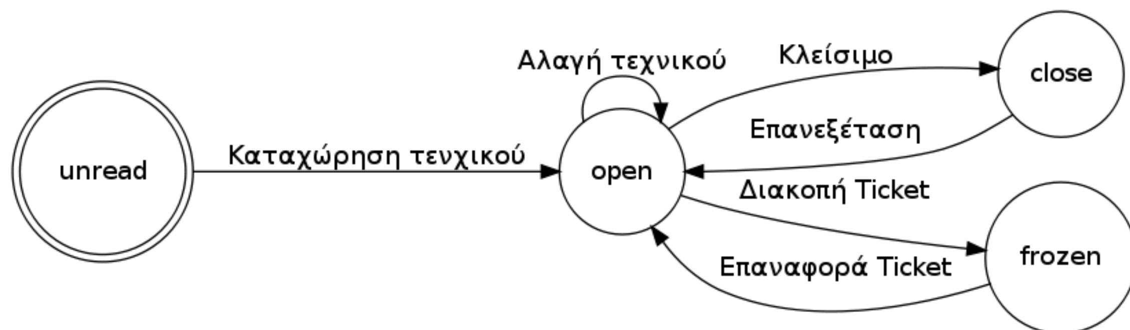
Το πεδίο element συγκεκριμένα για το Noc το σημείο οπού καταγράφονται τα eclass-email (για ticket Eclass), socket (για ticket για ticket τηλεφώνου και internet), ip (για ticket ip- η υπάρχον ip), hardware (είδος συσκευής π.χ Ups, server) το πεδίο element δεν επιτρέπετε να είναι null.

Το πεδίο advice καταγράφει τα στοιχεία (pcname για network – ticket), phone (ticket τηλεφώνου) , όνομα/μοντέλο /εταιρεία (hardware – ticket), ίσως για μερικά truble ticket όπως Eclass να μην χρειάζεται αυτό το πεδίο όποτε του δίνουμε τη δυνατότητα στη βάση δεδομένων να είναι null.

Το πεδίο description είναι τύπου text, ο χρήστης κατά την δημιουργία ενός truble – ticket υποχρεούται να δώσει μια περιγραφή του προβλήματος με αυτό τον τρόπο οι τεχνικοί έχουν καλύτερή εικόνα του προβλήματος που καλού ντε να αντιμετωπίσουν.

Το πεδίο name δίνει το χαρακτηρισμό του truble ticket (NetworkTicket, PhoneTicket, Eclass, New Connection και οποιαδήποτε είδος Truble – Ticket μπορεί να προκύψει στο μέλλον). Το πεδίο name παίρνει τη τιμή της κατηγορία.

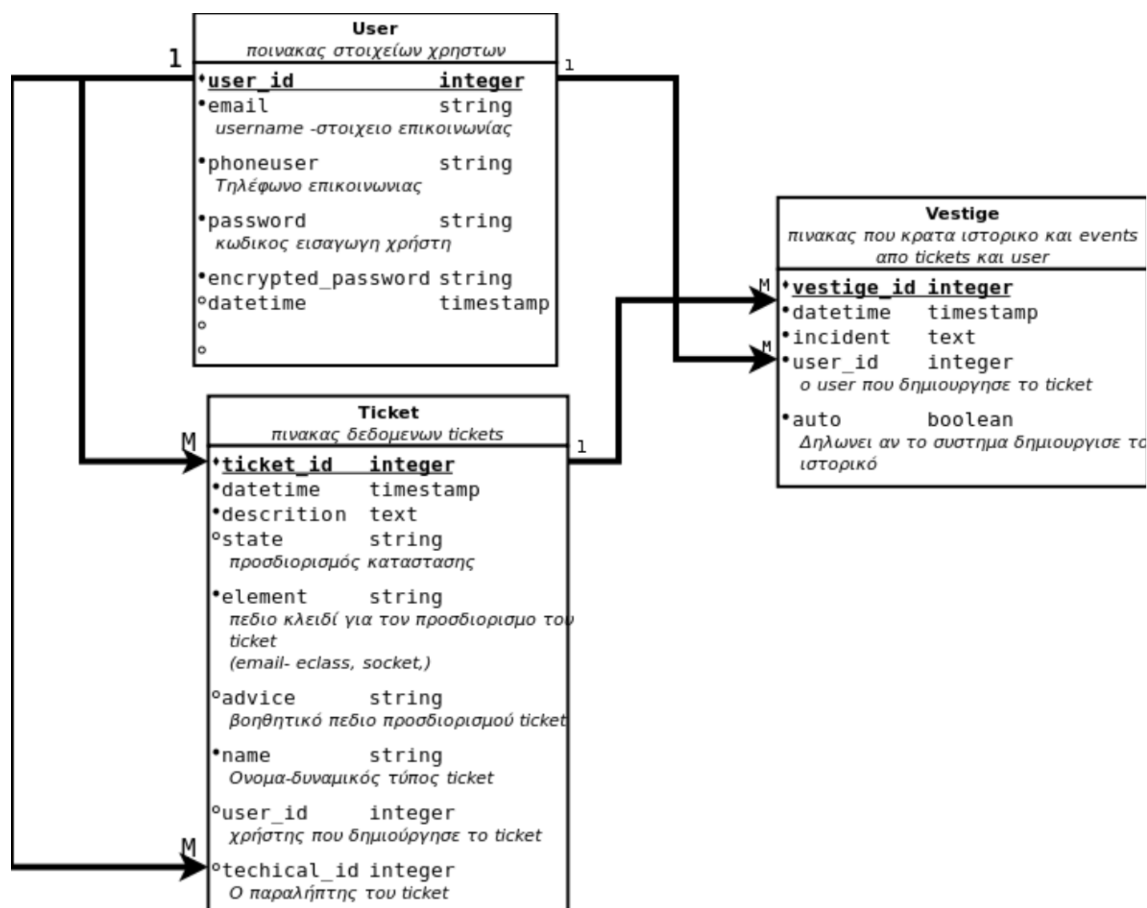
Το πεδίο state είναι πολύ σημαντικό για το σύστημα. Είναι το πεδίο όπου θα χρησιμοποιήσουμε για το FSM (Finite State Machine – Μηχανή Πεπερασμένων Καταστάσεων), οι τιμές είναι συγκεκριμένες unread, open, frozen, close και οι μεταβάσεις τις περιγράφουμε στο παρακάτω σχήμα.



Σχήμα 4:FSM καταστάσεις ticket

Τα πεδίο ticket_id είναι το κλειδί του πίνακά ticket

Το πεδίο datetimeT χωρίζετε σε δυο πεδία created_at και updated_at είναι τύπου timestamps και μέσο αυτών να κρατάμε πληροφορίες για το πότε δημιουργήθηκε , άλλαξε (state, name)ένα ticket .



Σχήμα 5: Οι Βασικοί Πίνακες και οι Σχέσεις τους

Ο πίνακας **Vestige** (ίχνη) ο οποίος είναι για το ιστορικό που θέλουμε να κρατάμε στην εφαρμογή μας. Κύριο πεδίο του είναι το incident τύπου text όπου καταγράφονται όλα τα κείμενα – ενέργειες των Tickets που είναι χρήσιμα για την εφαρμογή μας, το user_id ανήκει στο χρήστη που δημιούργησε το σχόλια στο πεδίο incident ενώ έχουμε και το πεδίο auto για να δηλώνουμε αν η δημιουργία πεδίου είναι από το σύστημα ή όχι. Τα πεδία vestige_id, datetimes είναι βοηθητικά στο σύστημα.

Οι χρήστες έχουν πολλά ticket που τους “ανήκουν” αυτά που έχουν δημιουργήσει και έχουν σχέση ένα προς πολλά και δεν αλλάζει η αντιστοιχία. Το κάθε ticket ανήκει σε έναν ή κανένα τεχνικό και το πεδίο techical_id είναι μεταβλητό. Στο κάθε ένα ticket έχει πολλές καταγραφές ιστορικού (vestige) που “ανήκουν” σε αυτό. Ο κάθε χρήστης (user) δημιουργεί και του “ανήκουν” πολλές καταγραφές έτσι έχουμε και μια δεύτερη σχέση ένα προς πολλά μεταξύ χρηστών και ιστορικού.

3.2 ΥΛΟΠΟΙΗΣΗ ΤΟΥ ΕΡΓΟΥ Α' ΣΤΑΔΙΟ

Στην υλοποίηση της εφαρμογής θα την παρουσιάσουμε σε δύο στάδια. Το πρώτο στάδιο είναι η γενικευμένη υλοποίηση του truble ticket συστήματος και περιέχει τους αρχικούς στόχους και τη δημιουργία των κύριων μοντέλων του συστήματος. Η εφαρμογή θα εγκατασταθεί στο server και θα ελεγχθεί, και ύστερα θα θέσουμε νέους στόχους.

3.2.1Εγκατάσταση Ruby on rails

Πρώτο βήμα εγκατάσταση RVM για να έχουμε τη δυνατότητα να διαχειριστούμε διαφορετικές εκδόσεις της ruby σε διαφορετικά τερματικά ταυτόχρονα και μας επιτρέπει να διαχειριστούμε μια νέα έκδοση της ruby πάνω στο έργο μας ώστε να τη δοκιμάσουμε αφήνοντας ανέπαφο το αρχικό.

```
$ curl -L get.rvm.io | bash -s stable
```

Δεύτερο βήμα εγκατάσταση της γλώσσας ruby

```
$ sudo apt-get install ruby1.9.1
```

Τρίτο βήμα εγκατάσταση βιβλιοθήκη ruby on rails

```
$ gem install rails
```

3.2.2 Δημιουργία της Εφαρμογής μας

Ανοίγουμε το φάκελο όπου θέλουμε να δημιουργήσουμε το Project μας και δίνουμε την εντολή από το τερματικό μας:

```
$ rails new peper
```

Αυτή η εντολή δημιουργεί μια εφαρμογή rails, με το όνομα peper Αυτόματα δημιουργήθηκε η δομή της εφαρμογής που φαίνεται παρακάτω.

```
|-- app
| |-- assets
| |-- images
| |-- javascripts
| `-- stylesheets
| |-- controllers
| |-- helpers
| |-- mailers
| |-- models
| `-- views
| `-- layouts
|-- config
| |-- environments
| |-- initializers
| `-- locales
|-- db
|-- doc
|-- lib
| `-- tasks
|-- log
|-- public
|-- script
|-- test
| |-- fixtures
| |-- functional
| |-- integration
| |-- performance
| `-- unit
```

```
|-- tmp
| |-- cache
| |-- pids
| |-- sessions
| `-- sockets
|`-- vendor
| |-- assets
| `-- stylesheets
| -- plugins
|Gemifile
```

app: Σε αυτό το κατάλογο βρίσκετε ο κώδικας που φτιάχνουμε για τη δίκια μας εφαρμογή

app/assets: Περιέχει τιμοκαταλόγους όπως stylesheets για αρχεία css, javascripts για αρχεία .js και το κατάλογο με τις εικόνες που χρησιμοποιούμε στην εφαρμογή.

app/controllers: περιέχει όλους τους controllers της εφαρμογής οι οποίοι είναι οραματιζόμενοι με το όνομα που έχουν στο model και view συν τη κατάληξη contoller.rb

app/models: Βρίσκονται τα αρχεία που περιγράφουν το κομμάτι model της εφαρμογής για το οποίο μιλήσαμε παραπάνω. Τα αρχεία έχουν όνομα μοντέλου συν τη κατάληξη rb. Το μοντέλο ανήκει στο ActiveRecord. Μέσα στα αρχεία το δείχνουμε με Class όνομα μοντέλου < ActiveRecord::Base

app/views: Περιέχει καταλόγους με τα ονόματα των αντίστοιχων controllers που "ακούνε", οι κατάλογοι περιέχουν τα αρχεία

app/views/layouts: Περιέχει αρχεία layouts τα οποία χρησιμοποιούνται ως πρότυπα για την διάταξη - εμφάνιση της web εφαρμογής στο brower και στα οποία ανταποκρίνονται τα views. Τα αρχεία έχουν κατάληξη html.erb

app/helpers: Τα helpers είναι τα αρχεία όπου γράφουμε κώδικα που επαναλαμβάνετε και τον καλούμε όπου αυτός χρειάζεται με την αντίστοιχη μέθοδο που έχουμε δημιουργήσει. Τα αρχεία έχουν όνομα_helper.rb .

Config: Είναι κατάλογος που τα αρχεία που περιέχει είναι σημαντικά για το περιβάλλον της rails, τη δρομολόγηση των ιστοσελίδων και τη βάση δεδομένων

db: Περιέχει τα σχετικά αρχεία με τη βάση δεδομένων και ένα από το τα πιο σημαντικά το schema.rb που είναι η χαρτογράφηση της βάσης δεδομένων.

Doc:Είναι ο κατάλογος όπου θα πρέπει να αποθηκεύεται η εφαρμογή rails όταν "τρέχει"

lib: Ειδικές βιβλιοθήκες που φορτώνονται κατά την εκκίνηση της εφαρμογής και δεν ανήκουν σε κάποιο συγκεκριμένο controller, model, view ή helper.

Public: Φάκελος με αρχεία που απευθύνεται στο web-brower. Τα αρχεία είναι δημόσια και είναι αρχεία δεν έχουν οριστεί που θα βρίσκονται μέσα από την εφαρμογή όπως για παράδειγμα σελίδες λάθους.

Script: βοηθητικά scripts για την παραγωγή αυτοματισμών και generators της εφαρμογής

Test: Εδώ δημιουργούμε τα test της εφαρμογής. Τεστ με τη μορφή σεναρίων για να μπορούμε να τα υλοποιήσουμε στην εφαρμογή

vendor: Περιέχει εξωτερικές βιβλιοθήκες που χρησιμοποιεί η εφαρμογή και φορτώνουν κατα την εκκίνηση.

Gemfile:Ένα από τα πιο σημαντικά αρχεία, εδώ γίνετε η εγκατάσταση βιβλιοθηκών ή πακέτα εφαρμογών τα λεγόμενα gems της ruby.

#TODO: Χρειαζόμαστε να παρακολουθήσουμε το project μας οπότε βάζουμε το Git στο κατάλογο peper. Μέσα στο κατάλογο peper δίνουμε την εντολή

```
$ git init
```

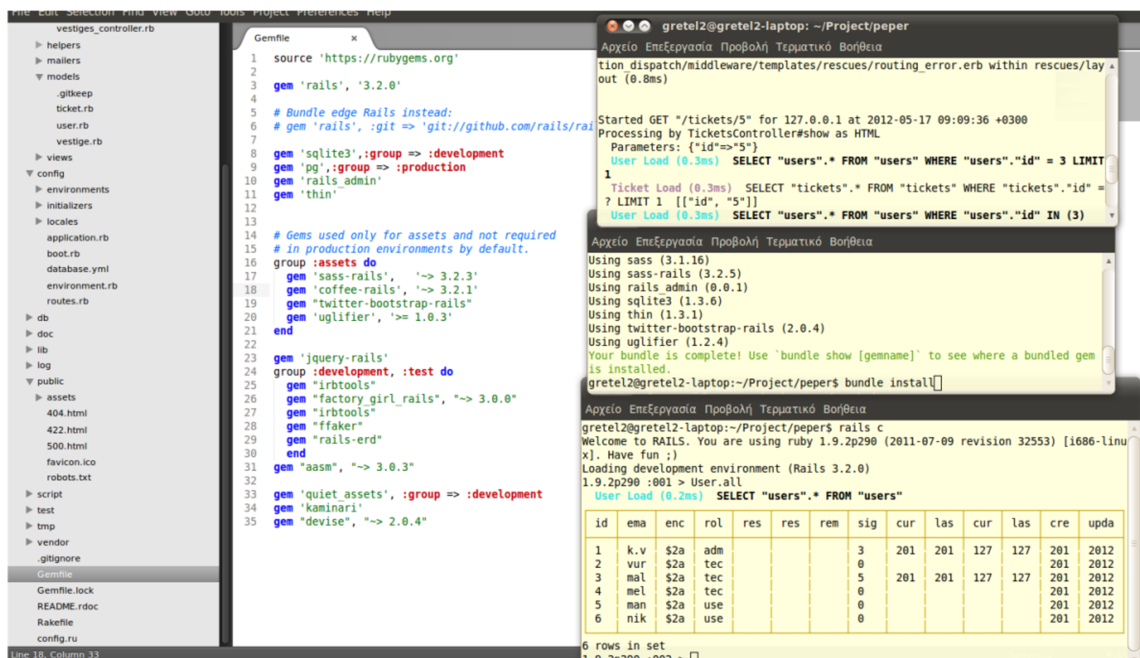
Και κάνουμε το πρώτο Commit ορίζοντας το πρώτο ιστορικό μας.

```
$ git commit -m 'Begin'
```

3.2.3 Περιβάλλον Εργασίας της Ανάπτυξης

Σημαντική λεπτομέρεια αποτελεί το περιβάλλον εργασίας μιας και δεν έχουμε αναφερθεί ακόμα σε αυτό' εδώ είναι ένα καλό σημείο να πούμε για τρόπο εργασίας όταν δημιουργούμε κώδικα. Απαραίτητα εργαλεία είναι ένα πρόγραμμα επεξεργασίας πηγαίου κώδικα και το τερματικό.

Στο τερματικό τρέχουμε τρία πράγματα το rails server όπου βλέπουμε τα αποτελέσματα της εφαρμογής στο τοπικό δίκτυο. Το rails console διεπαφή που δίνουμε εντολές στην εφαρμογή μας χρήσιμο για να δοκιμάσουμε το κώδικα πως αντιδρά. Ένα απλό τερματικό στο φάκελο εργασίας για να δώσουμε εντολές. Το περιβάλλον εργασίας διαμορφώνετε όπως βλέπουμε στη παρακάτω εικόνα.



Εικόνα 6: Περιβάλλον Εργασίας

3.2.4 Υλοποίηση των αρχικών στόχων

#TODO: Δημιουργία controller για την κεντρική σελίδα home/page.

Χρησιμοποιούμε το κατασκευαστή (generator)της RoR δίνουμε εντολή να περιέχει ο controller τη μέθοδο index που είναι το action της αρχικής σελίδας μας.

```
$ rails generate controller home index
```

Το επόμενο βήμα είναι να κάνουμε Restfull το αντικείμενο. Στο αρχείο peper/confing/routes.rb ορίζουμε ως διαδρομή στο μητρικό URL

```
Peper::Application.routes.draw do

root :to => "home#index"

get "home/index"
```

#**TODO** Προσθέτουμε μερικές βιβλιοθήκες (gems) για το desing της εφαρμογής.
Στο αρχείο app/Gemfile γράφουμε το όνομα τις βιβλιοθήκης.

```
gem "twitter-bootstrap-rails"
```

κάθε φορά που προσθέτουμε μια βιβλιοθήκη δίνουμε την εντολή

```
$ bundle install
```

για να κατεβεί και να εγκατασταθεί στα αρχεία του project μας.
Κάνουμε εγκατάσταση των CSS του bootstrap στην εφαρμογή μας

```
$ rails g bootstrap:install
```

Επιλογή του κύριου layout στην εφαρμογή μας που θα διαμορφώσουμε κατάλληλα

```
$rails g bootstrap:layout application fixed
```

Η επιλογή να χρησιμοποιήσουμε τα css του twitter έγινε γιατί οι πελάτες μας χρειάζονται διαδραστικά αντικείμενα στην εφαρμογή που θα προσεγγίζουν με διαφορετική εμφάνιση τις καταστάσεις και τις πληροφορίες και θα διαχωρίζονται πληροφορίες όπως ημερομηνίες, χρώματα καταστάσεων ticket. Ο διαχωρισμός των εργασιών , οι πίνακες εμφάνισης των εργασιών και λειτουργική απεικόνιση των διαδραστικών αντικειμένων κατά στοίχιση ώστε να είναι εύχρηστη η διεπαφή μας.

#**TODO** Δημιουργία μοντέλου για τις ανάγκες των χρηστών

Μια εύχρηστη βιβλιοθήκη της ruby on rails είναι η βιβλιοθήκη devise . Καλύπτει τις ανάγκες των χρηστών και παρέχει τις δυνατότητες που χρειαζόμαστε για το μοντέλο μας. Προσθέτουμε το παρακάτω στο αρχείο peper/Gemfile.rb και επαναλαμβάνουμε τη διαδικασία εγκατάστασης βιβλιοθήκης.

```
gem "devise", "~> 2.0.4"
```

Κάνουμε εγκατάσταση για την κατασκευή των μεθόδων μέσα στην εφαρμογή μας.

```
$rails generate devise:install
```

Και επιλέγουμε το μοντέλο – κλάση όπου θα το εφαρμόσουμε δηλαδή στο User

```
$rails generate devise USER
```

Στο αρχείο app/db/migrate/hash.create.user.rb προσθέτουμε τα πεδία που χρηματιζόμαστε για το μοντέλο μας .

#**TODO**: Δημιουργία μοντέλου Ticket

```
$rails generate Ticket category:string descrition:text element:
string advice:string user_id:integer technical_id:integer
state:string
```

Φτιάχνουμε το μοντέλο Ticket με τη βοήθεια κατασκευαστή της RoR, μας παρέχει προεπιλεγμένα actions τα οποί θα αφαιρέσουμε στη συνέχεια. Μετά το όνομα του μοντέλου ορίσουμε τα πεδία – αντικείμενα του πίνακα με τα αντίστοιχους τύπους.

#TODO: Δημιουργία μοντέλου Vestige

```
$rails generate scaffold Vestige incident:text user_id:integer
auto:boolean state:string
```

Ομοίως με το μοντέλο Ticket

#**TODO**: Σχέσεις μεταξύ των μοντέλων.

Οι σχέσεις των μοντέλων μέσα στην εφαρμογή ορίζονται στα αρχεία των μοντέλων. Τα αρχεία έχουν το όνομα το μοντέλου και κατάληξη rb. Τα δημιουργούμε και γράφουμε τον αντίστοιχο κώδικα σε κάθε ένα.

Στο μοντέλο Ticket

```
class Ticket < ActiveRecord::Base

belongs_to :user

belongs_to :technical, :class_name => "User"

has_many :vestiges

end
```

Στο μοντέλο του User

```
class User < ActiveRecord::Base

has_many :tickets

end
```

Στο μοντέλο που κρατά το ιστορικό της εφαρμογής

```
class Vestige < ActiveRecord::Base

belongs_to :ticket

belongs_to :user

end
```

3.2.5 Υλοποίηση της συνεδρίας των χρηστών στο σύστημα

#TODO: Δημιουργία περιβάλλον με όνομα profile για τον απλό χρήστη

```
$rails g controller profile
```

Ομοίως κάνουμε το αντικείμενο Restfull

#TODO: Δημιουργία περιβάλλοντος εργασίας τεχνικών με όνομα todo

```
$rails g controller todo
```

Ομοίως κάνουμε το αντικείμενο Restfull

#TODO: Δημιουργία links devise στο layout

Οι χρήστες του συστήματος ανεξαρτήτου την ιδιότητα τους οπουδήποτε μέσα στην ιστοσελίδα να έχουν πρόσβαση στα links εισόδου και εξόδου απο τις συνεδρίες του συστήματος. Γι' αυτό μιας και δεν θέλουμε να επαναλαμβανόμαστε δημιουργούμε ένα partial με όνομα _devise.htm.erb

```
<div class="well sidebar-nav">

<% if user_signed_in? %>

Είσοδος ως <%= current_user.email %>.</br>

<%= link_to "Logout", destroy_user_session_path, :method =>
:delete %>

<p>

<%= link_to "My Tickets", profile_path(current_user),
:class => 'btn' %>

</p><% else %>

<%= link_to "Sign up", new_user_registration_path %> or

<%= link_to "Sign in", new_user_session_path %>
```

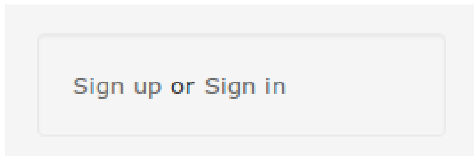
```
<% end %>

</div>
```

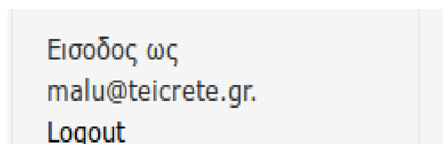
Καλούμε στο layout το partial με τον εξής τρόπο.

```
<div class="well sidebar-nav">

<%= render :partial => "devise" %>
```



Εικόνα 7: Links Εισόδου



Εικόνα 8: Links Εξόδου

Έτσι έχουμε δεξιά στην οθόνη τις παραπάνω εικόνες περιπτώσεων των links όπου και αν είμαστε μέσα στην εφαρμογή.

#TODO :Κατά την είσοδο των Χρηστών είσοδος στο αντίστοιχο περιβάλλον.
Για το σκοπό από τη βιβλιοθήκη devise αυτό χρησιμοποιούμε τη μέθοδο after_sing_in βάζοντας τη κατάλληλη συνθήκη στο αρχείο application_controller.rb

```
def after_sign_in_path_for(resource)

if current_user.is_user?

profile_path(current_user)

else

todo_path(current_user)

end

end
```

Οι διαφορετικοί τύποι χρηστών όπως προαναφέραμε έχουν διαφορετικές διεργασίες και περιβάλλοντα, έτσι κατά την είσοδο του πρέπει να διαχωρίσουμε σε ποιο περιβάλλον ανήκουν και ύστερα να χτίσουμε τις αρμόδιες διεργασίες που απαιτούνται και θα μας ζητήσουν οι πελάτες μας.

3.2.6 Υλοποίηση διαχείριση tickets από τους χρήστες

#TODO: Δημιουργία Ticket από χρήστη
Ένα ticket θα δημιουργηθεί μέσο του μοντέλου χρήστη όποτε προσθέτουμε στο μοντέλο του χρήστη τη μέθοδο create_ticket,

```
def create_ticket(*params)

collection = self.tickets.create(params)

collection.first

end
```

στο controller ticket βάζω τις μεθόδους new και create και στο view φάκελο tickets δημιουργώ τα αρχεία _form.html.erb όπου φτιάχνω τις φόρμες εισαγωγής στοιχείων, new.html.erb που επιστέφει τις τιμές για την δημιουργία ενός στοιχείου.

Και τοποθετώ ένα link ώστε να έχει προσύμβαση ο χρήστης

#TODO: Ο χρήστης να μπορεί να έχει πρόσβαση στα tickets που έχει ανοίξει – δημιουργήσει.

```
class ProfileController < ApplicationController

def show

@user = User.find(params[:id])

@tickets = @user.tickets

respond_with(@user)

end

end
```

#TODO: Οι χρήστες με την ιδιότητα tech και admin πρέπει να έχουν πρόσβασή σε όλα tickets.

Δημιουργώ το αρχείο view/todos/_tickets.html.erb

#TODO: Δημιουργία μηχανής πεπερασμένων καταστάσεων για το πεδίο state του μοντέλου ticket.

Η βιβλιοθήκη που θα χρησιμοποιήσουμε είναι η AASM αρχικά τη προσθέτουμε στο Gemfile και σύμφωνα με το σχήμα FSM στο αρχείο ticket. Στο μοντέλο μας προσθέτω το παρακάτω κώδικα.

```
class Ticket < ActiveRecord::Base

include AASM

aasm_column :state

aasm_initial_state :unread #αρχκική τιμή

aasm_state :active
```

```
aasm_state :frozen

aasm_state :close

aasm_state :unread

aasm_event :do_open do

transitions :to => :active, :from => [:unread, :close,
:frozen, :active]

end

aasm_event :do_close do

transitions :to => :close, :from => [:active, :unread]

end

aasm_event :do_frozen do

transitions :to => :frozen, :from =>[:active]

end
```

#TODO: Τα tickets τα αναλαμβάνει ένας τεχνικός ο admin μπορεί να κάνει αλλαγή τεχνικού σε ένα ticket. Δημιουργία μεθόδου ανάθεσης ticket σε τεχνικό από τον ίδιο και αλλαγή ανάθεσης από τον admin.
Ορισμός της μεθόδου assing στο μοντέλο Ticket

```
class Ticket < ActiveRecord::Base


def assign_to(user, by)

    if self.can_do_event?(:do_close)

         self.technical = user

         self.do_open!

         return true

    else

         return false

end
```

Στο controller του ticket

```
class TicketsController < ApplicationController



def assign

    if current_user.is_admin?

        @tech = User.find(params[:technical_id])

        @ticket = Ticket.find(params[:ticket_id])

        @ticket.assign_to(@tech,current_user)

    end


    if current_user.is_tech?

        @ticket = Ticket.find(params[:ticket_id])

        @ticket.assign_to(current_user,current_user)

    end


    redirect_to ticket_url(params[:ticket_id])

end
```

#TODO: Δημιουργία στοιχείων Vestige που ανήκουν σε στοιχείο ticket.

Όμοια και εδώ ένα στοιχείο vestiges θα υπάρχει εφόσον υπάρχει και το ticket στο οποίο ανήκει άρα θα δημιουργούμε το ιστορικό μέσο του ticket.

```
class VestigesController < ApplicationController


def create
```

```
    if current_user.is_user?
      # Βρίσκουμε το ticket σύμφωνα με το user που
      # το δημιούργησε
      @ticket =     current_user.tickets.\
        find_by_id(params['ticket_id'])
    else
      @ticket = Ticket.find_by_id(params['ticket_id'])
    end

    # Δημιουργία πεδίου vestiges
    @vestige Controller =
    @ticket.vestiges.create(params[:vestige])
    @vestige.user = current_user #
    #πεδία εξαρτώμενα από άλλα μοντέλα
    @vestige.state = @ticket.state
    @vestige.auto = false #
    @vestige.save
  end

  end
```

#TODO: Διαμόρφωση view για την εισαγωγή σχολίων στο ticket. Σχόλιο από ένα χρήστη θέλουμε όταν ανοίγει να δεί ένα ticket έτσι η φόρμα εισαγωγής θα είναι στο show του ticket.

Δηλώνουμε τη διαδρομή vestiges και ότι ακολουθούμε μόνο τη μέθοδο create

```
  class VestigesController < ApplicationController

  def create
    if current_user.is_user?
      @ticket =
      current_user.tickets.find_by_id(params['ticket_id'])
    else
      @ticket = Ticket.find_by_id(params['ticket_id'])
    end

    @vestige = @ticket.vestiges.create(params[:vestige])
    @vestige.user = current_user
    @vestige.state = @ticket.state
    @vestige.auto = false
    @vestige.save
  end

  end

  <%= form_for [@ticket, @ticket.vestiges.build], :method=>
  :post, :html => { :class => 'well' } do |f| %>

  <%= f.text_area :incident, :class => 'text_area' %>
```

```
Peper::Application.routes.draw do

resources :vestiges,:only => [:create]

end
```

Παρακάτω φαίνεται η φόρμα εισαγωγής και η προβολή του ιστορικού σε κάθε ticket.

Πληροφορίες

Ιστορικό	Χρήστης	Ημερομηνία	Κατάσταση	Αυτόματο
Αυτός ο αριθμός τηλεφώνου έχει μπει σε προσωρινή αποσνδεση	malu@teicrete.gr	Mon, 30 Apr 2012 12:46:12 +0000	unread	ΟΧΙ

Δώσε πληροφορίες

Χειαζετε αλλαγή τ

Δημιουργία

Εικόνα 9: Εμφάνιση Ιστορικού

#TODO: Δυνατότητα αλλαγή κατάστασης του πεδίου state του πίνακα ticket από τους χρήστες με ιδιότητα τεχνικού και admin (workes) από το περιβάλλον και αποθήκευση της ενέργειας στο ιστορικό.

```
class TicketsController < ApplicationController


def change_state

unless current_user.is_user?

@ticket = Ticket.find(params[:ticket_id])

event = params[:event].to_sym

if
@ticket.aasm_permissible_events_for_current_state.include?
event
```

```
@ticket.send event

@ticket.save

@ticket.vestiges.create incident:"Change state
to"+@ticket.state, auto: false,

user: current_user, state: @ticket.state

end

Peper::Application.routes.draw

--> get :change_state
```

Οι πελάτες μας κατά τη διάρκεια της εργασίας τους θέλουν να έχει το τεχνικό προσωπικό την δυνατότητα να χαρακτηρίζει το ticket ανάλογα στάδιο επίλυσης του ticket να χαρακτηρίζει τις καταστάσεις που έχουμε ορίσει.

3.2.7 Εγκατάσταση σε Server και έλεγχος καλής λειτουργίας

Η εφαρμογή έχει καλύψει τις πιο βασικές απαιτήσεις και μπορούμε να την ανεβάσουμε στο server και η επιλογές οδήγησαν στη πλατφόρμα δικτυακών εφαρμογών heroku το όποιο χρησιμοποιεί τη βάση δεδομένων postgres. Για την καλή λειτουργία του site χρειάστηκε να γίνουν ορισμένα επιπρόσθετα βήματα

#TODO: Να γίνουν precompile τα assets πριν εγκαταστήσουμε την εφαρμογή στο server.

```
$ rake assets:precompile
```

#TODO: Δήλωση ότι στη παραγωγή η εφαρμογή θέλουμε να επικοινωνεί με τη postgres βάση δεδομένων.

```
gem 'pg',:group => :production
```

#TODO: Ανεβάζουμε την εφαρμογή στο server

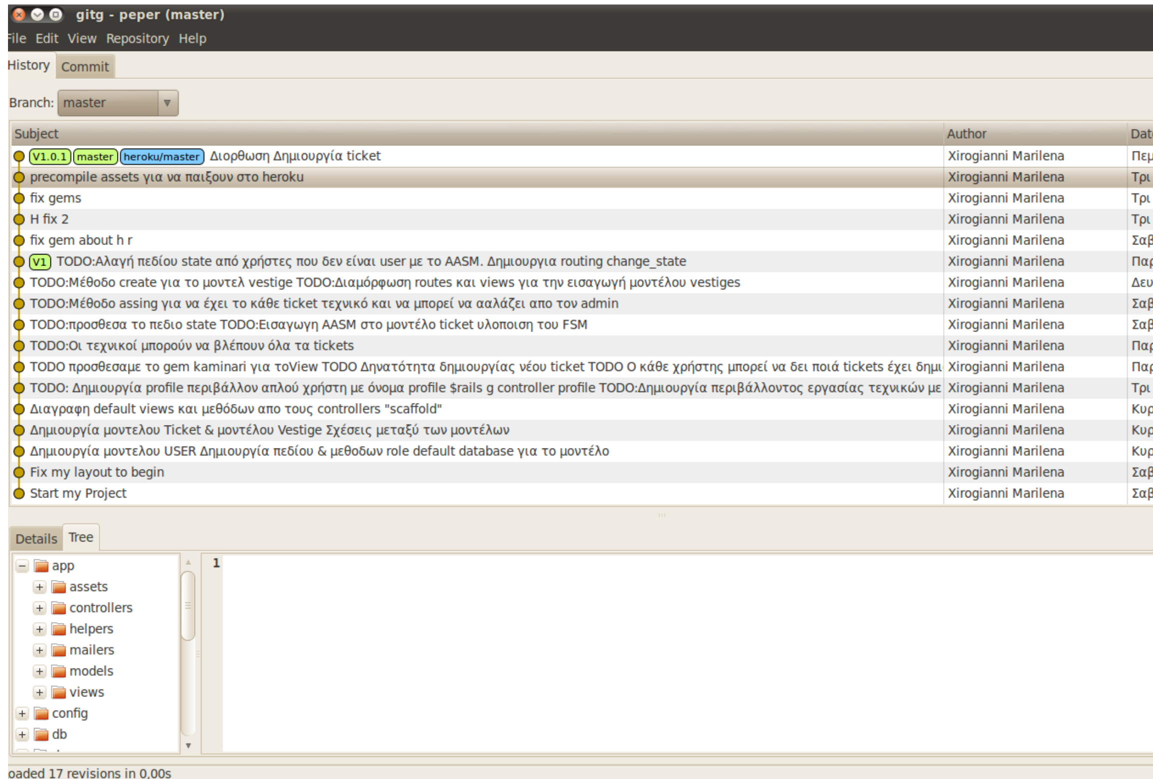
Από το τερματικό μας ανοίγουμε την επικοινωνία μας με το heroku για να συνδεθούμε

```
$ heroku open
```

Την αναβάθμιση στο σύστημα μας την κάνουμε πολύ εύκολα χρησιμοποιώντας τη βοήθεια του Git. Κάθε φορά που κάνουμε αλλαγές στο κώδικα τις προσθέτουμε στο branch κάνοντας τη διαδικασία “commit” έτσι δίνοντας την εντολή:

```
$ git push heroku master
```

Κάνουμε αναβάθμιση στην εφαρμογή μας και περνάμε όλες οι αλλαγές την εφαρμογή στο server. Εδώ με τη βοήθεια της γραφικής αναπαράστασης του Git παρατηρούμε τα commit και τα Brance(εκδόσεις) της εφαρμογής. Μπορούμε να δούμε το δέντρο των καταλόγων της εφαρμογής και τις διαφορές των αρχείων στα ενδιάμεσα commit ή branch , καθώς και σε ποια έκδοση βρισκόμαστε.



Εικόνα 10: Γραφικό Περιβάλλον Εκδόσεων Α΄ Στάδιο

#TODO: Εισαγωγή βάσης δεδομένων στο heroku

```
$heroku addons:add heroku-postgresql:dev
```

#TODO: διαδικασία backup της βάσης δεδομένων.

```
$heroku addons:add pgbackups:auto-week
```

3.3 ΥΛΟΠΟΙΗΣΗ ΤΟΥ ΕΡΓΟΥ ΣΕ Β' ΣΤΑΔΙΟ

Μέχρι τώρα έχουμε καλύψει τις γενικές ανάγκες του συστήματος, οι ιδιαιτερότητες που συναντάμε για τη διαχείριση των εργασιών. Ο τρόπος εργασίας στο κέντρο καθορίζει τη μορφή της διεπαφής. Οι χρήστες σε οποιαδήποτε κατηγορία και αν ανήκουν αλληλεπιδρούν με το σύστημα και κάνουν περιήγηση μέσα σε αυτό σύμφωνα τις διεργασίες που τους αναλογούν. Σε αυτό το σημείο θα παρουσιάσουμε τις διεπαφές με μια μικρή περιγραφή για την λειτουργία τους.

3.3.1 Υλοποίηση νέων διεργασιών ιστορικού ticket

#TODO: Δυνατότητα δημιουργία ιστορικού όταν δημιουργούμε ένα ticket, όταν γίνετε αντιστοιχία σε ένα τεχνικό και όταν αλλάζει κατάσταση.
Όταν δημιουργηθεί ένα ticketafter_create do |t|

```
self.vestiges.create incident: "Ticket created", auto:
true,

user: t.user, state: t.state

end
```

στη μέθοδο assign, όπου το ticket γίνετε υλικό εργασίας του αρμόδιου τεχνικού.

```
@ticket.vestiges.create incident: "Ticket assigned to
#{@ticket.technical_id}", auto: true,

user: current_user, state: @ticket.state
```

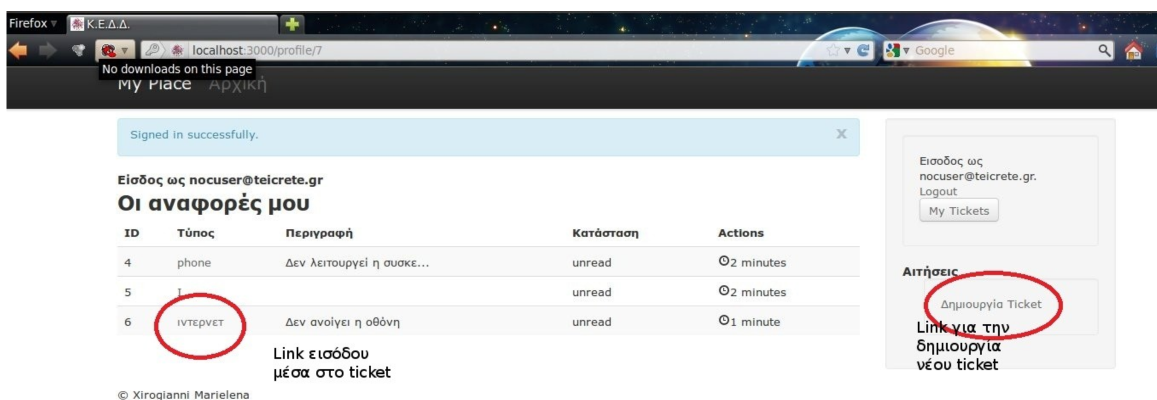
στη μέθοδο change_state ,οπου το ticket αλάζει κατάσταση.

```
@ticket.vestiges.create incident:"Change state
to"+@ticket.state, auto: false,

user: current_user, state: @ticket.state
```

3.3.2 Διαμόρφωση διεπαφής σύμφωνα με τα σενάρια χρήσης

#TODO: Διαμόρφωση της διεπαφής, είσοδος χρήστη με ιδιότητα user
Εδώ βλέπουμε το περιβάλλον εισόδου του χρήστη με ιδιότητα user όπου αριστερά παρουσιάζονται τα αιτήματα που έχει δημιουργήσει και δεξιά οι επιλογές.

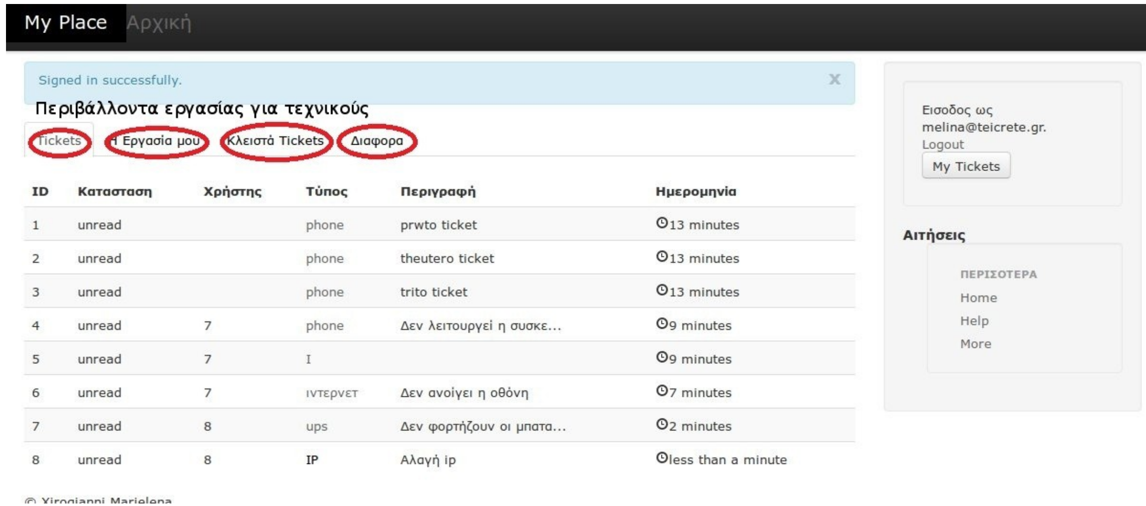


Εικόνα 11: Περιβάλλον profile

Ο χρήστης όταν θέλει να δει περισσότερες λεπτομερές για το κάθε ticket, μπαίνει στο link που βρίσκετε αριστερά στο τύπο του ticket ενώ στο δεξιά link μπορεί να δημιουργήσει νέο.

#TODO: Διαμόρφωση της διεπαφής, είσοδος χρήστη με ιδιότητα tech (τεχνικού).

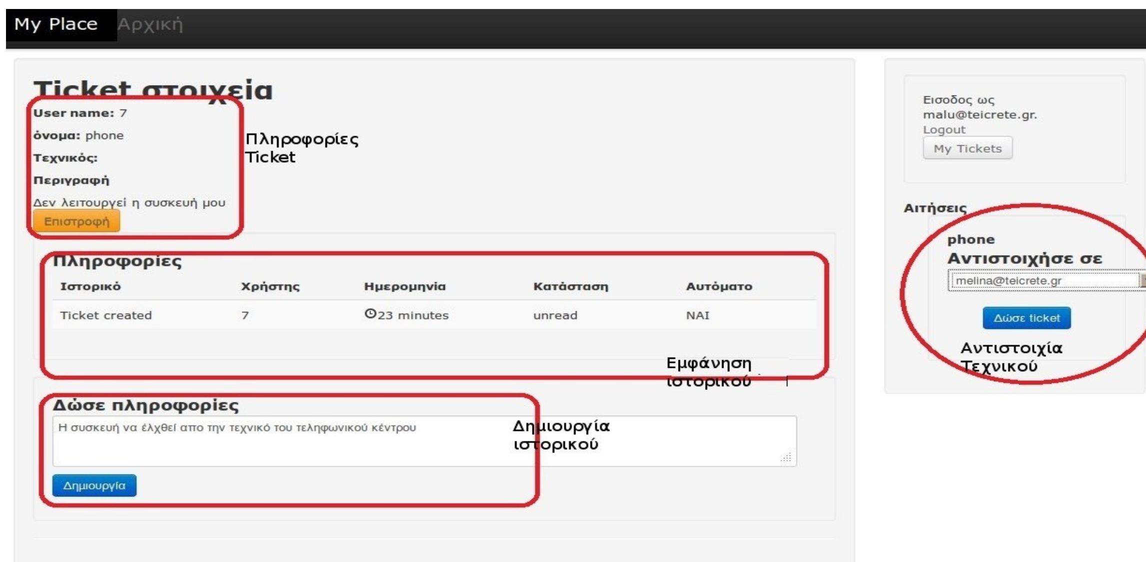
Οι χρήστες με ιδιότητα τεχνικού έχουν παραπάνω αρμοδιότητες και υποχρεώσεις προς το σύστημα. Η διεπαφή τους παρέχει διαχωρισμό των εργασιών και τους φέρνει σε διαφορετικά περιβάλλοντα για να δουν τα ticket που έχουν δημιουργήσει(ομοίως με τους απλούς χρήστες), τα ticket που χρειάζονται τεχνικό, τα ticket που έχουν αναλάβει, τα κλειστά ticket και άλλη διαχειριστή εργασιών.



Εικόνα 12: Περιβάλλον Εργασίας Workers

#TODO: Διαμόρφωση της διεπαφής ,είσοδος χρήστη με ιδιότητα admin (τεχνικού).

Η εμφάνιση των tickets είναι όμοια σε όλα τα περιβάλλοντα και αλλάζουμε μόνο τις δυνατές διεργασίες με την ιδιότητα του χρήστη, Παρακάτω βλέπουμε τη διεπαφή του χρήστη admin στο περιβάλλον του ticket να έχει την επιπρόσθετη λειτουργία να αλλάζει υπεύθυνο τεχνικό. Σε αυτή τη διεπαφή περιέχονται και άλλα partial οπως τα tickets/_actions.html.erb, profile/_actions.html.erb και application/_devise.html.erb .



Εικόνα 13: Περιβάλλον Εργασίας Ticket και Admin

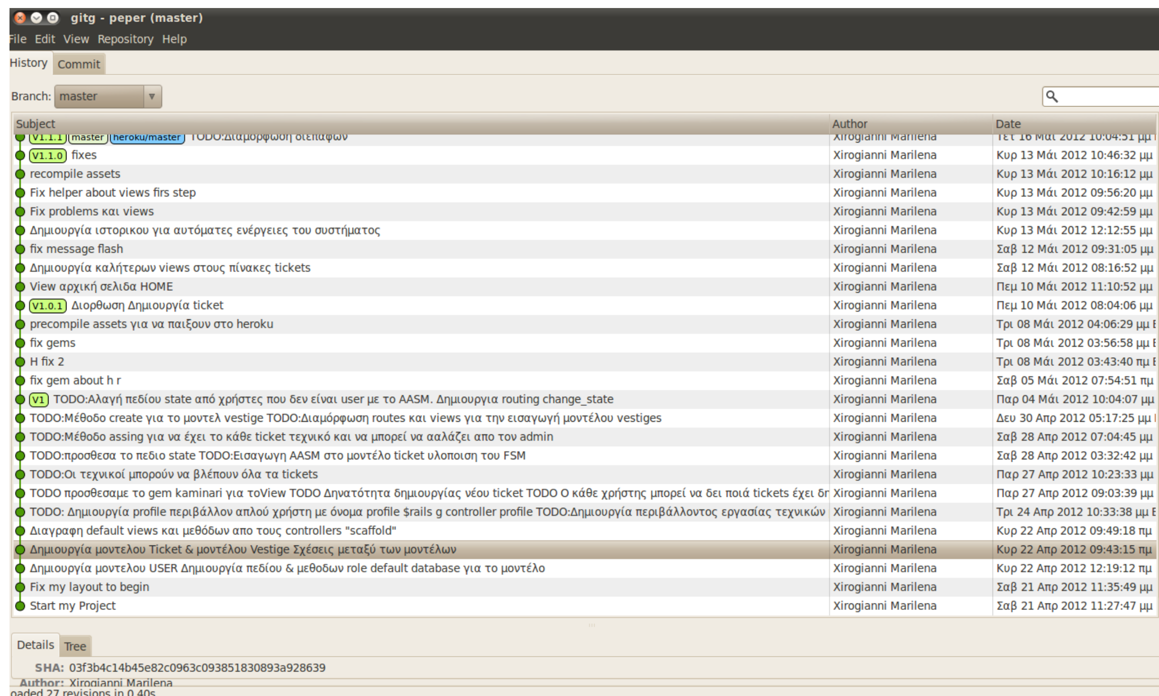
Για την δημιουργία της διεπαφής ακολουθήσαμε τις τεχνικές ανάπτυξης του framework χρησιμοποιήσαμε τη δυνατότητες του layout και των partials. Οι σελίδες μας τις συνθέτουν μικρά κομματάκια .

3.3.3 Ενημέρωση συστήματος

Ομοίως σε όλη τη διάρκεια της ανάπτυξης εφαρμόζουμε τον έλεγχο ανάπτυξης στο σύστημα και δημιουργούμε καταγραφές ιστορικού (commit) με το Git. Ενημερώνουμε το σύστημα μας στο heroku με την εντολή

```
$ git push heroku master
```

Στην παρακάτω εικόνα βλέπουμε πως έχουμε αναπτύξει τα στάδια της εφαρμογής σε γραφικό περιβάλλον έτσι όπως έχουν διαμορφωθεί έως τώρα . Διακρίνουμε της εκδόσεις της εφαρμογής και τα ονόματα που έχουμε δώσει στα brance και τα σχόλια ώστε να διακρίνουμε καλύτερα μέσα στο χρόνο ή από άλλου συναδέλφους τις αλλαγές στο κώδικα κατά την ανάπτυξη.



Εικόνα 14: Γραφική Αναφορά Των Εκδόσεων της Εφαρμογής στο Β΄ Στάδιο

ΚΕΦΑΛΑΙΟ 4

|Θέματα Ασφάλειας|

4.1 ΘΕΜΑΤΑ ΑΣΦΑΛΕΙΑΣ ΣΤΗ RUBY ON RAILS

Η πλατφόρμα ανάπτυξης λογισμικού (framework) πέρα να κτίζουμε εφαρμογές πρέπει να βοηθάει για την ασφάλεια της εφαρμογής. Το 75% των επιθέσεων γίνετε στο επίπεδο της εφαρμογής και λιγότερο σε επίπεδο βάσης ή εξυπηρετητή. Αν η εφαρμογή μας ακολουθήσει σωστά τη χρήση και ανάπτυξη με βάση το framework θα είμαστε σε θέση να δημιουργήσουμε ασφαλή συστήματα.

Η ασφάλεια εξαρτάται από τους προγραμματιστές και μερικές φορές τις μεθόδους ανάπτυξης της εφαρμογής. Η rails μελετώντας ορισμένες επιθέσεις έφτιαξε τις δομές για να τις αποφύγουμε με εύκολο τρόπο παρακάτω θα παρουσιάσουμε τομείς επίθεσης σε ένα σύστημα και τεχνικές επιθέσεων σε αυτές και τελικά πως εξαλείφονται στη rails.

4.1.1 Συνεδρίες χρηστών

Η Rails χρησιμοποιεί τον όρο συνεδρία χρήστη (session) για να μπορέσει να προσδιορίσει τις ενέργειές – αιτήσεις που ανήκουν σε κάθε χρήστη. Χωρίς τις συνεδρίες θα έπρεπε να προσδιορίζουμε την ταυτότητα του χρήστη για κάθε αίτηση. Η Rails δημιουργεί αυτόματα μια συνεδρία αν κάποιος μπει στο σύστημα ή φορτώνει την υπάρχουσα συνεδρία αν ο χρήστης χρησιμοποιεί ήδη την εφαρμογή.

Μια συνεδρία αποτελείτε από ένα hash και ένα αριθμό id με 32 χαρακτήρες. Κάθε μπισκότο (cookie) που στέλνετε στο brower του πελάτη περιέχει το id της συνεδρίας , ο brower το στέλνει πίσω σε κάθε αίτημα προς τον server έτσι εξασφαλίζετε η μοναδικότητα της. Το id της συνεδρίας αποτελείτε από τυχαίες τιμές που λαμβάνονται από την τρέχουσα ώρα με τυχαίους αριθμούς 0 ή 1 και μέσα στον επεξεργαστή της ruby μεταφράζονται σε μια συμβολοσειρά . Μέχρι σήμερα ο MD5 είναι ασυμβίβαστες οι συμβολοσειρές που παράγει, θεωρητικά είναι δυνατόν να παραχθούν όμοια hash με απειροελάχιστη πιθανότητα και δεν έχει καταγραφεί καμία επίθεση από αυτό το γεγονός έως τώρα.

4.1.2Επίθεση συνεδρίας από το μέσο μετάδοσης

Σε κάθε αίτημα η εφαρμογή φορτώνει τα στοιχεία του χρήστη προσδιορίζοντας τα στοιχεία από τη συνεδρία χωρίς να ζητά ξανά πιστοποίηση. Το id της συνεδρίας αποθηκεύετε στα cookies. Ο κάθε ένας που παίρνει ένα cookies από κάποιον άλλον μπορεί να χρησιμοποιήσει την εφαρμογή ως χρήστης και να υποκλέψει τα δικαιώματα του. Η Rails προσφέρει ασφαλή σύνδεση μέσο του SSL ώστε αν βρισκόμαστε σε μη ασφαλές δίκτυο η υποκλοπή (παράδειγμα με την διαδικασία sniff)των cookies να μην μαρτυρήσει το id της συνεδρίας.

4.1.3 Επίθεση μέσο συνεδρίας από τον υπολογιστή μας

Κατά την περιήγηση στο σύστημα παράγονται δεδομένα καλό θα είναι να μην αποθηκεύουμε αυτά στη συνεδρία αλλά στη βάση δεδομένων, όταν ένας χρήστης επαναλαμβάνει αιτήματα σε ένα σύστημα και δεν ενημερώνετε η βάση δεδομένων ενεργεί προς το σύστημα με τα αρχικά δεδομένα που είναι λανθασμένα. Να μην αποθηκεύονται κρίσιμα δεδομένα αντικείμενα συνεδρίας γιατί σβήνοντας cookies ή κλείνοντας το πρόγραμμα περιήγησης χάνονται. Ή αν ο χρήστης αποθηκεύσει την συνεδρία στον brower τότε κάποιος άλλος χρήστης μπορεί να έχει πρόσβαση σε αυτά.

Η Rails παρέχει διάφορους μηχανισμούς για την αποθήκευση hashes και id συνεδριών οι πιο σημαντικοί από αυτούς είναι το **ActiveRecord::SessionStore** και το **ActionDispatch::Session::CookieStore** . Το πρώτο αποθηκεύει το hash και τη χρονική διάρκεια της συνεδρίας στη βάση και σε κάθε νέο αίτημα που λαμβάνει δημιουργεί νέο που αντικαθιστά το προηγούμενο στη βάση. Το δεύτερο βάζει όριο στο μέγεθος των cookies στα 4KB έτσι δεν μπορούμε να έχουμε πολλές πληροφορίες σε μια συνεδρία, το id της συνεδρίας αποθηκεύεται στο cookies και προτίθεται στο τέλος του ένα αναγνωριστικό για τη δημιουργία της κωδικοποίησης και δίνετε από το κλειδί που ορίζουμε εμείς μέσα στο αρχείο της εφαρμογής enviroment.rb .

4.1.4 Επιθεση μέσο συνεδρίας σε παράλληλο χρόνο χρήσης.

Επίθεση καθήλωση συνεδρίας ορίζουμε την επίθεση όπου ο κακόβουλος χρήστης αναγκάζει την εφαρμογή να δώσει προκαθορισμένη id στη συνεδρία έτσι δεν χρειάζεται να κάνει ενέργειες να την υποκλέψει την έχει ήδη. Η Rails το αντιμετωπίζει απλά με τη πρόσθεση μίας γραμμής κώδικα =>resset_session ελέγχει αν κάθε αίτημα (action) που έρχεται και αρνήται την πρόσβαση εάν αν πληροφορίες δεν ταιριάζουν γιατί σημάνει ότι προήλθε από εξωτερικούς παράγοντες και όχι από την εφαρμογή έτσι δε δέχεται τη νέα συνεδρία με γνωστό id προέλευσης από άγνωστο χρήστη.

Επίθεση λήξης συνεδρίας, αν μια συνεδρία δεν έχει λήξει ακόμα και αν έχει κλείσει τον brower για να αποφύγουμε να χρησιμοποιηθεί μια ανοικτή συνεδρία στη Rails υπάρχει η μέθοδος session.sweep με την οποία ορίζουμε μέσα στην εφαρμογή χρονικό περιθώριο μιας συνεδρίας ανενεργής για ένα ορισμένο χρονικό διάστημα.

Η CSRF (Cross – Site – Forgery) είναι από τις συχνές επιθέσεις σε ιστοσελίδες, λειτουργεί με κώδικα που εισάγουμε στην ιστοσελίδα και εκτελείται όταν υπάρχει ενεργός πιστοποιήμενος χρήστης στο σύστημα. Ο χρήστης ενεργοποιεί την εκτέλεση του κώδικα χωρίς να το καταλάβει κάνοντας κλικ απλά σε ένα διαδραστικό αντικείμενο διεπαφής όπως μια εικόνα ή πίσω από ένα κουμπί εκτέλεσης κ.τ.λ. Στη Rails δυσκολευόμαστε να καταφέρουμε μια τέτοια επίθεση. Πρώτον η CSRF επίθεση πρέπει να περιέχει αίτημα RESTfull ως προς την εφαρμογή, κάθε αίτημα να συνοδεύεται και από ένα ρημα Post ή Get. Δευτέρων στη Rails χρησιμοποιούμε τη μέθοδο protect_from_forgery που προσθέτει μια ένδειξη ασφάλειας από τον χρήστη προς τον server . Αυτή η ενδειξη αν δεν είναι αυτή που περιμένει ο server η συνεδρία κλείνει και δημιουργείται νέα, έτσι το κακόβουλο αίτημα δεν φτάνει ποτέ στον προορισμό του.

4.1.5 Ανακατεύθυνση ιστοσελίδων και αρχείων

Η ανακατεύθυνση σε μια δικτυακή εφαρμογή είναι η μετάβαση σε μια ικανοποιημένη εφαρμογή ή αρχείου αγνώστου προέλευσης και υπόληψης. Ως επίθεση συ μένει οτι πως ο εισβολέας όχι μόνο έχει μεταφέρει το χρήστη σε ιστοσελίδα – παγίδα αλλά μπορεί να αποκομίσει στοιχεία για αυτόνομη επίθεση. Είναι ευθύνη του προγραμματιστή να διαφυλάξει τη μείωση τέτοιων επιθέσεων.

4.1.6 Ανακατεύθυνση ιστοσελίδας

Όταν ο χρήστης έχει τη δυνατότητα να περάσει τμήματα στο URL για ανακατεύθυνση η εφαρμογή είναι ευάλωτη σε επίθεση και να καταλήξουμε να οδηγηθούμε σε μια ψεύτικη ιστοσελίδα και να αισθανόμαστε όπως την αρχική. Αυτό που μπορούμε να κάνουμε όταν δημιουργούμε την εφαρμογή να χρησιμοποιούμε τη μέθοδο redirect_to ακολουθούμενη από δικές μας παραμέτρους να την ανακατευθύνουμε σε ένα κύριο action και να μην ακολουθήσει το πρόσθετο url ή να μην δώσουμε δικαίωμα στο χρήστη να προσθέτει στο url.

4.1.5 Ανακατεύθυνση αρχείων

Στην εφαρμογή ο χρήστης έχει το δικαίωμα να ανεβάσει αρχεία. Η επίθεση σε αυτό το γεγονός γίνετε είτε τα ανεβασμένα αρχεία να αντικαταστήσουν αρχεία της εφαρμογής είτε να περιέχουν εκτελέσιμο κώδικα και παρέμβουν στο σύστημα. Για να αποφύγουμε αυτού του είδους την επίθεση θα πρέπει να προσέχουμε τα ονόματα των αρχείων που επιτρέπουμε να περάσουν και καλό είναι τα ανεβασμένα αρχεία να αποθηκεύονται ένα επίπεδο πιο χαμηλά από την εφαρμογή. Ωστόσο για το κατέβασμα αρχείων μπορούμε να τοποθετούμε φίλτρα ώστε να μην αποσπώνται από την εφαρμογή αρχεία με σημαντικές πληροφορίες.

4.2 ΕΠΙΘΕΣΕΙΣ ΣΕ ΛΟΓΑΡΙΑΣΜΟΥΣ ΔΙΑΧΕΙΡΗΣΤΗ

Οι λογαριασμοί διαχειρηστή κατέχονται από χρήστες με περισσότερες αρμοδιότητες διαχείρισης οντοτήτων στις εφαρμογές ή λογαριασμούς με πρόσβαση σε απόρρητα αρχεία στοιχείων.

4.2.1 Ενδοδικτυακή

Οι διαδικτυακές και επιθέσεις σε προνομιούχους λογαριασμούς είναι οι πιο συχνές διότι παρέχουν προνομιακή πρόσβαση .Οι επιθέσεις διαδικτυακά γίνονται με XSS δηλαδή επεμβαίνουμε στην εφαρμογή μέσο δικτύου και μπορούμε να επεκτείνουμε την επίθεση με τη συγκατάβαση του server.

4.2.1 Μαζική αποστολή στοιχείων

Η ομαδική εκχώρηση τιμής, λαμβάνει περισσότερα από ένα στοιχεία μαζί σε ένα μοντέλο . Αν δε πάρομε μέτρα προφύλαξης επιτρέπει στον επιτιθέμενο να δημιουργήσει μέσα από το μοντέλο στήλη στο πίνακα με ορίσματα της επιλογής του. Ακόμα η επίθεση μπορεί να επεκταθεί αν ο πίνακας είναι αλληλεξαρτώμενος με άλλους.

Η Rails για να αποφύγει αυτή την επίθεση παρέχει δύο μεθόδους στο Active Record για να λύσει το πρόβλημα της μαζικής αποστολής. Η μέθοδος **attr_proteced** παίρνει μια λίστα όπου τα χαρακτηριστικά της δεν είναι προσβάσημα αν κάνουμε μαζική αποστολή για παράδειγμα:

```
attr_accessible :email, :password, :password_confirmation,
:remember_me
```

Η μέθοδος attr επιτρέπει να κάνουμε μαζική εκχώρηση τιμής και ως ρόλος του χρήστη για παράδειγμα

```
attr_accessible :email, :password, :password_confirmation,
:remember_me, :role, :as => :admin
```

το πεδίο role της εφαρμογής μας μπορεί να αλλάξει μόνο από κάποιον που είναι ήδη διαχειριστής ως προς την εφαρμογή. Για την εκχώρηση τιμών σε αυτό το πεδίο χρειάζεται να χρησιμοποιήσουμε τη δεύτερη μέθοδο assing_attributes όπου δέχεται τιμές μόνο όταν ισχύει η παραπάνω συνθήκη. Αν θέλουμε να αλλάξουμε το πεδίο role σε ένα χρήστη δίνουμε το κώδικα

```
user.assing.attributes({:email =>”email ”, :is_admin
=>true}, :as =>:admin)
```

Αξίζει να αναφερθεί μια ακόμα υπηρεσία της rails για τις ομαδικές εκχωρήσεις τιμών για να προστατεύσει ολόκληρο το έργο είναι να επιβάλει σε όλα τα μοντέλα να καθορίζουν τα προσβάσημα χαρακτηριστικά τους. Θέτουμε

```
config.active_record.whitelist_attributes = true
```

έτσι ενεργοποιούμε μια άσπρη λίστα για ομαδική αποστολή στοιχείων.

ΚΕΦΑΛΑΙΟ 5

|Άλλα frameworks - Συμπεράσματα|

5.1 FRAMEWORK ΚΑΙ Η RAILS

Η ruby on rails είναι framework το οποίο γεννήθηκε από τη φιλοσοφία “προγραμμάτισε εφαρμογές με ευτυχία” και τα γεγονότα δείχνουν ότι πέτυχε τους στόχους της. Για να είμαστε αντικειμενικοί θα παρουσιάσουμε παρακάτω και άλλα framework που χρησιμοποιούνται ευρέος για την ανάπτυξη εφαρμογών ιστοσελίδων και είναι αναγνωρισμένα καθώς μέσω αυτών έχουν υλοποιηθεί διεθνής αναγνωρισμένες ιστοσελίδες.

5.1.1 Django

Το Django είναι framework για τη γλώσσα προγραμματισμού Python και ακολουθεί το Montel -View – Controller . Βασικός στόχος του Django είναι να κάνει εύκολη τη δημιουργία πολύπλοκων βάσεων δεδομένων σε ιστοσελίδες. Βέβαια στερεί στο ότι δεν έχει γραφικό περιβάλλον και περιέχει μόνο τα απαραίτητα. Είναι πιο χρήσιμο αν θέλουμε κάνουμε εφαρμογές που θα χρησιμοποιήσουν άνθρωποι με τεχνικές γνώσεις. Η κοινότητα του ακολουθεί άκαμπτη νοοτροπία ως προς του κανόνες και ακόμα αν χρειαστεί να παραβούν αμέσως το αναιρούν . Δεν είναι εύκολη η προσέγγιση νέων ατόμων η τεκμηρίωση του κώδικά είναι μινιμαλιστική και υποθέτουν θα πρέπει να γνωρίζουμε το σύνολο της φιλοσοφίας της Python. Το Django είναι πιο εναποθηκευμένο περιορίζοντας τις δυνατότητες ανάπτυξης.

5.1.2 Symfony

Το Symfony στηρίζεται σε μια δημοφιλή γλώσσα προγραμματισμού δημιουργημένη για ιστοσελίδες την PHP. Όλοι περιμένουμε να κατέχει αξιοσέβαστες δυνατότητες και έτσι συμβαίνει αλλά αν ήταν τέλειο δεν θα υπήρχε η ανάγκη να δημιουργηθεί ένα νέο όπως η RoR ή το Cake στηριζόμενο πάλι σε PHP .

Το Symfony ακολουθεί την αρχιτεκτονική Montel – View -Controller επίσης. Λόγο της PHP διαθέτει μεγάλη κοινότητα και και δωρεάν διάθεση υλικού εκμάθησης του framework. Η εφαρμογή αποτελείτο από κομμάτια και προσφέρει από προεπιλογή ασφάλεια από CSRF και XSS επιθέσεις απαιτεί από τον προγραμματιστή όλα τα υπόλοιπα όπως τη ρύθμιση της βάσης δεδομένων κατά την διάρκεια της ανάπτυξης αφήνει έτσι το προγραμματιστή με περισσότερες ευθύνες.

Η php έχει μια έλλειψη στην συμβατότητα νέας έκδοσης της γλώσσας, αυτό επηρεάζει άμεσα τη διάρκεια ζωής του έργου. Είναι σοβαρό πρόβλημα να διατηρηθούν τα έργα που έχουν δημιουργηθεί και να παρέχουμε νέες τεχνολογίες σε νέους πελάτες .

5.1.3 Cake php

Το Cake php είναι το νέο framework με php που εμφανίστηκε μετά το Symfony και η αιτία ήταν η Ruby on Rails. Περιέχει αρκετά από τα χαρακτηριστικά που συναντάμε και στη RoR όπως το χωρισμό της εφαρμογής σε μικρότερα κομμάτια , την αρχιτεκτονική Montel – View – Controller, την χαρτογράφηση Object Relation Maping που παραδεχόμαστε ότι είναι λειτουργικά ομως υπάρχουν και κάποια τα οποία δε τα συναντάμε στο Cake εξίσου λειτουργικά για τις εφαρμογές ιστοσελίδων.

Η RoR τίμια να έχει περισσότερα πλεονεκτήματα ανάπτυξης. Το Cake php δεν διαθέτει migration μοντέλα για βάσης δεδομένων και πρέπει να ρυθμίσουμε το αρχείο SQL για τη δημιουργία και την ενημέρωση της βάσης δεδομένων. Στη RoR είναι πιο εύκολο να έχουμε πρόσβαση σε μεταβλητές ανάμεσα στο MVC. Για παράδειγμα στο controller είναι απευθείας προσβάσιμες από το view, το Cake PHP απαιτεί τη χρήση της μεθόδου set() για να μπορέσει να το κάνει. Η δρομολόγηση (RESTfull) των αιτήσεων δεν είναι τόσο αναπτυγμένη όσο στη RoR. Τέλος το CakePHP δεν διαθέτει κάποιο αντίστοιχο εργαλείο όπως το capistrano της RoR για την ανάπτυξη της εφαρμογής σε server. Και δεν είναι Agile.

5.2 ΚΡΙΤΗΡΙΑ ΚΑΙ ΠΡΟΫΠΟΘΕΣΕΙΣ ΠΛΑΤΦΟΡΜΩΝ ΑΝΑΠΤΗΞΗΣ

Επιπρόσθετα κριτήρια που μας οδηγούν στην ανακήρυξη τη Ruby on Rails ως ανταγωνιστικού framework είναι ότι καλύπτει σε μεγάλο βαθμό τις απαιτήσεις της αγοράς.

Προϋπολογισμός κατά κόστος αγοράς

Η RoR είναι πιο ενεργή ως κοινότητα οργανώνει συνέδρια συνεχώς, παρέχει υποστήριξη εκμάθησης και τεκμηριωμένα βοηθήματα (documentations) για το framework και βιβλιοθήκες. Η κοινότητα είναι φιλική προς τους αρχάριους και αυτό είναι εντός της πολιτικής της. Παρέχει δωρεάν διαλέξεις μαθημάτων και tutorials.

Το framework παρέχει την ικανότητα να δημιουργούμε εφαρμογές χωρίς να έχουμε αρκετές γνώσεις της ruby σε σχέση με άλλα framework που πρέπει να είσαι καλός γνώστης php και Python , και έχουμε άμεση επαφή με το κομμάτι της ιδιαιτερότητας της εφαρμογής σε λιγότερο χρόνο. Γίνετε πιο γρήγορα κατανόηση η νοοτροπία της και ο τρόπος εργασίας στην ανάπτυξη γιατί απαντήσουμε εύκολα και γρήγορα εφαρμογές οι οποίες λειτουργούν χωρίς να έχουμε εξειδικευμένες γνώσεις σε όλους τους τομείς των δικτυακών εφαρμογών .

Ο χρόνος παράδοσης του έργου είναι ο πιο γρήγορος στην αγορά χάρι στη agile ανάπτυξη της ruby και μειώνουμε το προϋπολογισμό του έργου από την έρευνα περνάμε από την σκέψη απευθείας στη πράξη και το παραδίδουμε. Οι ευτυχισμένοι προγραμματιστές είναι πιο παραγωγικοί.

Συνεχής υποστήριξη και ανάπτυξη

Ο ισχυρός πυρήνας της κοινότητας της RoR αναπτύσσει συνεχώς το κωδικά, είναι εκείνη που διέδωσε το μοντέλο MVC και προωθεί νέες καινοτομίες στο διαδίκτυο. Υπάρχει συνεχείς ροή νέων βιβλιοθηκών (gems) . Την περιβάλει μια πολύ καλή κριτική για τη δημιουργία των testing και της στατικής ανάπτυξης.

Επεκτασιμότητα

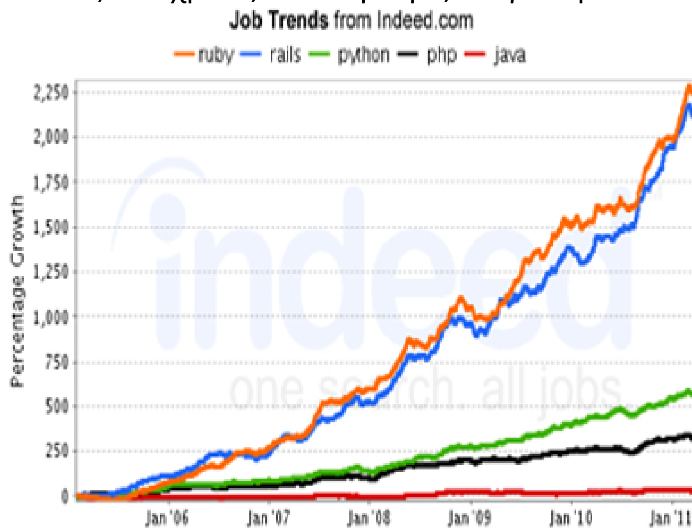
Όταν προκύπτουν νέα προβλήματα στο κόσμο της ανάπτυξης δυναμικών ιστοσελίδων η λύση τους θα πρέπει να είναι σε θέση να ανταποκριθεί και να νικήσει το πρόβλημα. Όταν αναπτύσσουμε ένα έργο πρέπει να μπορούμε να επεκτείνουμε τις δυνατότητες και να μπορεί να ανταποκρίνεται στις νέες τεχνολογίες. Η έγκυρη Agile ανάπτυξη της RoR δεν συναντάτε σε άλλα framework. Μπορεί να καλύψει νέες δυνατότητες ανάπτυξης του έργου και να εισάγει νέες τεχνολογίες στο παραδοτέο έργο.

Δημογραφικότητα

Η δημογραφικότητα ορίζει πόσο καλά το εργατικό δυναμικό τροφοδοτεί τη τεχνολογία και εξυπηρετεί τις ανάγκες του προϊόντος. Στα έργα τις RoR οι προγραμματιστές έχουν τη ομαδική εργασία και αλληλεπίδραση έτσι έχουμε μια σύνοψη διαφορετικών ιδεών στη λύση του προβλήματος. Η ίδια η Ruby on Rails εφαρμόζει νέες τεχνολογίες που θεωρούνται απαραίτητες για την δημιουργία δικτυακών εφαρμογών. Τείνει να δημιουργεί πρότυπα για την επίλυση συχνών προβλημάτων αλλά και να είναι ανοικτή σε καινούργιες μεθοδολογίες επίλυσης προβλημάτων , η παραγωγή πρότυπων παράγεται μέσα από την διαφορετική προσέγγιση που έχει ο κάθε προγραμματιστής για την επίλυση του προβλήματος .

Αποτελέσματα

Οι εταιρίες ενδιαφέρονται για το χρόνο παράδοσης του έργου, πολλές από αυτές έχουν καταλήξει ότι κατά την ανάπτυξη του έργου η ιστοσελίδα έχει παραδοθεί στο διαδίκτυο ενώ γίνετε χρήση της κατά τη διάρκεια της ανάπτυξης. Παρακάτω βλέπουμε ένα διάγραμμα ζήτησης θέσεων εργασίας προγραμματιστών και τις αντίστοιχες γλώσσες που χρειάζεται να γνωρίζουν για την ανάπτυξη εφαρμογών.



Εικόνα 15: Γράφημα ζήτησης θέσεων εργασίας για τηντην ανάπτυξη web εφαρμογών

Η αξιοπιστία για τη δημιουργία ιστοσελίδων αποδεικνύεται με τη παρουσίαση αξιόλογων διεθνών ιστοσελίδων οι οποίες έχουν παραχθεί με ruby. Μερικά από αυτά είναι Github, Twitter, Scribd, Fanpop, Geni, Crazy Egg, Basecamp, Wayfaring, Slideshare, Jango.

Η Rails μας δίνει ένα επίπεδο παραγωγικότητας που δεν μπορούμε να το συναντήσουμε σε κάποιο άλλο framework επειδή κάθε ιστοσελίδα web ξεκινάει με τον ίδιο τρόπο. Η ομοιότητα των εφαρμογών είναι τόσο κοντινή που αν αλλάξουμε ένα παράδειγμα σε διαφορετικές εφαρμογές δεν διαφέρουν πολύ μεταξύ τους, τα περισσότερα συνδέονται μεταξύ τους και πολλές φορές είναι το ίδιο. Για παράδειγμα δεν υπάρχουν μεγάλες διαφορές για την ανάπτυξη μιας δικτυακής εφαρμογής για την ενοικίαση αυτοκινήτων και μιας δικτυακής εφαρμογής βιβλιοθήκης δανεισμού βιβλίων.

5.3 ΣΥΜΠΕΡΑΣΜΑΤΑ

Στη διαδικασία της πτυχιακής εργασίας υλοποιήσαμε ένα σύστημα διαχείρισης διεργασιών η επιλογή του μέσου υλοποίησής είναι σημαντικός παράγοντας. Καθορίζει τον τρόπο αντιμετώπισης του προβλήματος και τις δυνατές διαστάσεις του έργου.

Η ανάγκη που γεννήθηκε η ruby και η πλατφόρμα RoR έχει λύσει το πρόβλημα "πρέπει να εστιάσουμε στους ανθρώπου, στο πώς οι άνθρωποι προγραμματίζουν ή χειρίζονται τις εφαρμογές των υπολογιστών" Matz . Δημιουργήσαμε ένα σύστημα για να καλύψουμε τις τεχνολογικές ανάγκες εργασίας σε κέντρο υποστήριξης ψηφιοποιήσαμε τις ηλεκτρονικές υπηρεσίες και προσφοράς χειρωνακτικής εργασίας σε αιτήσεις-αναφορές (ticket) στο σύστημα μας.

Το **αντίκτυπο** που είχε η όλη διαδικασία στο αντίστοιχο φοιτητή ως ανάπτυξης λογισμικού κρίνετε ως τη καλύτερη που θα μπορούσε να είχε βρεθεί. Έχοντας επαφή με τη δικτυακή ενημέρωση form , βοηθημάτων (tutorials), άρθρων και το Github και όταν αναπτύχθηκε η εφαρμογή σε ένα λειτουργικό στάδιο καταλαβαίνει κανείς ότι τίποτα δεν είναι τυχαίο στη RoR. Είναι ένα framework το οποίο καλύπτει το φάσμα των δικτυακών εφαρμογών αφήνοντας ενδιάμεσα κενά να βάλει ο προγραμματιστής τα κομμάτια της ιδιαίτερης εφαρμογής του. Αν για κάτι έχει κατηγορηθεί η RoR είναι ότι είναι μονότονη γιατί ακολουθεί σταθερές μεθοδολογίες και όλα μοιάζουν μεταξύ τους. Κάτι τέτοιο φυσικά δεν ισχύει γιατί η RoR δίνει το κορμό και ο προγραμματιστής είναι ελεύθερος να παραβεί τους κανόνες αρκεί να κατανοήσει τις ιδιότητες τους κάτι που δεν το έχουμε συναντήσει έως τώρα κάπου αλλού.

Οι **εργασιακές συνθήκες** σε επαγγελματικό επίπεδο είναι σαφώς πιο ευχάριστες. Η μεθοδολογία Agile απαιτεί επικοινωνία μεταξύ των ατόμων της ομάδας και των πελατών. Πέρα τον ανθρωποκεντρικό χαρακτήρα της εργασίας και την αποδοτικότητα της ομάδας παράγετε ένα παιχνίδι, το παιχνίδι παράγει δημιουργικότητα οτι απαιτείται από ένα προγραμματιστή σε όλη τη διάρκεια της ζωής του.

Πέρα από τα πλεονεκτήματα που υπάρχουν στο τρόπο εργασίας και στον επιχειρηματικό τομέα η Rails μπορεί να το δει κανείς ως **εκπαιδευτικό εργαλείο** web εφαρμογών. Όπως όλοι έχουμε καταλάβει το web εφαρμογές πέρα από την υλοποιήση των εφαρμογών τους για να καταφέρουν να λειτουργήσουν χρησιμοποιούν άλλα εργαλεία που με τη σειρά τους αποτελούντα από μικρότερα κομμάτια και καλύπτουν πολλές διαστάσεις από εφαρμογές και υλικό του ίντερνετ. Είναι τόσο μεγάλο ο κλάδος που δεν μπορεί κανείς να μάθει όλες τις πτυχές του η rails δίνει τη δυνατότητά να δημιουργήσουμε εφαρμογές με λίγες γνώσεις ανά κλάδο ή να μας παραπέμψει εύκολα σε επόμενο βήμα όταν ο εκπαιδευόμενος είναι έτοιμος.

5.4 ΜΕΛΛΟΝΤΙΚΗ ΕΡΓΑΣΙΑ ΚΑΙ ΕΠΕΚΤΑΣΕΙΣ

Το στάδιο της ανάπτυξης της εφαρμογής μας είναι δυνατόν να χρησιμοποιηθεί από την ομάδα διαχείρισης δικτύων του ΤΕΙ ως έχει. Η δομή της εφαρμογής είναι τέτοια ώστε να μπορούν να προστεθούν νέες υπηρεσίες με εύκολο τρόπο ικανοποιώντας τις συνεχώς μεταβαλλόμενες ανάγκες των χρηστών. Η Agile μεθοδολογία ανάπτυξής μας επιτρέπει να προσθέσουμε νέα σενάρια χρήσης με την εισαγωγή νέων μοντέλων και να επεκτείνουμε το σύστημα.

Η προσπάθεια να κάνουμε ανάλυση στο συνολικό πρόβλημα και μετά να το υλοποιήσουμε στη πράξη απέτυχε. Τίποτα δεν μπορεί να αντικαταστήσει την άμεση επαφή του πελάτη με την εφαρμογή στο στάδιο της ανάπτυξης. Η αρχική εικόνα που έχουν στη σκέψη και οι προσδοκίες από το έργο γίνονται πράξη σε μικρά κομματάκια που παραδίδονται σταδιακά συνθέτουν ένα σύνολο.

Μελλοντικά μπορούν να προστεθούν στην εφαρμογή η πιστοποίηση των χρηστών μέσω του κεντρικού LDAP του ΤΕΙ, καλύτερη παρουσίαση και επέκταση των στατιστικών, καθώς και ή ενσωμάτωση συστήματος online chat. Ο περιορισμένος χώρος και χρόνος ανάπτυξης της παρούσας πτυχιακής εργασίας δεν επέτρεψε την ανάπτυξη αυτών των λειτουργιών.

ΒΙΒΛΙΟΓΡΑΦΙΑ

References

[1] AGILE WEB DEVELOPMENT WITH RAILS 4E (RAILS 3.1) - RUBY, THOMAS, HANSSON - PRAGMATIC (2011) GUIDES

[2] ΕΙΣΑΓΩΓΗ ΣΤΗ ΘΕΩΡΙΑ ΥΠΟΛΟΓΙΣΜΟΥ MICHAEL SIPSER (2007)

[3] GIT INTERNALS SCOTT CHACON (2008)

[4] HTTP://GUIDES.RUBYONRAILS.ORG

[5] HTTP://GUIDES.RUBYONRAILS.ORG/MIGRATIONS.HTML 30/5/2012

[6] HTTP://GUIDES.RUBYONRAILS.ORG/ACTIVE_RECORD_VALIDATIONS_CALLBACKS.HTML 30/5/2012

[7] HTTP://GUIDES.RUBYONRAILS.ORG/ASSOCIATION_BASICS.HTML 30/5/2012

[8] HTTP://GUIDES.RUBYONRAILS.ORG/ACTIVE_RECORD_QUERYING.HTML 30/5/2012

[9] HTTP://GUIDES.RUBYONRAILS.ORG/LAYOUTS_AND_RENDERING.HTML 30/5/2012

[10] HTTP://GUIDES.RUBYONRAILS.ORG/FORM_HELPERS.HTML 30/5/2012

[11] HTTP://GUIDES.RUBYONRAILS.ORG/ACTION_CONTROLLER_OVERVIEW.HTML 30/5/2012

[12] HTTP://GUIDES.RUBYONRAILS.ORG/ROUTING.HTML 30/5/2012

[13] HTTP://GUIDES.RUBYONRAILS.ORG/ACTIVE_SUPPORT_CORE_EXTENSIONS.HTML 30/5/2012

[14] HTTP://GUIDES.RUBYONRAILS.ORG/I18N.HTML 30/5/2012

[15] HTTP://GUIDES.RUBYONRAILS.ORG/SECURITY.HTML 30/5/2012

[16] HTTP://GUIDES.RUBYONRAILS.ORG/DEBUGGING_RAILS_APPLICATIONS.HTML 30/5/2012

[17] HTTP://GUIDES.RUBYONRAILS.ORG/CONFIGURING.HTML 30/5/2012

[18] HTTP://GUIDES.RUBYONRAILS.ORG/COMMAND_LINE.HTML 30/5/2012

[18] HTTP://GUIDES.RUBYONRAILS.ORG/ASSET_PIPELINE.HTML 30/5/2012

[20] HTTPS://GITHUB.COM/PLATAFORMATEC/DEVISE 30/5/2012

[21] HTTPS://GITHUB.COM/RUBYIST/AASM 30/5/2012

[22] HTTPS://GITHUB.COM/SFERIK/RAILS_ADMIN 30/5/2012

[23] HTTP://WWW.GRAPHVIZ.ORG/HOME.PHP 1/3/2011

[24] HTTP://WWW.HSC.COM/PORTALS/0/AGILE%20PROCESS.JPG 1/3/2011

[25] HTTP://BLOG.HEROKU.COM/ 1/3/2011

[26] HTTP://GIT-SCM.COM/ABOUT 1/3/2011

[27] HTTP://WWW.ROCKSTARPROGRAMMER.ORG/POST/2008/APR/06/DIFFERENCES-BETWEEN-MERCURIAL-AND-GIT/ 1/3/2011

[28] HTTP://WWW.SYMFONY-PROJECT.ORG/GETTING-STARTED/1_4/EN/01-INTRODUCTION 25/4/2011

[29] HTTP://TOOLS.IETF.ORG/HTML/RFC5789 25/4/2011

[30] HTTP://WWW.INDEED.COM/TRENDGRAPH/JOBGRAPH.PNG?25/4/2011

[31] HTTP://ARTICLES.BUSINESSINSIDER.COM/2011-05-11/TECH/30035869_1_RUBY-RAILS-CUSTOM-SOFTWARE 25/4/2011

[32] HTTP://WWW.CHRISWPAGE.COM/2009/05/CHOOSING-THE-BEST-PLATFORM-FOR-THE-JOB-CMS-SOLUTION-PHP-MVC-DJANGO-OR-RUBY-ON-RAILS/ 31/5/2012

[33] HTTP://MURRAIN.NET/2008/11/CAKE-VS-SYMFONY.HTML 12/5/2012

[34] HTTP://GAUTAMREGE.WORDPRESS.COM/2010/01/10/PHPCAMP-CAKEPHP-VS-RUBY-ON-RAILS/ 10/5/2012

[36] HTTP://WWW.SETFIREMEDIA.COM/BLOG/50-OF-THE-BEST-WEBSITES-DEVELOPED-USING-RUBY-ON-RAILS 1/6/2012

ΠΑΡΑΡΤΗΜΑ Α

Βασικές 'Εννοιες

Στα παράδειγμα που χρησιμοποιούμε στο πρώτο κεφάλαιο για δώσουμε σαφή λειτουργίες της Ruby on Rails χρησιμοποιούμε το μοντέλο - πίνακα Toy με πεδία material, name, demotation.

Με την έννοια **αίτηση στο σύστημα** αναφερόμαστε σε οποιαδήποτε ticket μπορούμε να δημιουργήσουμε στο σύστημα, αίτηση βλάβης σύνδεσης τηλεφώνου, σύνδεσης ιντερνέτ, λογαριασμού eclass υπηρεσία blog, υλικού όπως υπολογιστή, τηλεφωνικής σύνδεσης ip διεύθυνσης. Αίτηση για νέα σύνδεση τηλεφωνίας, σύνδεσης ιντερνέτ, αίτηση για νέα wifi συσκευής-σύνδεσης. Πληροφορίες για νέο υλικό εξοπλισμού όπως ups, servers,racks,swith κτλ.

Με την έννοια **αίτηση στην εφαρμογή** αναφερόμαστε στα action μέσα στην εφαρμογή. Τα actions δημιουργού ντε από μια δράση στη διεπαφή, μεταξύ των μοντέλων model-git-controller και βάσης δεδομένων.

Διαχείριση έκδοσης (version control) είναι η διαχείριση των αλλαγών σε έγγραφα, προγράμματα ηλεκτρονικών υπολογιστών, μεγάλων ιστοσελίδων και άλλων συλλογών πληροφοριών που προσθέτει ιδιαίτερες δυνατότητες σε ένα μεγαλύτερο λογισμικό.

Σχεσιακή βάση είναι μια συλλογή δεδομένων οργανωμένη σε συσχετισμένους πίνακες που παρέχει ταυτόχρονα ένα μηχανισμό για ανάγνωση, εγγραφή, τροποποίηση ή και πολύπλοκες διαδικασίες πάνω σε δεδομένα.

Object-Relation-Maping (ORM) είναι λογισμικό για τη μετατροπή δεδομένων μεταξύ βάσεων και αντικειμενοστραφή γλωσσών προγραμματισμού.

Πρωτόκολλο Μεταφοράς Υπερκειμένου (HTTP) είναι η κύρια μέθοδος που χρησιμοποιούν τα πρωτόκολλα για να μεταφέρουν δεδομένα ανάμεσα σε διακομιστή (server) και πελάτη (clients)

Ανοικτό λογισμικό (Open Source) χαρακτηρίζεται το μοντέλο διάθεσης λογισμικού όπου ο πηγαίος κώδικας είναι διαθέσιμος σε όπιον ενδιαφέρετε γι' αυτόν. Οι βασικοί όροι χρήσης αυτού είναι α) η ελεύθερη χρήση και β) η αντιγραφή / αναδιανομή και μεταβολή / βελτιώση κώδικα.

Plugin ορίζεται ένα σύστημα συστατικών ενός λογισμικού που προσθέτει επιπλέον δυνατότητες και ιδιότητες σε ένα άλλο μεγαλύτερο λογισμικό.

XSS επίθεση πρόκειται για επιθέσεις μέσο εκτελέσεις προγραμμάτων που βρίσκονται στο server ή στο client. Η παρεμβολή σε αυτά τα εκτελέσιμες λειτουργίες είναι και η αντίστοιχη επίθεση.

Σχήματα

Για το **Σχήμα 1.1 Montel-View-Controller** δημιουργήσαμε το παρακάτω κώδικα σε γλώσσα dot:

```
digraph G
{
rankdir=RL;



Database [shape=Msquare,
        image="D.png",
        style=radial,
        gradientangle=90];
View [shape=Mdiamond ,
            image="V.png",
        gradientangle=80,
        style=radial];

Browser[image="b.png"];
Controller[image="C.png"];
Model[image="M.png"];

Browser -> Controller;
Controller -> View;
View -> Browser;
Controller -> Model;
Model -> Controller;
Model -> Database;
Database -> Model;
}
```

Για το **Σχήμα 1.2 Μηχανή Πεπερασμένων Καταστάσεων** δημιουργήσαμε το παρακάτω κώδικα σε γλώσσα dot:

```
digraph finite_state_machine {
    rankdir=LR;

    node [shape = doublecircle] "q1";
     node [shape =circle]"q2" "q3" "q4";
    "q1" -> "q2"  [ label = "1" ]
    "q2" -> "q3"  [label = "0"]
    "q2" -> "q4"  [ label = "1"]
    "q3" -> "q4"  [ label = "1"]
    "q4" -> "q1"  [label ="0"]

}
```

Για το **Σχήμα 2.1 Rails και MVC** δημιουργήσαμε το παρακάτω κώδικα σε γλώσσα dot:

```
digraph G
{
rankdir=RL;

Database [shape=Msquare,
      image="D.png",
                 label = "Ξ’Ξ±ΟƒΞ·
Ξ”ΞµΞ΄ΞΏΞΌΞΞ½Ο‰Ξ½",
      style=radial,
```

```
     gradientangle=90];
View [shape=Mdiamond ,
                image="V.png",
                label = "Views",
     gradientangle=80,
     style=radial];
Routing [shape=Msquare,
                image="R.png",
                label = "Routing",
     style=radial,
     gradientangle=90];
1 [shape=Msquare,
                image="user.png",
                label = "",
     gradientangle=80,
     style=radial];


Controller[image="C.png",
label = "controller"];
Model[image="M.png",label = "Active Record"];
Controller ->Model;
Routing -> Controller
Model -> Controller;
Controller -> View;
1 -> Routing;


 View -> 1;
Model ->Database;
Database ->  Model;

}
```

Για το **Σχήμα 3.1 FSM** καταστάσεις ticket δημιουργήσαμε το παρακάτω κώδικα σε γλώσσα dot:

```
digraph finite_state_machine {
rankdir=LR;
node [shape = doublecircle]; "unread"
node [shape = circle]; "close" "open" "frozen" "close";
```

```
"unread" -> "open" [ label = "Καταχώρηση τενχικού" ];
"open" -> "open" [label = "Αλαγή τεχνικού"]
"open" -> "frozen" [label ="Διακοπή Ticket"]
"frozen" -> "open" [label ="Επαναφορά Ticket"]
"open" -> "close" [label ="Κλείσιμο"]
"close" -> "open" [label = "Επανεξέταση"]
}
```

Για το **Σχήμα 3.2 Βασικοί πίνακες και οι σχέσεις τους** Χρησιμοποιήσαμε το προγραμα Dia Diagram Editor. http://dia-installer.de/index.html.en

ΠΑΡΑΡΤΗΜΑ Β

|Κώδικας|

Ο ΚΩΔΙΚΑΣ ΤΩΝ ΑΡΧΕΙΩΝ

app/controller/application.rb

```
class ApplicationController < ActionController::Base
  protect_from_forgery
def after_sign_in_path_for(resource)
    if current_user.is_user?
      profile_path(current_user)
    else
      todo_path(current_user)
    end
  end
end
```

app/controller/home.rb

```
class HomeController < ApplicationController
  def index
  end
end
```

app/controller/page.rb

```
class PageController < ApplicationController
 def help
```

```
  end
  def more
   end
end
```

app/controller/profile.rb

```
class ProfileController < ApplicationController
  respond_to :html
  def show
    @user = User.find(params[:id])
    @tickets = @user.tickets
    respond_with(@user)
  end
end
```

app/controller/ticket.rb

```
# encoding: utf-8
class TicketsController < ApplicationController
 respond_to :html
    def new
    @ticket=current_user.tickets.build
    respond_with(@ticket)
  end
  def create
    @ticket = current_user.create_ticket(params[:ticket])
    @ticket.save
    redirect_to profile_url(current_user)
  end
  def show
```

```
    if current_user.is_user?
      # Fetch ticket throught user
      @ticket = current_user.tickets.find(params[:id])
    else
           @ticket = Ticket.includes(:user).find(params[:id])
    end
    @vestiges = @ticket.vestiges.all
    respond_with(@ticket)
  end
  def assign
      if current_user.is_admin?
      @tech = User.find(params[:technical_id])
      @ticket = Ticket.find(params[:ticket_id])
      @ticket.assign_to(@tech,current_user)
    end
    if current_user.is_tech?
      @ticket = Ticket.find(params[:ticket_id])
      @ticket.assign_to(current_user,current_user)
    end
    @ticket.vestiges.create incident: "Προώθηση ticket σε 
#{@ticket.technical_id}", auto: true,
                            user: current_user, state: 
@ticket.state
    redirect_to ticket_url(params[:ticket_id])
  end
  def change_state
    unless current_user.is_user?
      @ticket = Ticket.find(params[:ticket_id])
      event = params[:event].to_sym
      if 
@ticket.aasm_permissible_events_for_current_state.include? event
        # I can use a case when here
```

```
        @ticket.send event

        @ticket.save

        @ticket.vestiges.create incident:"Αλλαγή κατάστασης σε
"+@ticket.state, auto: false,

                              user: current_user, state:
@ticket.state

      end

    end

    redirect_to ticket_url(params[:ticket_id])

  end

end
```

app/controller/todo.rb

```
class TodosController < ApplicationController

  def show

    @new_tickets =
Ticket.includes(:user).unread.page(params[:page])

    @my_tickets  = Ticket.assigned_to(current_user)

   end

end
```

app/controller/vestiges.rb

```
class VestigesController < ApplicationController


 def create

    if current_user.is_user?

      # Find the ticket throught User

      @ticket  =
current_user.tickets.find_by_id(params['ticket_id'])

    else

      @ticket  = Ticket.find_by_id(params['ticket_id'])
```

```
    end
    # Create a new vestige for the ticket
    @vestige  = @ticket.vestiges.create(params[:vestige])
    # Update ticket parameters
    @vestige.user  = current_user
    @vestige.state = @ticket.state
    @vestige.auto  = false
    @vestige.save
    redirect_to ticket_path(@ticket)
  end
end
```

app/helpers/applications.rb

```
module ApplicationHelper
def twitterized_type(type)
    case type
      when :alert
        "warning"
      when :error
        "error"
      when :notice
        "info"
      when :success
        "success"
      else
        type.to_s
    end
  end
 def get_home_path
    if current_user.nil?
      return "/"
```

```
    end
    if current_user.is_admin?
      return"/admin"
    end
    if current_user.is_user?
      profile_path(current_user)
    else
      todo_path(current_user)
    end
  end
end

 # Gives bootstrap class for button
  def aasm_type(type)
    case type
      when 'unread'
        "btn-danger"
      when 'active'
        "btn-warning"
      when 'close'
        "btn-success"
      when 'frozen'
        "btn-inverse"
      else
        "btn-#{type.to_s}"
    end
  end

    def aasm_type_ev(type)
    case type
      when 'do_open'
```

```
        "btn-danger"
      when 'active'
        "btn-warning"
      when 'do_close'
        "btn-success"
      when 'do_frozen'
        "btn-inverse"
      else
        "btn-#{type.to_s}"
    end
  end
```

app/mondels/ticket.rb

```
class Ticket < ActiveRecord::Base
  include AASM
  aasm_column :state

  belongs_to :user
  belongs_to :technical, :class_name => "User"
  has_many :vestiges

   after_initialize do |t|
    t.aasm_enter_initial_state if t.state.nil?
  end
  after_create do |t|
    # Δημιουργία ιστορικου για τη πράξη δημιουργίας ticket
    self.vestiges.create incident: "Ticket created", auto: true,
                              user: t.user, state: t.state
  end
```

```
    scope :not_closed,   where('state != ?',"close")
    scope :assigned_to,  lambda { |user| where('technical_id =
?', user.id) }


  aasm_initial_state :unread #αρχκική τιμή
  aasm_state :open
  aasm_state :frozen
  aasm_state :close
  aasm_state :unread


  aasm_event :do_open do
    transitions :to => :open, :from => [:unread, :close, :frozen,
:open]
  end
  aasm_event :do_close do
    transitions :to => :close, :from => [:open, :unread]
  end
  aasm_event :do_frozen do
    transitions :to => :frozen, :from =>[:open]
  end
    def can_do_event?(event)
      self.aasm_events_for_current_state.include?(event)
    end
    def assign_to(user, by)
    if self.can_do_event?(:do_close)
      self.technical = user
      self.do_open!
      save
      return true
    else
      return false
```

```
    end
  end
end
```

app/mondel/user.rb

```
class User < ActiveRecord::Base

  devise :database_authenticatable, :registerable,
         :recoverable, :rememberable, :trackable, :validatable

  attr_accessible :email, :password, :password_confirmation,
:remember_me
  attr_accessible :email, :password, :password_confirmation,
:remember_me, :role, :as => :admin


 scope :workers, where("role = 'admin' OR role= 'tech'")
  def create_ticket(*params)
   collection = self.tickets.create(params)
   collection.first
 end
has_many :tickets
   def is_admin?
    self.role == "admin"
  end
  def is_tech?
    self.role == "tech"
  end
  def is_user?
```

```
    self.role == "user"
  end
end
```

app/montel/vestiges.rb

```
class Vestige < ActiveRecord::Base
  belongs_to :ticket
  belongs_to :user
end
```

app/views/applications

app/views/applications/_actions.html.erb

```
<ul class="nav nav-list">
  <li class="nav-header">Περισσοτερα</li>
    <li><%= link_to "Home", root_path  %></li>
    <li><%= link_to "Help", page_help_path  %></li>
    <li><%= link_to "More", page_more_path  %></li>
</ul>
```

app/views/applications/_devise.html.erb

```
<div class="well sidebar-nav">
    <% if user_signed_in? %>
```

```
    Εισοδος ως <%= current_user.email %>.</br>


    <%= link_to "Logout", destroy_user_session_path, :method =>
:delete %>

    <p>

      <%= link_to "My Tickets", profile_path(current_user),
:class => 'btn' %>

    </p>

      <% else %>

    <%= link_to "Sign up", new_user_registration_path %> or

    <%= link_to "Sign in", new_user_session_path %>

  <% end %>

</div>
```

app/views/applications/_messages.html.erb

```
<% flash.each do |type, message| %>

  <div class="alert alert-<%= twitterized_type(type) %>">

    <a class="close" data-dismiss="alert">x</a>

    <p><%= message %></p>

  </div>

<% end %>
```

app/views/home.html.erb

```
div class="span8">

  <h3 >ΚΑΛΩΣΗΡΘΑΤΕ ΣΤΟ ΣΥΣΤΗΜΑ ΔΙΑΧΕΙΡΙΣΗΣ TICKET NOC</h3>

  <p><%= image_tag "r.jpg", :class=>'thumbnail offset' %></p>
```

```
</div>
```

app/views/layouts/application.html.erb

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>Κ.Ε.Δ.Δ.</title>
    <%= csrf_meta_tags %>

    <%= stylesheet_link_tag "application", :media => "all" %>

    <link href="images/favicon.ico" rel="shortcut icon">
    <link href="images/apple-touch-icon.png" rel="apple-touch-
icon">
    <link href="images/apple-touch-icon-72x72.png" rel="apple-
touch-icon" sizes="72x72">
    <link href="images/apple-touch-icon-114x114.png" rel="apple-
touch-icon" sizes="114x114">
  </head>
  <body>
    <div class="navbar navbar-fixed-top">
      <div class="navbar-inner">
          <div class="container">
          <a class="btn btn-navbar" data-target=".nav-collapse"
data-toggle="collapse">
            <span class="icon-bar"></span>
            <span class="icon-bar"></span>
            <span class="icon-bar"></span>
```

```
          </a>
          <a class="brand" href="<%= get_home_path %>">My
Place</a>
          <div class="container nav-collapse">
            <ul class="brand">
                <ll><%= link_to "Αρχική", "/"  %></li>
            </ul>
          </div>
        </div>
      </div>
    </div>
    <div class="container">
      <div class="content">
        <div class="row">
           <div class="span9">
            <%= render :partial => "application/_messages",
:locals => {:flash => flash} %>
              <%= yield %>
            </div>
            <div class="span3">
              <div class="well sidebar-nav">
                <%= render :partial => "devise" %>


                <h4>Αιτήσεις</h4>
                <ul class="well sidebar-nav">
                <%= render :partial => "actions" %>
                 </ul>
              </div><!--/.well -->
            </div><!--/span-->
        </div><!--/row-->
      </div><!--/content-->
```

```
      <footer>
        <p>&copy; Xirogianni Marielena</p>
      </footer>

    </div> <!-- /container -->

    <%= javascript_include_tag "application" %>

  </body>
</html>
```

app/view/profile.html.erb

```
<h4> Είσοδος ως <%=  @user.email %></h4>
<h2>Οι αναφορές μου </h2>
<table class="table table-striped">
  <thead>
    <tr>
      <th>ID</th>
      <th>Τύπος</th>
      <th>Περιγραφή</th>
      <th>Κατάσταση</th>
      <th>Actions</th>
    </tr>
  </thead>
  <tbody>
    <% @tickets.each do |t| %>
      <tr>
        <td><%= t.id %></td>
        <td><%=link_to t.category, ticket_path(t) %></td>
```

```
        <td><%=truncate t.descrition, :length => 25, :omission =>
'...' %></td>

        <td><%= t.state %></td>

        <td nowrap="nowrap"><i class="icon-time"></i><%=
distance_of_time_in_words(t.created_at, Time.now)%></td>

      </tr>

    <% end %>

  </tbody>

</table>
```

app/views/tickets

app/views/tickets/_actions

```
<h4><%=truncate @ticket.category %></h4>

<% if current_user.is_admin? %>

  <h3> Αντιστοιχήσε σε <h3>

  <%= form_for @ticket, :url => ticket_assign_path(@ticket),
:method=> :post do |f| %>

    <p style="text-align:center;">

     <%= select_tag(:technical_id, options_for_select(
User.workers.collect{ |u| ["#{u.email} ", u.id] }) ) %>

   </p>

   <p style="text-align:center;">

    <%= f.submit "Δώσε ticket", :class => 'btn btn-primary' %>

  </p>

  <% end %>

<% end %>

<% if current_user.is_tech? && !@ticket.new_record? %>

  <%= form_for @ticket, :url => ticket_assign_path(@ticket),
:method=> :post do |f| %>

    <p style="text-align:center;">
```

```
      <%= f.hidden_field(:technical_id, :value=> current_user.id)
%>

      <%= f.submit "Πάρε Ticket", :class => 'btn btn-primary' %>

    </p>

  <% end %>

<% unless current_user.is_user? %>

  <h3> Ενέργειες <h3>

  <p style="text-align:center;">

  <% @ticket.aasm_permissible_events_for_current_state.each do
|event| %>

    <% if (event != :do_open) %>

      <%= link_to event.to_s,

                    ticket_change_state_path(@ticket, :event=>
event),

                   :class => "btn #{aasm_type_ev(event.to_s)}" %>

    <% else %>

      <% if (@ticket.state != 'active') &&
(!@ticket.technical_id.nil?)%>

          <%= link_to event.to_s,

              ticket_change_state_path(@ticket, :event=> event),

                        :class => "btn
#{aasm_type_ev(event.to_s)}" %>

      <% end %>

    <% end %>

  <% end %>

  </p>

  <%end%>

  <%end%>
```

app/views/tickets/_form

```
<%= form_for @ticket, :method=> :post, :html => { :class =>
'form-horizontal' } do |f| %>
```

```
<div class="span8">
  <fieldset>
        <legend>Δημιουργία Ticket</legend>
    <div class="control-group">
      <%= f.label "Τύπος", :class => 'control-label'%>
      <div class="controls">
        <%= f.text_field :category,  :class => 'text_field' %>
      </div>
    </div>
    <div class="control-group">
      <%= f.label "Βασικά στοιχεία", :class => 'control-label' %>
      <div class="controls">
        <%= f.text_field :element, :class => 'text_field'
%>*(Δώσε στοιχεία όπως μπρίζα σύνδεσης/τηλεφώνου</br>,ussername
eclass,κτλ)
      </div>
    </div>
    <div class="control-group">
      <%= f.label "Βοηθητικα στοιχεία", :class => 'control-label'
%>      <div class="controls">
        <%= f.text_field :advice, :class => 'text_field' %>
      </div>
    </div>


    <div class="control-group">
      <%= f.label "Περιγραφή", :class => 'control-label' %>
      <div class="controls">
        <%= f.text_area :descrition, :class => 'text_area span3',
:rows=>4 %>
      </div>
    </div>
  </div>
```

```
    <div class="form-actions">
      <%= f.submit nil, :class => 'btn btn-primary' %>
      <%= link_to 'Cancel', profile_path(current_user), :class =>
'btn btn-warning' %>
    </div>
  </fieldset>
<% end %>
```

app/view/ticket/new.html.erb

```
<%= render :partial => 'form' %>
```

app/view/ticket/show.html.erb

```
<div class="well">
<h1> Ticket στοιχεία </h1>
<p>
  <b>User name:</b>
   <%= @ticket.user_id %>
</p>
<p>
  <b>όνομα:</b>
  <%= @ticket.category %>
</p>
<b>Τεχνικός:</b>
  <%= @ticket.technical_id %>
</p>
<p><b>Περιγραφή</b></p>
<div>
  <%= @ticket.descrition %>
</div>
```

```
<div><%= link_to 'Επιστροφή',  get_home_path, :class=> 'btn btn-
warning' %>

</div>

<div class="well">

<h3> Πληροφορίες </h3>

<table class="table table-striped">

  <thead>

  <tr>

    <th>Ιστορικό</th>

    <th>Χρήστης</th>

    <th>Ημερομηνία</th>

    <th>Κατάσταση</th>

    <th>Αυτόματο</th>

  </tr>

</thead>

<% @vestiges.each do |v| %>

  <tr>

    <td> <%= v.incident %></td>

    <td> <%= v.user_id %></td>

    <td nowrap="nowrap"><i class="icon-time"></i><%=
distance_of_time_in_words(v.created_at, Time.now)%></td>

    <td> <%= v.state %></td>

    <td> <%= v.auto ? "ΝΑΙ" : "ΟΧΙ" %></td>

  </tr>

<% end %>

</table>

</div>

<%= form_for [@ticket, @ticket.vestiges.build], :method=> :post,
:html => { :class => 'well' } do |f| %>

  <h3> Δώσε πληροφορίες</h3>

  <fieldset>

    <div class="control-group">
```

```
      <div class="controls">
        <%= f.text_area :incident, :class => 'text_area span8', 
:rows=>2 %>
      </div>
      <%= f.submit "Δημιουργία", :class => 'btn btn-primary' %>
    </div>
  </fieldset>
<% end %>
<hr/>
</div>
```

app/veiws/todos/_mytickets.html.erb

```
<table class="table table-striped">
  <thead>
    <tr>
    <th>ID</th>
    <th>Τεχνικός</th>
    <th>Τύπος</th>
    <th>Περιγραφή</th>
    <th>Ημερομηνία</th>
  </tr>
  </thead>
<% @my_tickets.each do |t| %>
  <tr>
    <td><%= t.id %></td>
    <td><%= t.user_id %></td>
    <td><%=link_to t.category, ticket_path(t) %></td>
    <td><%=truncate t.descrition, :length => 25, :omission => 
'...' %></td>
    <td nowrap="nowrap"><i class="icon-time"></i><%= 
distance_of_time_in_words(t.created_at, Time.now)%></td>
  </tr>
```

```
<% end %>
</table>
```

app/veiws/todos/_new_tickets.html.erb

```
<table class="table table-striped">
  <thead>
    <tr>
    <th>ID</th>
    <th>Κατασταση</th>
    <th>Χρήστης</th>
    <th>Τύπος</th>
    <th>Περιγραφή</th>
    <th>Ημερομηνία</th>
  </tr>
  </thead>
<% @new_tickets.each do |t| %>
  <tr>
    <td><%= t.id %></td>
    <td><%= t.state%>
    <td><%= t.user_id%>
    <td><%=link_to t.category, ticket_path(t) %></td>
    <td><%=truncate t.descrition, :length => 25, :omission =>
'...' %></td>
    <td nowrap="nowrap"><i class="icon-time"></i><%=
distance_of_time_in_words(t.created_at, Time.now)%></td>
  </tr>
<% end %>
</table>
```

app/veiws/todos/show.html.erb

```
<div class="tabbable">
    <ul class="nav nav-tabs">
    <li class="active"><a href="#1" data-toggle="tab">
Tickets</a></li>
    <li><a href="#2" data-toggle="tab">Η Εργασία μου</a></li>
    <li><a href="#3"data-toggle="tab">Κλειστά Tickets</a></li>
    <li><a href="#4"data-toggle="tab">Διαφορα</a></li>
  </ul>
  <div class="tab-content">
      <div class="tab-pane active" id="1">
      <%= render :partial => "new_tickets" %>
    </div>
      <div class="tab-pane" id="2">
     <%= render :partial => "my_tickets" %>
    </div>
    <div class="tab-pane" id="3">
      <%=  "old_ticket" %>
    </div>
    <div class="tab-pane" id="4">
      <  "Διαφορα επιπροσθετα </br>  για τους τεχνικούς" >
    </div>
  </div>
</div>
```

app/confing/routes.rb

```
Peper::Application.routes.draw do
  get "page/help"
  get "page/more"
```

```
  mount RailsAdmin::Engine => '/admin', :as => 'rails_admin'
  resources :tickets, :only => [:new, :create, :show] do
    resources :vestiges,:only => [:create]
    post :assign
    get  :change_state
  end
   devise_for :users
  resources :profile, :only => [:show]
  resources :todos,   :only => [:show]
    root :to => "home#index"
  get "home/index"
end
```

ΔΡΟΜΟΛΟΓΗΣΕΙΣ ΑΙΤΗΣΕΩΝ

Ρήμα	Path-URL	Δράση/μέθοδος	Χρησιμοτητα
GET	/home/index(.:format)	index	**Εμφάνιση αρχική σελιδα**
POST	/tickets/:ticket_id/vestiges	create	**Δημιουργία ιστορικού μέσο ticket**
POST	/tickets/:ticket_id/assign	assign	**Επέμφαση πεδίο στοιχείου ticket**
GET	/tickets/:ticket_id/change_state	change_state	**Αλλαγή πεδίου ticket.state**
POST	/tickets(.:format)	create	**Δημιουργία στοιχείου ticket**
GET	/tickets/new	new	**Επιστρέφει μια φόρμα HTML για τη δημιουργία νέου ticket**
GET	/tickets/:id	show	**Εμφανίζει το συγκεκριμένο ticket**
GET	/users/sign_in	sessions#new	**Επιστρέφει μια φόρμα HTML για είσοδο χρήστη**
POST	/users/sign_in	sessions#create	**Έναρξη συνεδρία χρήστη**
DELETE	/users/sign_out	sessions#destroy	**Λήξη συνεδρίας χρήστη**
POST	/users/password(.:format)	/passwords#create	**Δημιουργία κωδικού user**
GET	/users/password/new(.:format)	passwords#new	**Επιστρέφει μια φόρμα HTML με τιμή κωδικού**
GET	/users/password/edit(.:format)	passwords#edit	**Επιστρέφει μια φόρμα HTML για τη επεξεργασία του κωδικού**
PUT	/users/password(.:format)	passwords#update	**Κάνει αναβαθμηση στο κωδικό**
GET	/profile/:id	show	**Εμφανίζει το view profile**
GET	/todo/:id	show	**Εμφανίζει το view todo**
GET	/home/index	index	**Εμφανιζει όλη τη Home page**
GET	/page/help	page	Εμφάνηση των περιεχόμενων
GET	/page/more	page	Εμφάνηση των περιεχόμενων

ΠΑΡΑΡΤΗΜΑ Γ

|Περίληψη|

Ο βασικός στόχος της πτυχιακής εργασίας είναι η ανάπτυξη και υλοποίηση μιας εύχρηστης διαδικτυακής εφαρμογής η οποία παρέχει δυνατότητες οργάνωσης για την παροχή υπηρεσιών συντήρησης και αντιμετώπισης βλαβών σε θέματα δικτυακών υποδομών και υπηρεσιών τηλεματικής. Η εφαρμογή αναπτύσσετε με βάση τις προϋποθέσεις σε σύστημα truble ticket με τελική κατευθυντήρια τις ανάγκες του Κέντρου Ελέγχου και Διαχείρισης Δικτύων (ΚΕΔΔ) του ΤΕΙ Κρήτης ακολουθούμενη τη μεθοδολογία ανάπτυξης εφαρμογών Ruby on Rails.

Για να αντιμετωπίσουμε το πρόβλημα ορίσαμε ως σύστημα Truble ticket είναι ο μηχανισμός που χρησιμοποιείται από ένα οργανισμό για να ανιχνεύει αναφορές, αιτήσεις ή ανάλυση σε κάποιου είδους προβλήματος. Οι ανάγκες του οργανισμού σχηματίζουν το τύπο και τη πολιτική διαχείρισης των tickets και καθορίζονται από τις απαιτήσεις του. Ο τύπος περιέχει τη κατηγορία που ορίζετε από το κέντρο – οργανισμό ,τις πληροφορίες και τα στοιχεία που συμβάλουν στην άμεση επίλυση του προβλήματος . Η διαχείριση συνήθως έχει άμεση σχέση με το πλήθος των χρηστών και με το καταστατικό που χρησιμοποιούμε για το ticket.

Για την υλοποίηση της εφαρμογής επιλέξαμε το framework Ruby on Rails. Βασίζετε στη γλώσσα προγραμματισμού ruby. Πρόκειται για μια δυναμική ,ανακλαστική και αντικειμενοστραφής γλώσσα. Τα πάντα στη ruby αντιμετωπίζονται ως αντικείμενα και τα αποτελέσματα αυτών τα αντιλαμβάνεται πάλι ως αντικείμενα. Τα αντικείμενα δημιουργούνται με την κλήση ενός κατασκευαστή(constructor) και ένα ειδικό αντικείμενο (κλάση) που σχετίζεται με μια μέθοδο όπως η new().

Οι δυνατότητες της Rails είναι υψηλού επιπέδου και θα καθορίσουν τον τρόπο ανάπτυξης του συστήματος μας. Τα βασικά χαρακτηρίστηκα της είναι: Η αρχιτεκτονική MVC (Model – View - Controller) για την κατασκευή του κώδικα. Η μεθοδολογία και ανάλυση της ανάπτυξης Agile.Η δημιουργία Testing και οι αρχές Dry και CoC.

Το Model View Controller είναι μια από τις αρχιτεκτονικές ανάπτυξης διαδραστικών εφαρμογών. Η δομή του επιτρέπει το διαχωρισμό της εφαρμογής σε τρία επίπεδα - κορμούς το model, το controller και το view. Καθιστά την εργασία μας πιο εύκολη και ξεκαθαρίζουμε καλύτερα τη κάθε λειτουργία του κώδικα μας.

Dont Repeat Your Self (DRY) : Είναι η διαδικασία με την οποία δεν επαναλαμβάνουμε τα ίδια πράγματα πάνω από μια φορά. Η διαδικασία εφαρμόζεται σε πολλά επίπεδα στο κώδικά μέσα στο αρχείο, στο κώδικα που εμφανίζετε ξανά ακόμα και σε διαφορετικά αρχεία.

Agile είναι μεθοδολογία ανάπτυξης λογισμικού, δίνει τη δυνατότητα κατά τη διάρκεια της υλοποίησης να επαναπροσδιορίζουμε τις απαιτήσεις παραδίδοντας κάθε φορά ένα κομμάτι του έργου. Βασική αρχή έχει ότι τα πρόσωπα αλληλεπιδρούν με τα εργαλεία και τη διαδικασία ανάπτυξης και το λογισμικό αναπτύσσεται σύμφωνα με τις απαιτήσεις και είναι έγκυρο σε όλα τα στάδια παράδοσης.

Convention over configuration (CoC) :είναι η διαδικασία όπου ο προγραμματιστής πρέπει να αποφασίσει τι πρέπει να κάνει ώστε να γράφει λιγότερα και ευανάγνωστο κώδικα, για παράδειγμα στη ruby on rails πολύ σημασία έχουν τα ονόματα των αντικειμένων(πχ. Classes στο model και πίνακας στη Βάση Δεδομένων)

Γνωρίζοντας τα παραπάνω ξεκινήσαμε την υλοποίηση από το αρχικό στάδιο προσδιορισμό των αρχικών απαιτήσεων του συστήματος μας. Η ανάλυση εφαρμογής λογισμικού είναι το κομμάτι που πρέπει να γίνεται έστω ένα βήμα πριν την υλοποίηση έτσι για να γνωρίσουμε τις πραγματικές ανάγκες ‘ήρθαμε σε επαφή με το προσωπικό του οργανισμού.

Το Κέντρο Ελέγχου και Διαχείρισης Δικτύων του ΤΕΙ Κρήτης είναι οργανισμός υπεύθυνος για τη δημιουργία και διατήρηση λειτουργίας ποικίλων υπηρεσιών τηλεματικής, η δικτυακή επικοινωνίας μεταξύ των τμημάτων του και τη πρόσβαση στο Διαδίκτυο. Το truble ticket σύστημα είναι αναγκαίο να έχει διαδικτυακή υπόσταση και ηλεκτρονική δημιουργία αιτήσεων (tickets). Οι αιτήσεις δημιουργούνται και δημοσιεύονται από χρήστες μέλη του ΤΕΙ και τεχνικούς υποστήριξης. Η διαχείριση των αιτήσεων απαιτεί και διαχείριση των χρηστών. Ο τύπος των αιτήσεων διακρίνεται σε αιτήσεις τηλεφωνίας, παροχής δικτύου και υπηρεσιών (eclass, blogs κτλ), δηλώσεις βλαβών σε υπάρχουσες υπηρεσίες(πρόσβαση στο διαδίκτυο/eclass) και συσκευές χρήσης (τηλέφωνο, καλωδίωση, κτλ). Δηλώσεις βλαβών του τεχνικού εξοπλισμού (racks, switch, habs, servers, ups κτλ) και εισαγωγή πληροφοριών για ενημέρωση μεταξύ του προσωπικού τεχνικής υποστήριξης για την ενημέρωση των βάσεων δεδομένων.

Ακολουθώντας τη μεθοδολογία ανάπτυξης , οργανώσαμε το περιβάλλον εργασίας και τρόπο εργασίας. Οτιδήποτε θέλουμε ή χρειαζόμαστε να φτιάξουμε το χαρακτηρίζουμε με την αγγλική φράση TODO (= να πράξω). Κάθε TODO είναι ένα μικρό σενάριο χρήσης. Η εφαρμογή υλοποιείτε σε στάδια, αρχικά υλοποιούμε το βασικό κορμό και γίνετε εγκατάσταση σε server.

Ο βασικός κορμός περιέχει τα αρχικά μοντέλα που αλληλεπιδρούν μεταξύ τους για την λειτουργία των διεργασιών του truble ticket. Γίνετε δημιουργία των μοντέλων, σχέσεις των μοντέλων, δημιουργία συνεδρίων χρηστών, ανάπτυξη μεθόδων για την διαχείριση των μοντέλων και των αντικειμένων τους. Εδώ στο Α΄στάδιο υλοποιούμε σενάρια διεργασιών περισσότερο μέσα στο σύστημα.

Μετά το Α΄στάδιο η εφαρμογή γίνετε εγκατάσταση στη πλατφόρμα εφαρμογών heroku ενώ από την στιγμή δημιουργία του έργου διαχειριζόμαστε το ιστορικό ανάπτυξης με το πρόγραμμα Git.

Στο δεύτερο στάδιο η υλοποίηση συνεχίζετε περισσότερο με σενάρια χρήσης της εφαρμογής από τους χρήστες. Επικεντρώνετε κυρίως στη διέπαφή των χρηστών και τις διεργασίες τους προς το σύστημα αφού παρακολουθούν την ανάπτυξη οι απαιτήσεις περνούν συγκεκριμένη μορφή.