Τεχνολογικό Εκπαιδευτικό Ίδρυμα Κρήτης

Σχολή Τεχνολογικών Εφαρμογών

Τμήμα Μηχανικών Πληροφορικής

Πτυχιακή Εργασία

Διαδικτυακή εφαρμογή προσφοράς και ζήτησης εργασίας

Μαυροφοράκης Δημήτρης (ΑΜ: 3455)

Επιβλέπων καθηγητής: Κωστής Αϊβαλής

Ευχαριστίες

Ευχαριστώ την οικογένεια μου για την στήριξη που μου παρείχαν καθόλη την διάρκεια των σπουδών μου.

Abstract

The purpose of this project is to create a web application that will become a channel of communication between employers and prospective employees. Through this platform companies and organizations who are looking for employees can post ads for open job positions. Then users who are interested in these positions can apply by filling out the appropriate form.
Employers through the platform will be able to see the submitted resumes and then providing the required criteria the application will evaluate and recommend which candidate is suitable.

Σύνοψη

Σκοπός αυτής της εργασίας είναι η δημιουργία μιας διαδικτυακής εφαρμογής η οποία θα αποτελέσει έναν δίαυλο επικοινωνίας μεταξύ εργοδοτών και υποψήφιων εργαζομένων. Μέσω αυτής της πλατφόρμας οι επιχειρήσεις και οι οργανισμοί οι οποίοι αναζητούν εργαζομένους θα μπορούν να δημοσιεύουν αγγελίες για ανοικτές θέσεις εργασίας. Στην συνέχεια οι χρήστες που ενδιαφέρονται για αυτές τις θέσεις θα μπορούν να κάνουν αίτηση συμπληρώνοντας την κατάλληλη φόρμα. Οι εργοδότες μέσω της πλατφόρμας θα μπορούν να δουν τα βιογραφικά που έχουν υποβληθεί και στην συνέχεια δίνοντας τα απαιτούμενα κριτήρια αξιολόγησης η εφαρμογή θα αξιολογεί και θα προτείνει ποιός υποψήφιος είναι κατάλληλος.

Περιεχόμενα

Πίνακας Εικόνων

1. Εισαγωγή

1.1 Περίληψη

Σκοπός αυτής της πτυχιακής εργασίας είναι η δημιουργία μιας διαδικτυακής εφαρμογής, μέσω της οποίας οι επιχειρήσεις θα μπορούν να δημοσιεύουν τις διαθέσιμες θέσεις εργασίας τους και οι υποψήφιοι εργαζόμενοι να κάνουν αίτηση σε αυτές. Στην συνέχεια η κάθε αίτηση θα αξιολογείται από την εφαρμογή με βάση τα κριτήρια που δίνει ο κάθε εργοδότης και θα εμφανίζεται σε τι ποσοστό πληροί αυτά τα κριτήρια ο κάθε υποψήφιος. Η υλοποίηση της εφαρμογής θα γίνει σε Java χρησιμοποιώντας το Spring framework. Στο front-end θα χρησιμοποιηθούν Html, CSS, Javascript με τις βιβλιοθήκες JQuery και Bootstrap. Για την βάση δεδομένων θα χρησιμοποιηθεί MySQL.

1.2 Κίνητρο για τη διεξαγωγή της εργασίας

Η διαδικασίας αξιολόγησης των αιτήσεων για μια θέση εργασίας αποτελεί μια χρονοβόρα διαδικασία, ειδικά όταν μια εταιρεία λαμβάνει πολλά βιογραφικά. Μια διαδικτυακή εφαρμογή η οποία θα μπορεί να αυτοματοποιήσει αυτήν την διαδικασία, αξιολογώντας τα προσόντα των υποψηφίων βάσει των απαιτήσεων του εργοδότη, θα μπορούσε να εξοικονομήσει αρκετό χρόνο για την επιχείρηση και να αυξήσει την παραγωγικότητα.

1.3 Σκοπός και στόχοι της εργασίας

Στόχος αυτής της εργασίας είναι η ανάπτυξη μιας εφαρμογής η οποία θα μετατρέπει ένα σύνολο από μη-δομημένες πληροφορίες - όπως το βιογραφικό σημείωμα ενός υποψηφίου – σε δομημένη πληροφορία η οποία θα αποθηκεύεται σε μια βάση δεδομένων. Οι υποψήφιοι δεν θα στέλνουν τα βιογραφικά τους σε word ή PDF αρχεία, αλλά θα συμπληρώνουν μια ειδική φόρμα στην οποία θα αναφέρουν στα προσωπικά τους στοιχεία, την εργασιακή τους εμπειρία, την εκπαίδευση και τις ξένες γλώσσες που γνωρίζουν. Με αυτόν τον τρόπο, η πληροφορία ψηφιοποιείται και έχουμε την δυνατότητα να κάνουμε υπολογιστικές πράξεις, ανάλυση δεδομένων και εξαγωγή στατιστικών.

1.4 Δομή της εργασίας

Στο 1^{ο} κεφάλαιο αναφέρεται η περίληψη, το κίνητρο και ο σκοπός της εργασίας. Στο 2^{ο} κεφάλαιο γίνεται μια σύντομη εισαγωγή στην Java, τα frameworks της, την MVC αρχιτεκτονική. Στο 3^{ο} κεφάλαιο θα δούμε τις τεχνολογίες που θα χρησιμοποιηθούν και στο 4^{ο} κεφάλαιο την υλοποίηση της εφαρμογής.

2. Μεθοδολογία Υλοποίησης

2.1 Μέθοδος Ανάλυσης και ανάπτυξης εφαρμογής

Για την ανάπτυξη της εφαρμογής χρειαζόμαστε μια γλώσσα προγραμματισμού η οποία θα υποστηρίζεται από web, desktop και mobile. Η κάθε επιχείρηση μπορεί να έχει διαφορετικές απαιτήσεις γι αυτό είναι αναγκαίο να επιλέξουμε μια γλώσσα η οποία δεν θα μας δεσμεύει με κάποιο συγκεκριμένο λειτουργικό σύστημα ή πλατφόρμα.

2.2 Η γλώσσα προγραμματισμού JAVA

Η Java είναι μια αντικειμενοστραφής γλώσσα προγραμματισμού που σχεδιάστηκε από την εταιρεία Sun Microsystems. Ένα από τα βασικά πλεονεκτήματα της Java έναντι των περισσότερων άλλων γλωσσών είναι η ανεξαρτησία του λειτουργικού συστήματος και πλατφόρμας. Τα προγράμματα που είναι γραμμένα σε Java τρέχουν ακριβώς το ίδιο σε Windows, Linux, Unix και Macintosh χωρίς να χρειαστεί να ξαναγίνει μεταγλώττιση (compiling) ή να αλλάξει ο πηγαίος κώδικας για κάθε διαφορετικό λειτουργικό σύστημα. Για να επιτευχθεί όμως αυτό χρειαζόταν κάποιος τρόπος έτσι ώστε τα προγράμματα γραμμένα σε Java να μπορούν να είναι «κατανοητά» από κάθε υπολογιστή ανεξάρτητα του είδους επεξεργαστή αλλά και λειτουργικού συστήματος. Ο λόγος είναι ότι κάθε κεντρική μονάδα επεξεργασίας κατανοεί διαφορετικό κώδικα μηχανής. Ο συμβολικός κώδικας (assembly) που μεταφράζεται και εκτελείται σε Windows είναι διαφορετικός από αυτόν που μεταφράζεται και εκτελείται σε έναν υπολογιστή Macintosh. Η λύση δόθηκε με την ανάπτυξη της Εικονικής Μηχανής (Virtual Machine ή VM)[1].

Η εικονική μηχανή της Java

Αφού γραφεί κάποιο πρόγραμμα σε Java, στη συνέχεια μεταγλωτίζετται μέσω του μεταγλωττιστή javac, ο οποίος παράγει έναν αριθμό από αρχεία .class (κώδικας byte ή bytecode). Ο κώδικας byte είναι η μορφή που παίρνει ο πηγαίος κώδικας της Java όταν μεταγλωττιστεί. Όταν πρόκειται να εκτελεστεί η εφαρμογή σε ένα μηχάνημα, το Java Virtual Machine που πρέπει να είναι εγκατεστημένο σε αυτό θα αναλάβει να διαβάσει τα αρχεία .class. Στη συνέχεια τα μεταφράζει σε γλώσσα μηχανής που να υποστηρίζεται από το λειτουργικό σύστημα και τον επεξεργαστή, έτσι ώστε να εκτελεστεί (αυτό συμβαίνει με την παραδοσιακή Εικονική Μηχανή (Virtual Machine). Πιο σύγχρονες εφαρμογές της εικονικής Μηχανής μπορούν και μεταγλωττίζουν εκ των προτέρων τμήματα bytecode απευθείας σε κώδικα μηχανής (εγγενή κώδικα ή native code) με αποτέλεσμα να βελτιώνεται η ταχύτητα). Χωρίς αυτό δε θα ήταν δυνατή η εκτέλεση λογισμικού γραμμένου σε Java. Η JVM είναι λογισμικό που εξαρτάται από την πλατφόρμα, δηλαδή για κάθε είδος λειτουργικού συστήματος και αρχιτεκτονικής επεξεργαστή υπάρχει διαφορετική έκδοση του. Έτσι υπάρχουν διαφορετικές JVM για Windows, Linux, Unix, Macintosh, κινητά τηλέφωνα, παιχνιδομηχανές κλπ.

Garbage collector

Ακόμα μία ιδέα που βρίσκεται πίσω από τη Java είναι η ύπαρξη του συλλέκτη απορριμμάτων (Garbage Collector). Συλλογή απορριμμάτων είναι μία κοινή ονομασία που χρησιμοποιείται στον τομέα της πληροφορικής για να δηλώσει την ελευθέρωση τμημάτων μνήμης από δεδομένα που δε χρειάζονται και δε χρησιμοποιούνται άλλο. Αυτή η απελευθέρωση μνήμης στη Java είναι αυτόματη και γίνεται μέσω του συλλέκτη απορριμμάτων. Υπεύθυνη για αυτό είναι και πάλι η εικονική μηχανή η οποία μόλις «καταλάβει» ότι ο σωρός (heap) της μνήμης (στη Java η συντριπτική πλειοψηφία των αντικειμένων αποθηκεύονται στο σωρό σε αντίθεση με τη C++ όπου αποθηκεύονται κυρίως στη στοίβα) κοντεύει να γεμίσει ενεργοποιεί το συλλέκτη απορριμμάτων. Έτσι ο προγραμματιστής δε χρειάζεται να ανησυχεί για το πότε και αν θα ελευθερώσει ένα συγκεκριμένο τμήμα της μνήμης, ούτε και για σφάλματα δεικτών. Αυτό είναι ιδιαίτερα σημαντικό γιατί είναι κοινά τα σφάλματα προγραμμάτων που οφείλονται σε λανθασμένο χειρισμό της μνήμης.[1]

Πλεονεκτήματα της JAVA:

- Υποστήριξη σε πολλές πλατφόρμες
- Αυτόματη διαχείριση μνήμης. Ο Garbage Collector φροντίζει να αδειάζει την μνήμη από αντικείμενα δεν χρειάζονται.
- Ταχύτητα
- Δημοτικότητα

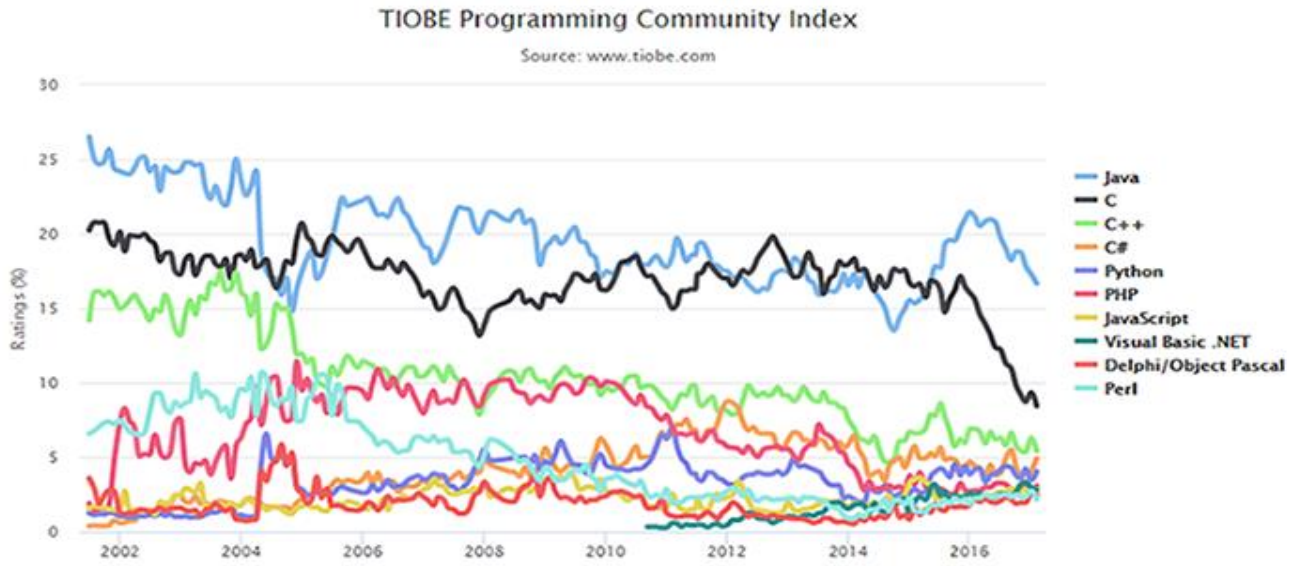
Μειονεκτήματα

- Δεν υποστηρίζεται πολλαπλή κληρονομικότητα
- Υψηλότερο κόστος ανάπτυξης λογισμικού σε σχέση με άλλες γλώσσες

Δημοτικότητα της Java

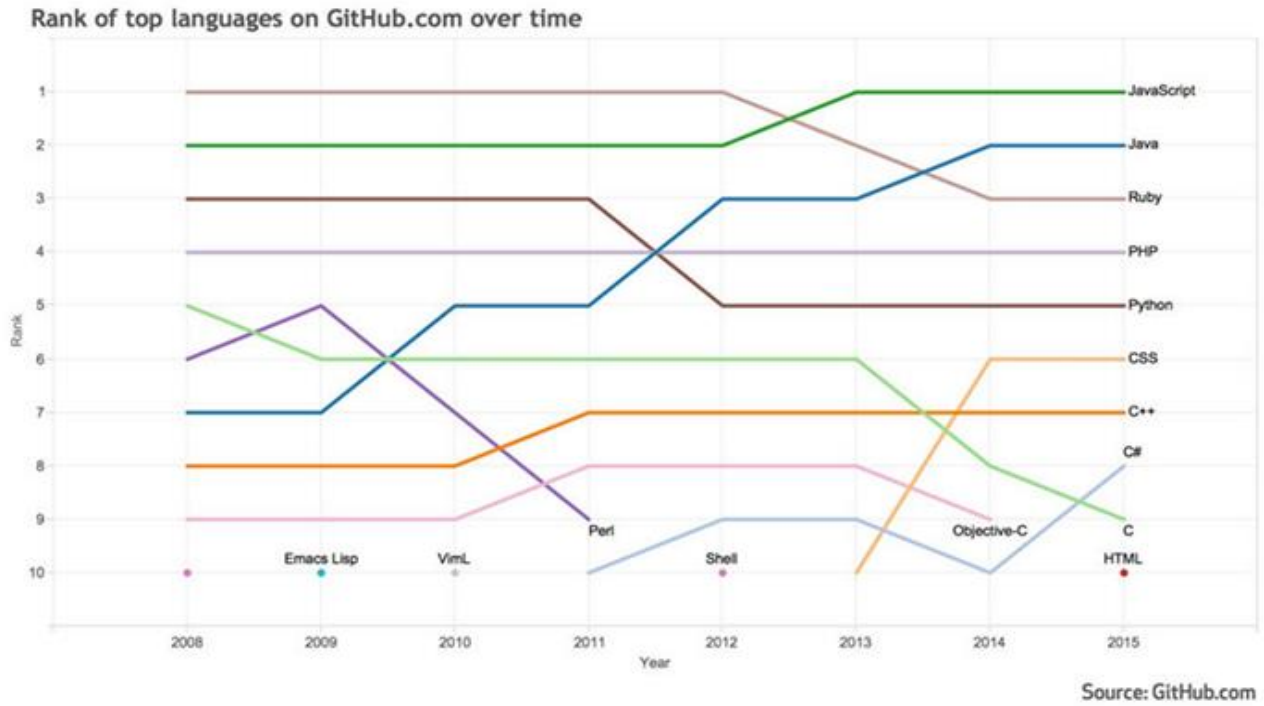
- ***TIOBE Programming Community index***:

Είναι ένας δείκτης για την δημοτικότητα των γλωσσών προγραμματισμού ο οποίος ανανεώνεται κάθε μήνα. Η κατάταξη βασίζεται στο πόσες γραμμές κώδικα έχουν γραφτεί για την κάθε γλώσσα. Οι υπολογισμοί γίνονται χρησιμοποιώντας μηχανές αναζήτησης όπως οι Google, Bing, Yahoo, Wikipedia.



Εικόνα 1 – δημοτικότητα γλωσσών προγραμματισμού
Πηγή: http://www.tiobe.com/tiobe-index

- **Κατάταξη με βάση τα αποθετήρια (Repositories) στο GitHub**



Εικόνα 2 – Αποθετήρια στο GitHub ανά γλώσσα προγραμματισμού

2.3 Java frameworks

2.3.1.Τι είναι ένα framework

Ένα framework (πλαίσιο εργασίας) αποτελεί τον «σκελετό» για την δημιουργία μιας εφαρμογής. Αποτελείται από βιβλιοθήκες και έτοιμα κομμάτια κώδικα πάνω στα οποία θα δουλέψει ο προγραμματιστής για να χτίσει την εφαρμογή του. Χρησιμοποιώντας framework, μειώνεται δραστικά ο χρόνος που απαιτείται για να αναπτυχθεί μια διαδικτυακή εφαρμογή, καθώς ο προγραμματιστής δεν ξεκινάει από το μηδέν.

2.3.2. Πλεονεκτήματα χρήσης framework

- Ταχύτερη ανάπτυξη εφαρμογής

Ένα framework αποτελείται από μεγάλες συλλογές με βιβλιοθήκες οι οποίες μπορούν να αυτοματοποιήσουν πολλές από τις εργασίες που πρόκειται να εκτελέσει ο προγραμματιστής και κατά συνέπεια να αυξηθεί η παραγωγικότητα του. Με αυτόν τον τρόπο ο προγραμματιστής δεν χρειάζεται να ξεκινήσει να γράφει κώδικα από το μηδέν. Για παράδειγμα, ένα framework μπορεί να έχει έτοιμες κλάσεις για σύνδεση και διαχείριση της βάσης δεδομένων.

- Μεγαλύτερη ασφάλεια

Τα χαρακτηριστικά ασφαλείας όπως είναι η πιστοποίηση χρηστών και η διαχείριση των δικαιωμάτων τους διαχειρίζονται από το Framework. Οι εγγραφές στη βάση δεδομένων είναι ασφαλείς, χωρίς SQL Injections.

- Υποστήριξη από την κοινότητα

Τα frameworks υποστηρίζονται από κοινότητες, forums, κανάλια IRC και άλλα. Οποιοσδήποτε αντιμετωπίζει κάποιο πρόβλημα με το framework μπορεί να απευθυνθεί άμεσα στην κοινότητα και να λάβει βοήθεια. Επίσης μπορεί να βρει άμεσα την απάντηση στο πρόβλημα του χωρίς καν να ρωτήσει, εάν κάποιος άλλος αντιμετώπισε παρόμοιο πρόβλημα στο παρελθόν και λύθηκε από την κοινότητα.

- Επεκτάσεις και μονάδες

Κάποια από τα μέλη που συμμετέχουν στις κοινότητες υποστήριξης δημοσιεύουν δωρεάν επεκτάσεις για το framework. Τέτοιες επεκτάσεις παρέχουν διασύνδεση με άλλες εφαρμογές μέσω κάποιου API κ.λ.π.

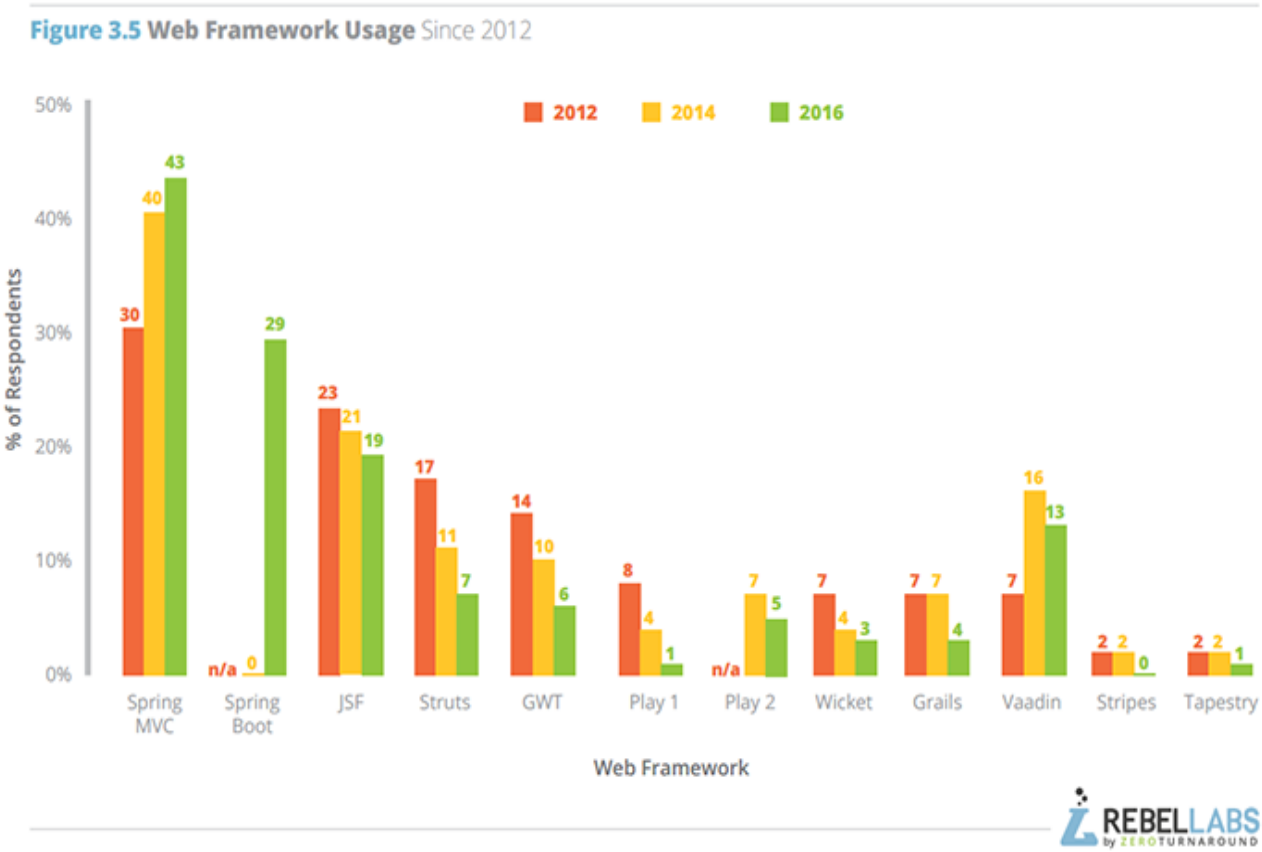
- Χρήση καλών μοντέλων προγραμματισμού

Τα περισσότερα Frameworks υποχρεώνουν τη χρήση του μοντέλο προγραμματισμού Model View Controller (MVC). Αυτό μας επιτρέπει να γνωρίζουμε πως θα δομηθεί ο κώδικας πριν ακόμα αρχίσουμε να γράφουμε, κάνοντας έτσι το τελικό προϊόν ποιο ποιοτικό.

- Ευκολία στον κώδικα

Τα Frameworks είναι πολύ καλά τεκμηριωμένα και δοκιμασμένα. Αν κάποια στιγμή χρειαστεί να έρθει ένας νέος προγραμματιστής σε μια ομάδα θα πρέπει να διαβάσει μόνο την τεκμηρίωση του Framework, και από εκεί και πέρα να κατανοήσει όλη την εφαρμογή. Στην περίπτωση που δεν έχει χρησιμοποιηθεί κάποιο Framework, τότε θα έπρεπε να καταναλώσει πολύ χρόνο και χρήμα ο νέος developer για να εκπαιδευτεί στο νέο σύστημα.[3]

Δημοτικότητα Java frameworks



Τα παραπάνω αποτελέσματα προέρχονται από έρευνα της ZeroTurnaround για το 2016, στην οποία συμμετείχαν πάνω από 2000 developers.[2]

Εικόνα 3 – Δημοτικότητα Java frameworks

2.4 Η αρχιτεκτονική MVC

Σκοπός των περισσότερων υπολογιστικών συστημάτων είναι να ανακτούν δεδομένα από μια πηγή και να τα εμφανίζουν στο χρήστη. Όταν ο χρήστης εισάγει δεδομένα στο σύστημα, αυτό ενημερώνει την πηγή με τα νέα δεδομένα. Αφού, η ανταλλαγή πληροφορίας στο σύστημα γίνεται ανάμεσα στην πηγή δεδομένων και στη διεπαφή χρήστη, το λογικό θα ήταν να συνδεθούν τα δυο αυτά τμήματα, ώστε να μειωθεί το μέγεθος του κώδικα του συστήματος αλλά και για να αυξηθεί η απόδοση της εφαρμογής.

Βέβαια, αυτή η προσέγγιση παρουσιάζει κάποια προβλήματα στην πράξη. Ένα πρόβλημα είναι ότι η διεπαφή χρήστη έχει την τάση να αλλάζει περισσότερες φορές σε σχέση με το σύστημα αποθήκευσης δεδομένων. Ένα άλλο πρόβλημα της σύνδεσης της διεπαφής χρήστη με την πηγή δεδομένων είναι ότι οι εφαρμογές έχουν την τάση να ενσωματώνουν τη "λογική" του συστήματος (application/business logic), και ειδικά στις εφαρμογές Παγκόσμιου Ιστού, η διεπαφή χρήστη αλλάζει πιο συχνά από τη "λογική". Για παράδειγμα, μπορεί να προστεθούν νέες σελίδες στη διεπαφή ή να τροποποιηθούν ήδη υπάρχοντα στυλ απεικόνισης μιας εφαρμογής. Αν η διεπαφή χρήστη και η "λογική" είναι συνδεδεμένες σε ένα ενιαίο τμήμα/αντικείμενο της εφαρμογής πρέπει, κάθε φορά που απαιτείται μια αλλαγή στη διεπαφή χρήστη, να τροποποιείται και όλο το τμήμα που περιέχει τη "λογική". Αυτό είναι πιθανό να εισάγει λάθη στην εφαρμογή και απαιτεί κάθε φορά επανέλεγχο όλου του τμήματος της "λογικής" μετά από κάθε, έστω και μικρή, αλλαγή της διεπαφής χρήστη

Η Model-View-Controller (MVC) (Reenskaug, 1979; Reenskaug 2003)αρχιτεκτονική λύνει το παραπάνω πρόβλημα διαχωρίζοντας την απεικόνιση των δεδομένων, της εφαρμογής που βασίζονται στην είσοδο δεδομένων από το χρήστη και τον τρόπο αποθήκευσης των δεδομένων σε τρία διακριτά τμήματα/αντικείμενα. Πιο συγκεκριμένα, το αντικείμενο Controller μεταφράζει την είσοδο που δέχεται από το χρήστη (συνήθως κλικ ποντικιού ή πληκτρολόγηση), και ενημερώνει το αντικείμενο Model ή/και το αντικείμενο View για να τροποποιηθούν κατάλληλα. Το View διαχειρίζεται την απεικόνιση των δεδομένων, ενώ το Model "κρατά" τη δομή των δεδομένων, απαντά στα "αιτήματα" για ενημέρωση των δεδομένων που δέχεται από το Controller, ενώ δέχεται και "αιτήματα" για ανάκτηση δεδομένων, συνήθως από το View.

Πρέπει να σημειωθεί ότι τα αντικείμενα View και Controller εξαρτώνται από το Model, ενώ το Model δεν εξαρτάται ούτε από το View ούτε από το Controller. Αυτό είναι ένα από τα βασικά πλεονεκτήματα όταν διαχωρίζονται οι κύριες λειτουργίες μιας εφαρμογής. Ο διαχωρισμός αυτός επιτρέπει στο Model να αναπτυχθεί και να ελεγχθεί ανεξάρτητα από τα υπόλοιπα, χωρίς δηλαδή να είναι απαραίτητη η ανάπτυξη λειτουργίας που θα απεικονίζει τα δεδομένα (View).[4]

Βασικά Πλεονεκτήματα MVC

- Διαχωρισμός Προβλημάτων (Separation of Concerns).

Αυτό είναι και το πιο βασικό πλεονέκτημα του MVC. Ουσιαστικά δημιουργείται μία εφαρμογή η οποία έχει τρία επίπεδα, το επίπεδο των models , το επίπεδο των controllers και το επίπεδο των views -που θα αναλυθούν παρακάτω- και το κάθε επίπεδο επιτελεί ξεχωριστό έργο και ταυτόχρονα συνεργάζεται με τα άλλα επίπεδα. Μία σωστή MVC εφαρμογή είναι εκείνη που τα τρία επίπεδα είναι ξεκάθαρα καθορισμένα και δεν συμπλέκονται. Για παράδειγμα είναι λάθος στο επίπεδο των View να υπάρχει κώδικας που «μιλάει» με την βάση δεδομένων και «τραβάει» δεδομένα.

- Επεκτασιμότητα.

Το δεύτερο πλεονέκτημα της MVC αρχιτεκτονικής είναι πολύ σημαντικό επίσης. «Επεκτασιμότητα» είναι η δυνατότητα που διαθέτει μία εφαρμογή , κατά την οποία μπορούμε μελλοντικά να προσθέσουμε λειτουργίες σε αυτή ή να αλλάξουμε κάποιες από τις ήδη υπάρχουσες λειτουργίες και να έχουμε άλλα αποτελέσματα. Για να το δούμε εντελώς απλά και κατανοητά, η πλατφόρμα WordPress είναι επεκτάσιμη με τη χρήση των διάφορων plugins διότι προσθέτουμε στις ήδη υπάρχουσες λειτουργίες και άλλες λειτουργίες. Τα προγράμματα που είναι φτιαγμένα με MVC αρχιτεκτονική έχουν βασικό χαρακτηριστικό ότι είναι επεκτάσιμα.

- Ελεγξιμότητα (Testability)

.Αυτό είναι ένα πολύ κρίσιμο χαρακτηριστικό. Οι MVC εφαρμογές έχουν την δυνατότητα να είναι ελέγξιμες και με τον τρόπο αυτό συντηρούνται πιο εύκολα. Ας κάνουμε ένα απλό παράδειγμα. Έστω ότι έχουμε μία εφαρμογή η οποία διαθέτει μία λειτουργία login, δηλαδή ζητά από τον χρήστη να πληκτρολογήσει κάποια στοιχεία σε μία φόρμα και εν συνέχεια τον εισάγει μέσα στο σύστημα. Αυτή τη λειτουργία την ελέγχει κάποιος loginController ο οποίος περιέχει κώδικα που διαχειρίζεται τα δεδομένα αυτά που εισήχθησαν από τον χρήστη. Αυτός ο controller θεωρείται μία «μονάδα» ή αλλιώς unit. Στα MVC frameworks μπορούμε με πολλή ευκολία να γράψουμε απλό κώδικα-tests με τον οποίο τεστάρουμε αυτόν τον controller

αλλά και κάθε μία από τις λειτουργίες του. Παίρνουμε τα αποτελέσματα και βλέπουμε η συγκεκριμένη μονάδα της εφαρμογής μας λειτουργεί σωστά.

- «Καθαρά» URLs.

Τα περισσότερα MVC frameworks για web applications δίνουν τη δυνατότητα να έχουμε «καθαρά» urls. Ας κάνουμε ένα παράδειγμα. Έστω ότι έχουμε ένα blog και πατάμε το link για να διαβάσουμε ένα άρθρο. Ένα τυπικό URL θα μπορούσε να ήταν http://webapptester.com/mvc-framework-first-impression/ [5]

3. Σχέδιο δράσης για την εκπόνηση της πτυχιακής

Σε αυτό το κεφάλαιο θα γίνουν αναφορές στις τεχνολογίες αιχμής (state of the art) που θα χρησιμοποιηθούν για την υλοποίηση της εργασίας

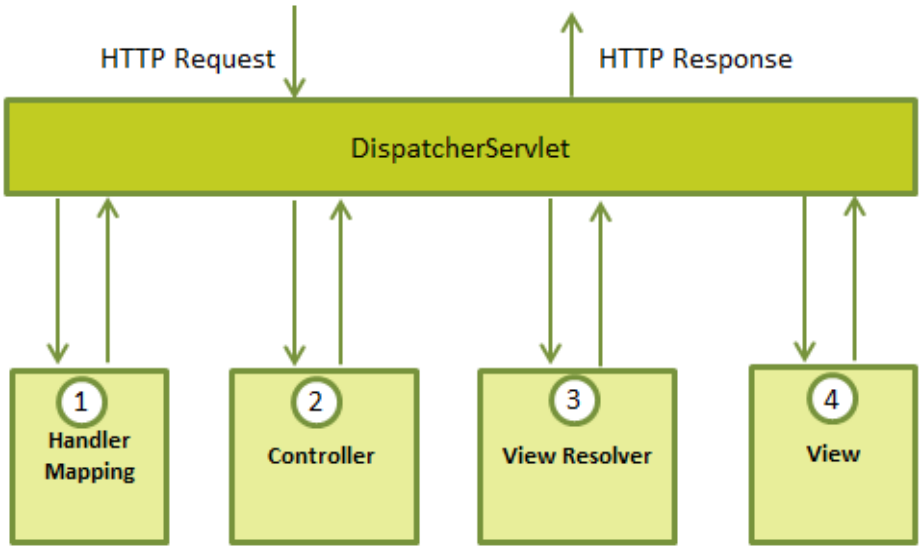
3.1 Spring MVC

Θεωρείται ως το πιο διαδεδομένο framework για εφαρμογές java. Είναι λογισμικό ανοικτού κώδικα(open source) και η πρώτη του έκδοση κυκλοφόρησε το 2003 από τον Rod Johnson. Είναι βασισμένο σε αρχιτεκτονική model-view-controller και χρησιμοποιείται για την ανάπτυξη ευέλικτων διαδικτυακών εφαρμογών. Το μοντέλο MVC διαχωρίζει τις πτυχές μιας εφαρμογές σε 3 στοιχεία: λογική εισόδου (input logic), επιχειρησιακή λογική (business logic) και διεπαφή χρήστη (UI logic).[6]

- Model: ενθυλακώνει τα δεδομένα της εφαρμογής
- View: Υπεύθυνο για την αναπαράσταση (rendering) των δεδομένων του model. Παράγει ουσιαστικά το html που θα λάβει ο φυλλομετρητής (browser) του χρήστη.
- Controller: Είναι υπεύθυνος για την διαχείριση των requests που στέλνει ο χρήστης. Χτίζει το κατάλληλο model και το στέλνει στο view για rendering.

Dispatcher Servlet

Το Spring web MVC είναι σχεδιασμένο πάνω σε έναν Dispatcher Servlet ο οποίος διαχειρίζεται όλα τα HTTP requests και responses. Η επεξεργασία των requests αναπαριστάται στην παρακάτω εικόνα



Εικόνα 4 – Dispatcher Servlet

Από την στιγμή που ο dispatcher servlet θα λάβει ένα HTTP request, θα ακολουθήσουν οι παρακάτω ενέργειες:

- HandlerMapping: Θα κληθεί ο κατάλληλος controller για να διαχειριστεί το request
- Ο controller θα λάβει το request και θα καλέσει την κατάλληλη μέθοδο-υπηρεσία (service method) ανάλογα το είδος του request (POST ή GET). Το service-method θα δημιουργήσει τα model data βασισμένο στο business logic και θα επιστρέψει το όνομα του view στον DispatcherServlet.
- Ο DispatcherServlet με την βοήθεια του ViewResolver θα χρησιμοποιήσει το κατάλληλο view.
- Ο DispatcherServlet περνάει τα δεδομένα του model στο view, το οποίο τελικώς θα εμφανιστεί στον browser του χρήστη.

Απαιτούμενες Ρυθμίσεις

Web.xml

Θα πρέπει να ορίσουμε ποιοι σύνδεσμοι (URLs) θα διαχειρίζονται και από ποιό DispatcherServlet. Μπορούμε να έχουμε πολλά DipatcherServlets και να καλείται διαφορετικό σε κάθε περίπτωση. Αυτή η διαδικασία ονομάζεται URL mapping και καθορίζεται στο αρχείο **web.xml.**

Το web.xml (Deployment descriptor) είναι το βασικό αρχείο μιας web εφαρμογής και περιέχει όλες τις απαραίτητες πληροφορίες που χρειάζεται να γνωρίζει ο server.

Για παράδειγμα:

```
<web-app id="WebApp_ID" version="2.4"
    xmlns="http://java.sun.com/xml/ns/j2ee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
    http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">

    <display-name>Spring MVC Application</display-name>

   <servlet>
      <servlet-name>dispatcher</servlet-name>
      <servlet-class>
         org.springframework.web.servlet.DispatcherServlet
      </servlet-class>
      <load-on-startup>1</load-on-startup>
   </servlet>

   <servlet-mapping>
      <servlet-name>HelloWeb</servlet-name>
      <url-pattern>/</url-pattern>
   </servlet-mapping>

</web-app>
```

Το αρχείο web.xml βρίσκεται στον φάκελο WebContent/WEB-INF. Όταν η εφαμοργή μας λάβει ένα HTTP request το οποίο ταιριάζει σε κάποιο pattern που έχουμε ορίσει – στην συγκεκριμένη περίπτωση οποιοδήποτε URL τελειώνει σε .jsp, τότε θα ψάξει για ένα αρχείο της μορφής [servlet-name]-servlet.xml στο φάκελο WebContent/WEB-INF. Στο παραπάνω παράδειγμα λοιπόν, όταν λάβει ένα URL με κατάληξη .jsp, θα ψάξει για το αρχείο **HelloWeb-servlet.xml,** πράγμα που σημαίνει ότι θα κληθεί ο **HelloWeb** DispatcherServlet**.**

Παράδειγμα δημιουργίας Spring project στο Eclipse

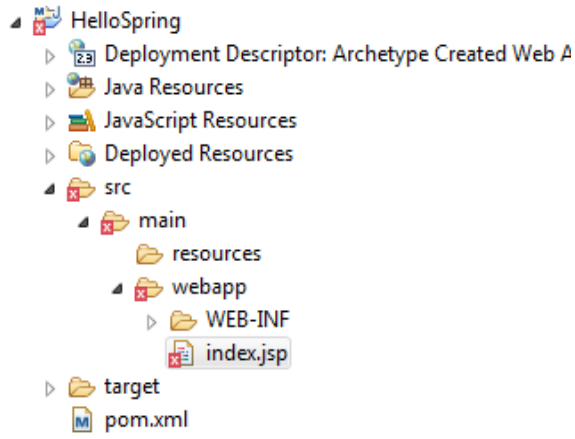
1. Δημιουργία Maven Project

Α. Επιλέγουμε File->New->Other…->Maven/Maven Project και στην συνέχεια δίνουμε το path που θα αποθηκευτεί το project.

Β. Στο achetype επιλέγουμε webapp

Γ. Δίνουμε τιμές στα Group Id και Artifact Id (π.χ. HelloSpring)

Παρακάτω βλέπουμε την δομή του maven project που δημιουργήσαμε:



2. Προσθήκη Spring dependencies στο **pom.xml**

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.thesis</groupId>
  <artifactId>HelloSpring</artifactId>
  <packaging>war</packaging>
  <version>0.0.1-SNAPSHOT</version>
  <name>HelloSpring Maven Webapp</name>
  <url>http://maven.apache.org</url>
       <properties>
              <spring.version>4.0.1.RELEASE</spring.version>
       </properties>
       <dependencies>
              <dependency>
                     <groupId>junit</groupId>
                     <artifactId>junit</artifactId>
                     <version>3.8.1</version>
                     <scope>test</scope>
              </dependency>
              <!-- Spring dependencies -->
              <dependency>
                     <groupId>org.springframework</groupId>
                     <artifactId>spring-core</artifactId>
                     <version>${spring.version}</version>
              </dependency>

              <dependency>
                     <groupId>org.springframework</groupId>
                     <artifactId>spring-web</artifactId>
                     <version>${spring.version}</version>
              </dependency>

              <dependency>
                     <groupId>org.springframework</groupId>
                     <artifactId>spring-webmvc</artifactId>
                     <version>${spring.version}</version>
              </dependency>
       </dependencies>
  <build>
    <finalName>HelloSpring</finalName>
            <plugins>
             <plugin>
                     <groupId>org.apache.tomcat.maven</groupId>
                     <artifactId>tomcat7-maven-plugin</artifactId>
                     <version>2.2</version>
             </plugin>
       </plugins>
  </build>
</project>
```

3. Ορισμός DispatcherServlet

Στο αρχείο **web.xml** ορίζουμε τον DispatcherServlet καθώς και το mapping του. Στην προκειμένη περίπτωση όλα τα HTTP requests θα διαχειρίζονται από το servlet με το όνομα dispatcher:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd" id="WebApp_ID" version="3.0">
	<display-name>Archetype Created Web Application</display-name>
	<servlet>
		<servlet-name>dispatcher</servlet-name>
		<servlet-class>
			org.springframework.web.servlet.DispatcherServlet
		</servlet-class>
		<load-on-startup>1</load-on-startup>
	</servlet>
	<servlet-mapping>
		<servlet-name>dispatcher</servlet-name>
		<url-pattern>/</url-pattern>
	</servlet-mapping>
</web-app>
```

4.Δημιουργούμε το αρχείο **dispatcher-servlet.xml** στον φάκελο WEB-INF.

```
<beans xmlns="http://www.springframework.org/schema/beans"
	xmlns:context="http://www.springframework.org/schema/context"
	xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
	xsi:schemaLocation="
http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-3.0.xsd">

	<context:component-scan base-package="com.thesis.controller" />

	<bean

	class="org.springframework.web.servlet.view.InternalResourceViewResolver">
		<property name="prefix">
			<value>/WEB-INF/views/</value>
		</property>
		<property name="suffix">
			<value>.jsp</value>
		</property>
	</bean>
</beans>
```

4. Δημιουργία controllers

Δημιουργούμε τον φάκελο java μεσα στον φάκελο main. Στην συνέχεια μέσα στον φάκελο java δημιουργούμε το package com/thesis/controller όπου θα αποθηκεύουμε τους controllers της εφαρμογής μας.

HelloWorldController.java :

```
package com.thesis.controller;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.servlet.ModelAndView;

@Controller
public class HelloWorldController {

	@RequestMapping("/hello")
	public ModelAndView showMessage() {
		ModelAndView mv = new ModelAndView("helloworld");
		String message = "Hello world!";
		mv.addObject("message", message);
		return mv;
	}

}
```

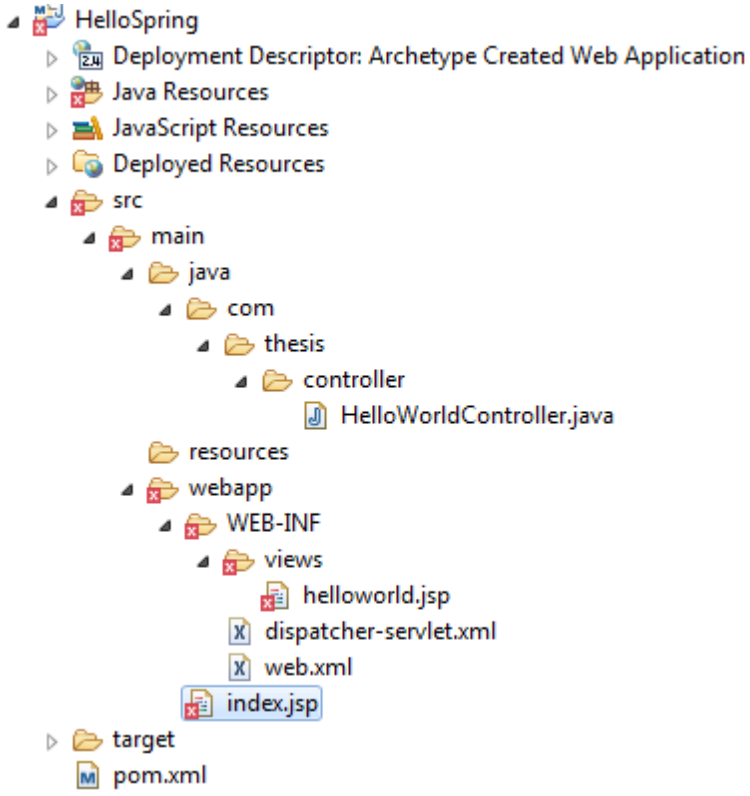
Το @RequestMapping annotation αντιστοιχεί συγκεκριμένα requests σε κάποια συγκεκριμένη κλάση ή μέθοδο. Στο παραπάνω παράδειγμα, τα requests της μορφής /hello καλούν την μέθοδο showMessage() .
Στην συνέχεια, δημιουργούμε ένα αντικείμενο τύπου ModelAndView, με όρισμα στον constructor το όνομα του view, το οποίο στην συνέχεια θα αναλυθεί από το ViewResolver. Στο παραπάνω παράδειγμα, το view που θα κληθεί θα είναι το /WEB-INF/views/helloworld.jsp. Καλώντας την μέθοδο addObject() μπορούμε να προσθέτουμε αντικείμενα στο view.

4.Δημιουργία view

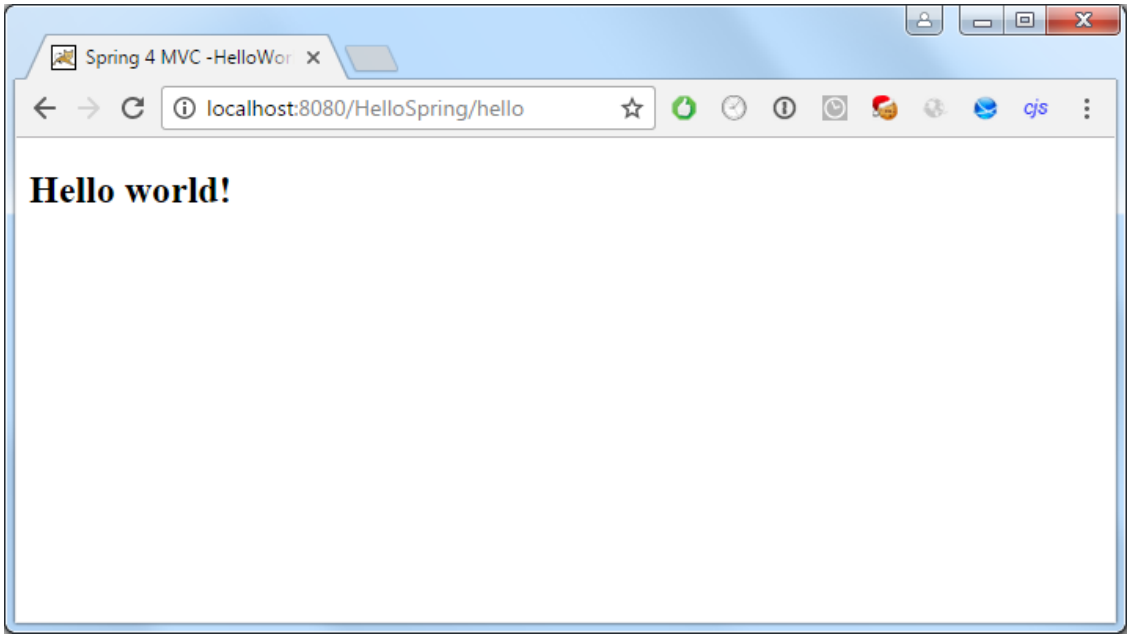
helloworld.jsp:

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Spring 4 MVC -HelloWorld</title>
</head>
<body>
	<h2>${message} </h2>
</body>
</html>
```

Τελική μορφή του project:



Για να τρέξουμε το project κάνουμε δεξί κλίκ run as->maven build και στο goals δίνουμε την τιμή **tomcat7:run**



3.2 Hibernate Framework

Το Hibernate Framework είναι λογισμικό ανοιχτού κώδικα (ελεύθερο λογισμικό) που σκοπό έχει να συνδέσει τα αντικείμενα που δημιουργούνται σε μια αντικειμενοστραφή γλώσσα προγραμματισμού (Java) με τους πίνακες μιας σχεσιακής βάσης δεδομένων. Η σύνδεση αυτή επιτυγχάνεται με την χρήση επιπρόσθετης πληροφορίας (metadata) που τοποθετείται κατάλληλα (μαζί με τον κώδικα Java ή σε ξεχωριστά xml αρχεία) και περιγράφει την αντιστοιχία μεταξύ των αντικειμένων και της βάσης δεδομένων. Γενικά το Hibernate προσφέρει την αυτόματη μετατροπή της μιας μορφής (αντικείμενα) στην άλλη (σχεσιακή βάση δεδομένων).

Το Hibernate συγκαταλέγεται στην κατηγορία του λογισμικού ORM (object/relational mapping). Το λογισμικό ORM στοχεύει στην δημιουργία μιας διεπαφής (interface) μεταξύ των διαδεδομένων σχεσιακών βάσεων δεδομένων και του αντικειμενοστραφούς προγραμματισμού. Με απλά λόγια, προσφέρει την χρησιμοποίηση μιας σχεσιακής βάσης δεδομένων σαν να ήταν αντικειμενοστραφής. Για να το επιτύχει αυτό δημιουργεί αντιστοιχίες μεταξύ των εννοιών του αντικειμενοστραφούς προγραμματισμού (συσχετίσεις, κληρονομικότητα, πολυμορφισμός) - που δεν υπάρχουν σε μια σχεσιακή βάση δεδομένων - και των πινάκων και σχέσεων μεταξύ των πινάκων μιας σχεσιακής βάσης. Με αυτό τον τρόπο ο προγραμματιστής βλέπει τελικά μια αντικειμενοστραφή βάση δεδομένων, παρ΄όλο που στην ουσία χρησιμοποιεί μια σχεσιακή. Έτσι ο προγραμματιστής χρησιμοποιεί τα αντικείμενα της συγκεκριμένης εφαρμογής, τα τροποποιεί σχετικά με τη λογική της εφαρμογής που αναπτύσσει και τα αποθηκεύει (τροποποιεί, διαγράφει και αναζητά) στην βάση ως αντικείμενα, σκεπτόμενος δηλαδή με αντικειμενοστραφείς έννοιες και όχι με βάση το σχήμα της σχεσιακής βάσης δεδομένων. Σε αυτό το σημείο είναι το Hibernate που, γνωρίζοντας την αντιστοιχία μεταξύ βάσης και λογικής της εφαρμογής, αναλαμβάνει να κατασκευάσει την κατάλληλη εντολή της SQL η οποία και στέλνεται τελικά στην βάση δεδομένων. Έπειτα, τα αποτελέσματα που επιστρέφει η βάση το Hibernate τα επιστρέφει στον προγραμματιστή ως αντικείμενα της εφαρμογής. Είναι δηλαδή ένα ενδιάμεσο επίπεδο μεταξύ της εφαρμογής και της βάσης δεδομένων.

Χαρακτηριστικά/Πλεονεκτήματα

- Παραγωγικότητα: Στην ανάπτυξη λογισμικού ένα μεγάλο μέρος της προγραμματιστικής προσπάθειας αφιερώνεται στην διεπαφή της εφαρμογής με τη βάση δεδομένων. Το Hibernate αυτοματοποιώντας τις βασικές λειτουργίες Δημιουργία/Ανάγνωση/Τροποποίηση/Διαγραφή (CRUD – Create Read Update Delete) επιτρέπει αρχικά στον προγραμματιστή να επικεντρώνει την προσπάθειά του στη λογική της εφαρμογής (business logic). Επίσης, υπάρχει η δυνατότητα να ακολουθηθούν δύο στρατηγικές ανάπτυξης λογισμικού: είτε αρχίζοντας από το μοντέλο δεδομένων είτε από τη βάση δεδομένων. Αυτό μειώνει σε μεγάλο βαθμό το χρόνο ανάπτυξης.

- Συντηρησιμότητα: Με τη χρήση του Hibernate γράφονται σημαντικά λιγότερες γραμμές κώδικα και ο κώδικας είναι πιο κατανοητός και καλογραμμένος. Αυτό κάνει την συντήρηση της εφαρμογής ευκολότερη.
- Ανεξαρτησία από τη βάση δεδομένων: Με τη συμβατότητα του Hibernate με διαφορετικές βάσεις δεδομένων και τη δυνατότητα σύνδεσής του με τη βάση μέσω δηλώσεων οριζομένων σε ειδικό αρχείο η αναπτυσσόμενη εφαρμογή μπορεί με ελάχιστες τροποποιήσεις να χρησιμοποιηθεί με βάσεις δεδομένων διαφορετικών κατασκευαστών. Το γεγονός αυτό στερεί μεν από το Hibernate την εκμετάλλευση των ιδιαίτερων χαρακτηριστικών της χρησιμοποιούμενης βάσης, όμως, και σε αυτή την περίπτωση, δίνεται η δυνατότητα χρήσης πηγαίας SQL μέσα στο Hibernate που εκμεταλλεύεται τα ιδιαίτερα αυτά χαρακτηριστικά. Αυτό βέβαια μειώνει την ανεξαρτησία του Hibernate. [7][8]

3.3 HTML

Η HTML (αρχικοποίηση του αγγλικού HyperText Markup Language, ελλ. Γλώσσα Σήμανσης Υπερκειμένου) είναι η κύρια γλώσσα σήμανσης για τις ιστοσελίδες, και τα στοιχεία της είναι τα βασικά δομικά στοιχεία των ιστοσελίδων.

Η HTML γράφεται υπό μορφή στοιχείων HTML τα οποία αποτελούνται από ετικέτες (tags), οι οποίες περικλείονται μέσα σε σύμβολα «μεγαλύτερο από» και «μικρότερο από» (για παράδειγμα <html>), μέσα στο περιεχόμενο της ιστοσελίδας. Οι ετικέτες HTML συνήθως λειτουργούν ανά ζεύγη (για παράδειγμα <h1> και </h1>), με την πρώτη να ονομάζεται ετικέτα έναρξης και τη δεύτερη ετικέτα λήξης (ή σε άλλες περιπτώσεις ετικέτα ανοίγματος και ετικέτα κλεισίματος αντίστοιχα). Ανάμεσα στις ετικέτες, οι σχεδιαστές ιστοσελίδων μπορούν να τοποθετήσουν κείμενο, πίνακες, εικόνες κλπ.

Ο σκοπός ενός web browser είναι να διαβάζει τα έγγραφα HTML και τα συνθέτει σε σελίδες που μπορεί κανείς να διαβάσει ή να ακούσει. Ο browser δεν εμφανίζει τις ετικέτες HTML, αλλά τις χρησιμοποιεί για να ερμηνεύσει το περιεχόμενο της σελίδας.

Τα στοιχεία της HTML χρησιμοποιούνται για να κτίσουν όλους του ιστότοπους. Η HTML επιτρέπει την ενσωμάτωση εικόνων και άλλων αντικειμένων μέσα στη σελίδα, και μπορεί να χρησιμοποιηθεί για να εμφανίσει διαδραστικές φόρμες. Παρέχει τις μεθόδους δημιουργίας δομημένων εγγράφων (δηλαδή εγγράφων που αποτελούνται από το περιεχόμενο που μεταφέρουν και από τον κώδικα μορφοποίησης του περιεχομένου) καθορίζοντας δομικά σημαντικά στοιχεία για το κείμενο, όπως κεφαλίδες, παραγράφους, λίστες, συνδέσμους, παραθέσεις και άλλα. Μπορούν επίσης να ενσωματώνονται σενάρια εντολών σε γλώσσες όπως η JavaScript, τα οποία επηρεάζουν τη συμπεριφορά των ιστοσελίδων HTML.

Οι Web browsers μπορούν επίσης να αναφέρονται σε στυλ μορφοποίησης CSS για να ορίζουν την εμφάνιση και τη διάταξη του κειμένου και του υπόλοιπου υλικού. Ο οργανισμός W3C, ο οποίος δημιουργεί και συντηρεί τα πρότυπα για την HTML και τα CSS, ενθαρρύνει τη χρήση των CSS αντί διαφόρων στοιχείων της HTML για σκοπούς παρουσίασης του περιεχομένου. [9]

Παράδειγμα HTML:

```
<!DOCTYPE html>
<html>
<head>
<title>Page Title</title>
</head>
<body>

<h1>This is a Heading</h1>
<p>This is a paragraph.</p>

</body>
</html>
```

3.4 CSS

Η CSS (Cascading Style Sheets-Διαδοχικά Φύλλα Στυλ) ή (αλληλουχία φύλλων στυλ) είναι μια γλώσσα υπολογιστή που ανήκει στην κατηγορία των γλωσσών φύλλων στυλ που χρησιμοποιείται για τον έλεγχο της εμφάνισης ενός εγγράφου που έχει γραφτεί με μια γλώσσα σήμανσης. Χρησιμοποιείται δηλαδή για τον έλεγχο της εμφάνισης ενός εγγράφου που γράφτηκε στις γλώσσες HTML και XHTML, δηλαδή για τον έλεγχο της εμφάνισης μιας ιστοσελίδας και γενικότερα ενός ιστοτόπου. Η CSS είναι μια γλώσσα υπολογιστή προορισμένη να αναπτύσσει στυλιστικά μια ιστοσελίδα δηλαδή να διαμορφώνει περισσότερα χαρακτηριστικά, χρώματα, στοίχιση και δίνει περισσότερες δυνατότητες σε σχέση με την html. Για μια όμορφη και καλοσχεδιασμένη ιστοσελίδα η χρήση της CSS κρίνεται ως απαραίτητη.[10]

Παράδειγμα CSS:

```
body {
    background-color: lightblue;
}

h1 {
    color: white;
    text-align: center;
}

p {
    font-family: verdana;
    font-size: 20px;
}
```

3.5 JAVASCRIPT

Η JavaScript (JS) είναι διερμηνευμένη γλώσσα προγραμματισμού για ηλεκτρονικούς υπολογιστές. Αρχικά αποτέλεσε μέρος της υλοποίησης των φυλλομετρητών Ιστού, ώστε τα σενάρια από την πλευρά του πελάτη (client-side scripts) να μπορούν να επικοινωνούν με τον χρήστη, να ανταλλάσσουν δεδομένα ασύγχρονα και να αλλάζουν δυναμικά το περιεχόμενο του εγγράφου που εμφανίζεται.

Η JavaScript είναι μια γλώσσα σεναρίων που βασίζεται στα πρωτότυπα (prototype-based), είναι δυναμική, με ασθενείς τύπους και έχει συναρτήσεις ως αντικείμενα πρώτης τάξης. Η σύνταξή της είναι επηρεασμένη από τη C. Η JavaScript αντιγράφει πολλά ονόματα και συμβάσεις ονοματοδοσίας από τη Java, αλλά γενικά οι δύο αυτές γλώσσες δε σχετίζονται και έχουν πολύ διαφορετική σημασιολογία. Οι βασικές αρχές σχεδιασμού της JavaScript προέρχονται από τις γλώσσες προγραμματισμού Self και Scheme. Είναι γλώσσα βασισμένη σε διαφορετικά προγραμματιστικά παραδείγματα (multi-paradigm), υποστηρίζοντας αντικειμενοστρεφές, προστακτικό και συναρτησιακό στυλ προγραμματισμού.

Χρησιμοποιείται και σε εφαρμογές εκτός ιστοσελίδων — τέτοια παραδείγματα είναι τα έγγραφα PDF, οι εξειδικευμένοι φυλλομετρητές (site-specific browsers) και οι μικρές εφαρμογές της επιφάνειας εργασίας (desktop widgets). Οι νεότερες εικονικές μηχανές και πλαίσια ανάπτυξης για JavaScript (όπως το Node.js) έχουν επίσης κάνει τη JavaScript πιο δημοφιλή για την ανάπτυξη εφαρμογών Ιστού στην πλευρά του διακομιστή (server-side).[11]

3.6 Apache Maven

Το Maven είναι λογισμικό διαχείρισης έργου(software project management tool) το οποίο χρησιμοποιείται κυρίως για Java projects. Βασίζεται στην δημιουργία ενός αντικειμένου μοντέλου έργου (project object model - POM) το οποίο μπορεί να διαχειριστεί το χτίσιμο(build) και τις εξαρτήσεις (dependencies) ενός project. Το Maven χρησιμοποιεί ένα αρχείο XML στο οποίο περιγράφεται πώς γίνεται build το project καθώς και ποίες είναι οι εξωτερικές βιβλιοθήκες java (dependencies) που απαιτούνται για να τρέξει το project. Η λήψη των βιβλιοθηκών γίνεται δυναμικά από ένα ή περισσότερα καταθετήρια (repositories), όπως το κεντρικό καταθετήριο του Maven, και στην συνέχεια αποθηκεύονται στην κρυφή μνήμα (cache).[12]

Παράδειγμα αρχείου pom.xml

```
<project>
 <modelVersion>4.0.0</modelVersion>
  <groupId>com.mycompany.app</groupId>
 <artifactId>my-app</artifactId>
 <version>1.0</version>

  <dependencies>
  <dependency>
  <groupId>junit</groupId>
   <artifactId>junit</artifactId>
   <version>3.8.1</version>
  <scope>test</scope>
    </dependency>
</dependencies>

     </project>
```

4.Κύριο μέρος πτυχιακής

4.1 Ανάλυση προβλήματος

Στόχος της εφαρμογής που θα αναπτύξουμε είναι να παρέχει ένα δίαυλο επικοινωνίας μεταξύ των επιχειρήσεων και των ατόμων που ψάχνουν για εργασία. Η εφαρμογή θα δίνει την δυνατότητα εγγραφής χρηστών είτε ως εταιρεία είτε ως υποψήφιος εργαζόμενος.

Οι εταιρίες θα μπορούν να τροποποιούν το προφίλ τους, να δημοσιεύουν αγγελίες εργασίας, να βλέπουν τους υποψηφίους εργαζόμενους που έχουν εγγραφεί στο σύστημα καθώς και τις αιτήσεις που έχουν γίνει για τις αγγελίες που έχουν δημοσιεύσει.

Οι υποψήφιοι εργαζόμενοι θα μπορούν να κάνουν αιτήσεις στις θέσεις εργασίας, χωρίς όμως να έχουν εγγραφεί απαραίτητα στο σύστημα. Με αυτόν τον τρόπο οι εταιρίες θα μπορούν να δημοσιεύουν απλά έναν σύνδεσμο είτε στα μέσα κοινωνικής δικτύωσης είτε σε άλλες ιστοσελίδες και όσοι ενδιαφέρονται για αυτήν την θέση θα μπορούν να κάνουν άμεσα αίτηση.

Κίνητρο για την χρήση αυτής της εφαρμογής από τις εταιρίες είναι ότι παρέχεται η δυνατότητα για αυτόματη αξιολόγηση των αιτήσεων που έχει λάβει μια εταιρεία με βάση τις απαιτήσεις της. Οι αιτήσεις για μια θέση εργασίας δεν θα γίνονται στέλνοντας ένα email με επισυναπτόμενο το βιογραφικό σε word ή pdf, αλλά συμπληρώνοντας μια φόρμα στην οποία θα αναφέρεται η εργασιακή εμπειρία, οι τίτλοι σπουδών και η γνώση ξένων γλωσσών. Η εταιρεία στην συνέχεια μέσω της εφαρμογής θα δίνει τον συντελεστή βαρύτητας για κάθε ένα από τα στοιχεία του βιογραφικού καθώς και τις ελάχιστες ή μέγιστες απαιτήσεις που έχει για την συγκεκριμένη θέση. Η εφαρμογή θα αξιολογεί τις αιτήσεις που έχουν γίνει με βάση τα κριτήρια που έχει δώσει η εταιρεα και θα εμφανίζει σε τι ποσοστό εκπληρώνονται αυτές οι απαιτήσεις από το βιογραφικό του υποψηφίου.

4.2 Απαιτήσεις συστήματος

Οι παρακάτω λειτουργίες πρέπει να παρέχονται από το σύστημα:

- Εγγραφή και σύνδεση χρήστη. Ο χρήστης θα μπορεί να εγγραφεί είτε ως εταιρεία είτε ως υποψήφιος εργαζόμενος. Τα στοιχεία που θα δίνει ο χρήστης κατά την εγγραφή θα είναι Όνομα χρήστη (Username), κωδικός πρόσβασης (Password), επιβεβαίωση κωδικού(Password confirm) και τύπος (Type), εάν είναι δηλαδή εταιρεία ή υποψήφιος εργαζόμενος. Η σύνδεση χρήστη θα γίνεται απλά με το Όνομα χρήστη και τον κωδικό. Επίσης θα πρέπει να γίνεται έλεγχος για τα δεδομένα που δίνει ο χρήστης (validation).
- Επεξεργασία προφίλ χρήστη.
- Δημοσίευση θέσης εργασίας. Εάν ο χρήστης έχει εγγραφεί ως εταιρεία θα μπορεί να δημοσιεύει αγγελίες εργασίας, εφόσον βέβαια έχει συνδεθεί στο σύστημα.
- Περιήγηση στις διαθέσιμες θέσεις εργασίας. Ο κάθε χρήστης ή επισκέπτης της εφαρμογής θα μπορεί να δει τις θέσεις που έχουν δημοσιευθεί.
- Αίτηση σε θέση εργασίας. Οι ενδιαφερόμενοι για μία θέση εργασίας θα μπορούν να υποβάλλουν το βιογραφικό τους, χωρίς να έχουν εγγραφεί απαραίτητα στο σύστημα.

Στην φόρμα υποβολής αίτησης θα αναφέρονται τα στοιχεία του χρήστη, εργασιακή εμπειρία, τίτλοι σπουδών, γνώσεις ξένων γλωσσών.

- Προβολή αιτήσεων που έχουν γίνει για μία θέση εργασίας.
- Αξιολόγηση αιτήσεων. Η εφαρμογή θα αξιολογεί τις αιτήσεις που έχει λάβει η εταιρεία και τις κατατάσσει με βάση τα επιθυμητά προσόντα που απαιτούνται για την εκάστοτε θέση εργασίας. Ο διαχειριστής της εταιρείας θα συμπληρώνει μια φόρμα δίνοντας τα κριτήρια αξιολόγησης, τα οποία είναι εργασιακή εμπειρία, εκπαίδευση και γνώση ξένων γλωσσών. Για κάθε ένα από τα κριτήρια αξιολόγησης θα δίνει την ελάχιστη ή μέγιστη επιθυμητή τιμή καθώς και τον συντελεστή βαρύτητας, ώστε η εφαρμογή στην συνέχεια να αξιολογήσει τα δεδομένα.

4.3 Σχεδιασμός Υλοποίησης

4.3.1 Δημιουργία Βάσης Δεδομένων

Αρχικά θα δημιουργήσουμε μια νέα βάση δεδομένων στην MySQL όπου θα αποθηκευτούν οι πίνακες που θα χρησιμοποιήσουμε στην εφαρμογή μας. Το όνομα της βάσης δεδομένων θα είναι **jobfinder.** Η δημιουργία της βάσης γίνεται με το παρακάτω query:

```
CREATE DATABASE jobfinder DEFAULT CHARACTER SET utf8 DEFAULT COLLATE
utf8_general_ci;
```

Παρακάτω θα δούμε την δημιουργία πινάκων στην βάση δεδομένων χρησιμοποιώντας τα κατάλληλα SQL queries.

4.3.1.1 Users

Σε αυτόν τον πίνακα θα αποθηκεύονται οι χρήστες του συστήματος. Ο κάθε χρήστης έχει κωδικό(id), όνομα χρήστη(username), κωδικό πρόσβασης(password) και ημερομηνία που δημιουργήθηκε (created_at).

```
CREATE TABLE users (
  id         int(10) NOT NULL AUTO_INCREMENT,
  username   varchar(50) NOT NULL,
  password   varchar(255) NOT NULL,
  created_at datetime NOT NULL,
  PRIMARY KEY (id));
```

4.3.1.2 Roles

Οι ρόλοι που μπορεί να έχει ένας χρήστης (π.χ εταιρεία, υποψήφιος, διαχειριστής κτλ). Ο ρόλος έχει κωδικό(id) και όνομα(name).

```
CREATE TABLE roles (
id     int(10) NOT NULL AUTO_INCREMENT,
name   varchar(255) NOT NULL,
PRIMARY KEY (id));
```

4.3.1.3 Users_roles

Σε αυτόν τον πίνακα αποθηκεύονται οι ρόλοι των χρηστών. Ο πίνακας αυτός προκύπτει από την σχέση πολλά προς πολλά μεταξύ χρηστών και ρόλος. Και τα δύο πεδία του πίνακα είναι ξένα κλειδιά που αναφέρονται στους πίνακες users και roles.

```
CREATE TABLE users_roles (
  role_id int(10) NOT NULL,
  user_id int(10) NOT NULL,
  PRIMARY KEY (role_id,
  user_id));

ALTER TABLE users_roles ADD INDEX fk_role_id (role_id), ADD
CONSTRAINT fk_role_id FOREIGN KEY (role_id) REFERENCES roles (id);

ALTER TABLE users_roles ADD INDEX fk_user_id (user_id), ADD
CONSTRAINT fk_user_id FOREIGN KEY (user_id) REFERENCES users (id);
```

4.3.1.4 Companies

Στον πίνακα companies αποθηκεύονται οι εταρίες. Οι εταιρίες μπορούν να έχουν κωδικό, όνομα, ΑΦΜ, χώρα, πόλη, διεύθυνση, email, τηλέφωνο. Το πεδίο user_id αναφέρεται στον κωδικό χρήστη, με τον οποίο έχει γίνει η εγγραφή.

```
CREATE TABLE companies (
  id         int(10) NOT NULL AUTO_INCREMENT,
  name       varchar(100) NOT NULL,
  tax_number int(10),
  country    varchar(100),
  city       varchar(100),
  address    varchar(100),
  email      varchar(100),
  telephone  varchar(100),
  user_id    int(10) NOT NULL,
  PRIMARY KEY (id));

ALTER TABLE companies ADD INDEX fk_company_user_id (user_id), ADD
CONSTRAINT fk_company_user_id FOREIGN KEY (user_id) REFERENCES users
(id);
```

4.3.1.5 Candidates

Στον πίνακα candidates θα αποθηκεύονται οι υποψήφιοι εργαζόμενοι. Ο κάθε υποψήφιος έχει κωδικό(id), όνομα(fname), επώνυμο(lname), ηλικία(age), γένος(gender), διεύθυνση(address), email, τηλέφωνο(telephone), κωδικό χρήστη(user_id) στον οποίο ανήκει. Το πεδίο user_id είναι ξένο κλειδί και αναφέρεται στο πεδίο id του πίνακα users.

```
CREATE TABLE candidates (
  id         int(10) NOT NULL AUTO_INCREMENT,
  fname      varchar(50) NOT NULL,
  lname      varchar(50) NOT NULL,
  age        int(3) NOT NULL,
  gender     varchar(10) NOT NULL,
  address    varchar(100),
  email      varchar(100),
  telephone varchar(100),
  user_id    int(10) NOT NULL,
  PRIMARY KEY (id));

ALTER TABLE candidates ADD INDEX fk_candidate_user_id (user_id), ADD
CONSTRAINT fk_candidate_user_id FOREIGN KEY (user_id) REFERENCES
users (id);
```

4.3.1.6 Job_categories

Κατηγορίες θέσεων εργασίας. Μια κατηγορία έχει κωδικό(id) και όνομα(name).

```
CREATE TABLE job_categories (
id         int(10) NOT NULL AUTO_INCREMENT,
name       varchar(100) NOT NULL,
PRIMARY KEY (id));
```

4.3.1.7 Jobs

Σε αυτόν τον πίνακα θα αποθηκεύονται οι θέσεις εργασίας. Μια θέση εργασίας έχει κωδικό(id), τίτλο(title), περιγραφή(description), τύπο απασχόλησης(employment_type), μισθό(salary), τοποθεσία(location), τηλέφωνο(telephone), email, ημ. δημοσίευσης(date_posted), κωδικό κατηγορίας(job_category_id), κωδικό εταιρείας(company_id) στην οποία ανήκει.

```
CREATE TABLE jobs (
id                  int(10) NOT NULL AUTO_INCREMENT,
title               varchar(100) NOT NULL,
description         varchar(255) NOT NULL,
employment_type     varchar(50) NOT NULL,
required_experience varchar(100),
salary              float,
location            varchar(255) NOT NULL,
telephone           varchar(20) NOT NULL,
email               varchar(50) NOT NULL,
date_posted         datetime DEFAULT NOW() NOT NULL,
job_category_id     int(10),
company_id          int(10) NOT NULL,
PRIMARY KEY (id));

ALTER TABLE jobs ADD INDEX fk_company_id (company_id), ADD CONSTRAINT
fk_company_id FOREIGN KEY (company_id) REFERENCES companies (id) ON
UPDATE Cascade ON DELETE Cascade;

ALTER TABLE jobs ADD INDEX fk_job_category_id (job_category_id), ADD
CONSTRAINT fk_job_category_id FOREIGN KEY (job_category_id) REFERENCES
job_categories (id);
```

4.3.1.8 Job_applications

Στον πίνακα **job_applications** αποθηκεύονται οι αιτήσεις για τις θέσεις εργασίας. Μία αίτηση περιέχει κωδικό(id), όνομα(fname), επώνυμο(lname), email, διεύθυνση(address) , τηλέφωνο(phone), ημ. δημιουργίας(created_at) , συνοδευτική επιστολή(cover_letter), κωδικός θέσης(job_id) στην οποία ανήκει η αίτηση, κωδικός υποψήφιου(candidate_id) που υπέβαλλε την αίτηση.

```
CREATE TABLE job_applications (
id           int(10) NOT NULL AUTO_INCREMENT,
fname        varchar(100) NOT NULL,
lname        varchar(100) NOT NULL,
email        varchar(100) NOT NULL,
address      varchar(255) NOT NULL,
phone        varchar(50) NOT NULL,
qualified    varchar(50),
created_at   datetime DEFAULT NOW() NOT NULL,
cover_letter varchar(512),
job_id       int(10) NOT NULL,
candidate_id int(10),
PRIMARY KEY (id));

ALTER TABLE job_applications ADD INDEX fk_job_id (job_id), ADD CONSTRAINT
fk_job_id FOREIGN KEY (job_id) REFERENCES jobs (id);

ALTER TABLE job_applications ADD INDEX fk_candidate_id (candidate_id), ADD
CONSTRAINT fk_candidate_id FOREIGN KEY (candidate_id) REFERENCES candidates
(id);
```

4.3.1.9 Job_applications_working_experiences

Σε μια αίτηση εργασίας μπορεί να περιλαμβάνονται πολλές εργασιακές εμπειρίες. Μια εργασιακή εμπειρία αποθηκεύεται στον πίνακα **job_applications_working_experiences** και περιλαμβάνει κωδικό(id), τίτλο(title), εταιρεία(company), περιγραφή(description), βιομηχανία(industry), ημ. εναρξης(start_date), ημ. λήξης(end_date), κωδικός αίτησης(job_application_id) στην οποία ανήκει. Η σχέση μεταξύ των οντοτήτων Εργασιακή εμπειρία και Θέση Εργασίας είναι πολλά-προς-ένα αντίστοιχα, άρα το πεδίο job_application_id είναι ξένο κλειδί και αναφέρεται στο πεδίο id του πίνακα job_applications.

```
CREATE TABLE job_applications_working_experiences (
id                  int(10) NOT NULL AUTO_INCREMENT,
title               varchar(100) NOT NULL,
company             varchar(100) NOT NULL,
description         varchar(255) NOT NULL,
industry            varchar(255) NOT NULL,
start_date          date NOT NULL,
end_date            date NOT NULL,
job_application_id int(10) NOT NULL,
PRIMARY KEY (int));

ALTER TABLE job_applications_working_experiences ADD INDEX
fk_working_experience_job_application_id (job_application_id), ADD
CONSTRAINT fk_working_experience_job_application_id FOREIGN KEY
(job_application_id) REFERENCES job_applications (id) ON UPDATE Cascade
ON DELETE Cascade;
```

4.3.1.10 Education_levels

Σε αυτόν τον πίνακα αποθηκεύονται τα επίπεδα εκπαίδευσης (π.χ μεταπτυχιακό ή διδακτορικό). Το κάθε επίπεδο έχει κωδικό(id), τίτλο(title), κατάταξη(ranking).

```
CREATE TABLE education_levels (
id      int(10) NOT NULL AUTO_INCREMENT,
title   varchar(20) NOT NULL,
ranking int(3) NOT NULL,
PRIMARY KEY (id));
```

4.3.1.11 Job_applications_education

Σε μία αίτηση εργασίας ο υποψήφιος μπορεί να καταχωρήσει πολλούς τίτλους σπουδών. Η εκπαίδευση, δηλαδή οι τίτλοι σπουδών θα αποθηκεύονται στον πίνακα **job_applications_education.** Τα πεδία του πίνακα είναι κωδικός(id), πτυχίο(degree), σχολή(school), πεδίο σπουδών(field_of_study), περιγραφή(description), βαθμός(grade), ημ. έναρξης(start_date), ημ. αποφοίτησης(end_date), κωδικός αίτησης(job_application_id) στην οποία ανήκει και κωδικός βαθμίδας εκπαίδευσης(education_level_id). Η σχέση αυτού του πίνακα με τον πίνακα job_applications είναι πολλά-προς-ένα, γι αυτό το πεδίο ***job_application_id*** είναι ξένο κλειδί το οποίο αναφέρεται στο πεδίο ***id*** του πίνακα **job_applications**. Επίσης ξένο κλειδί είναι και το πεδίο ***education_level_id*** το οποίο αναφέρεται στο πεδίο ***id*** του πίνακα **education_levels.**

```
CREATE TABLE job_applications_education (
id                  int(10) NOT NULL AUTO_INCREMENT,
degree              varchar(50) NOT NULL,
school              varchar(50) NOT NULL,
field_of_study      varchar(100),
description         varchar(255),
grade               float,
start_date          date,
end_date            date,
job_application_id int(10) NOT NULL,
education_level_id int(10) NOT NULL,
PRIMARY KEY (id));

ALTER TABLE job_applications_education ADD INDEX
fk_education_job_application_id (job_application_id), ADD CONSTRAINT
fk_education_job_application_id FOREIGN KEY (job_application_id) REFERENCES
job_applications (id) ON UPDATE Cascade ON DELETE Cascade;

ALTER TABLE job_applications_education ADD INDEX fk_education_level
(education_level_id), ADD CONSTRAINT fk_education_level FOREIGN KEY
(education_level_id) REFERENCES education_levels (id);
```

4.3.1.12 Languages

Στον πίνακα **languages** αποθηκεύονται οι ξένες γλώσσες. Η κάθε γλώσσα έχει κωδικό(id) και όνομα(name).

```
CREATE TABLE languages (
  id   int(10) NOT NULL AUTO_INCREMENT,
  name varchar(50) NOT NULL,
  PRIMARY KEY (id));
```

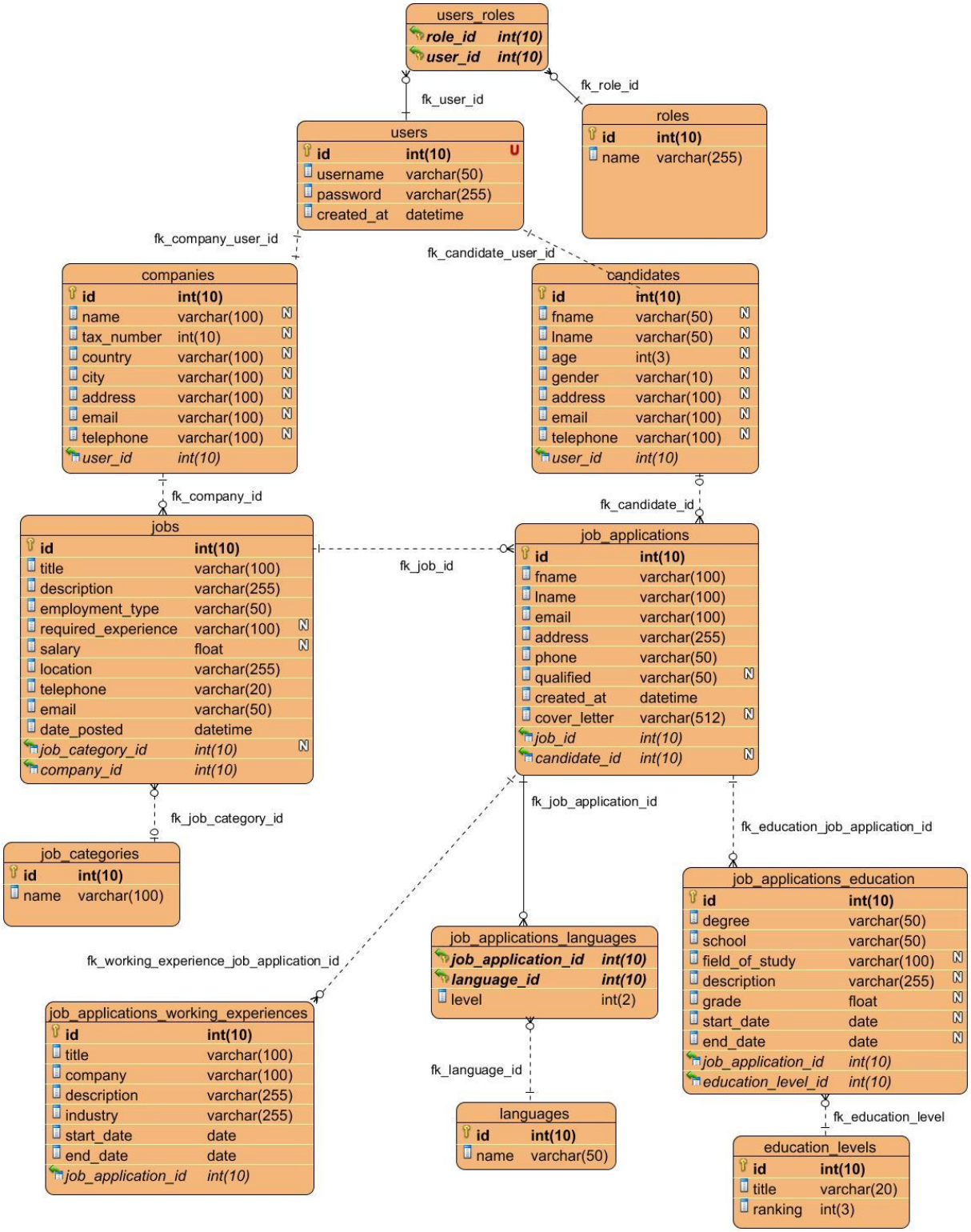
4.3.1.13 Job_application_languages

Σε μία αίτηση εργασίας, ένας υποψήφιος μπορεί να γνωρίζει πολλές γλώσσες. Η σχέση μεταξύ των οντοτήτων αίτηση εργασίας(job_application) και γλώσσας(language) είναι πολλά προς πολλά, άρα θα δημιουργηθεί ο πίνακας **job_applications_languages** στον οποίο θα περιλαμβάνονται τα πρωτεύοντα κλειδιά των δύο οντοτήτων καθώς και το επίπεδο γνώσης(level) μιας γλώσσας.

```
CREATE TABLE job_applications_languages (
  job_application_id int(10) NOT NULL,
  language_id        int(10) NOT NULL,
  level              int(2) NOT NULL,
  PRIMARY KEY (job_application_id,
  language_id));

ALTER TABLE job_applications_languages ADD INDEX fk_language_id
(language_id), ADD CONSTRAINT fk_language_id FOREIGN KEY (language_id)
REFERENCES languages (id) ON UPDATE Cascade ON DELETE Cascade;

ALTER TABLE job_applications_languages ADD INDEX fk_job_application_id
(job_application_id), ADD CONSTRAINT fk_job_application_id FOREIGN KEY
(job_application_id) REFERENCES job_applications (id) ON UPDATE Cascade ON
DELETE Cascade;
```

4.3.1.14 Εισαγωγή δεδομένων

```
INSERT INTO roles VALUES (1,'ROLE_COMPANY');
INSERT INTO roles VALUES (2,'ROLE_CANDIDATE');

INSERT INTO job_categories VALUES (1,'All Categories');
INSERT INTO job_categories VALUES (2,'IT - Telecommunications');
INSERT INTO job_categories VALUES (3,'Health Sector');
INSERT INTO job_categories VALUES (4,'Manufacturing');
INSERT INTO job_categories VALUES (5,'Business/ Management');
INSERT INTO job_categories VALUES (6,'Sales');
INSERT INTO job_categories VALUES (7,'Marketing');
INSERT INTO job_categories VALUES (8,'Hospitality');
INSERT INTO job_categories VALUES (9,'Education');
INSERT INTO job_categories VALUES (10,'Public Sector');
```



Εικόνα 5 – Η βάση δεδομένων

4.4 Υλοποίηση εφαρμογής

Ο πηγαίος κώδικας που θα δούμε στα επόμενα κεφάλαια είναι διαθέσιμος για λήψη στο github.com/dimimav/jobfinder-thesis.

Αρχικά κατεβάζουμε το eclipse IDE για Java EE από το link http://www.eclipse.org/downloads/packages/eclipse-ide-java-ee-developers/keplersr2

Μετά την εγκατάσταση του eclipse, δημιουργούμε ένα νέο maven project όπως είδαμε στο κεφάλαιο 3.1.

4.4.1 Xml configuration files

4.4.1.1 pom.xml

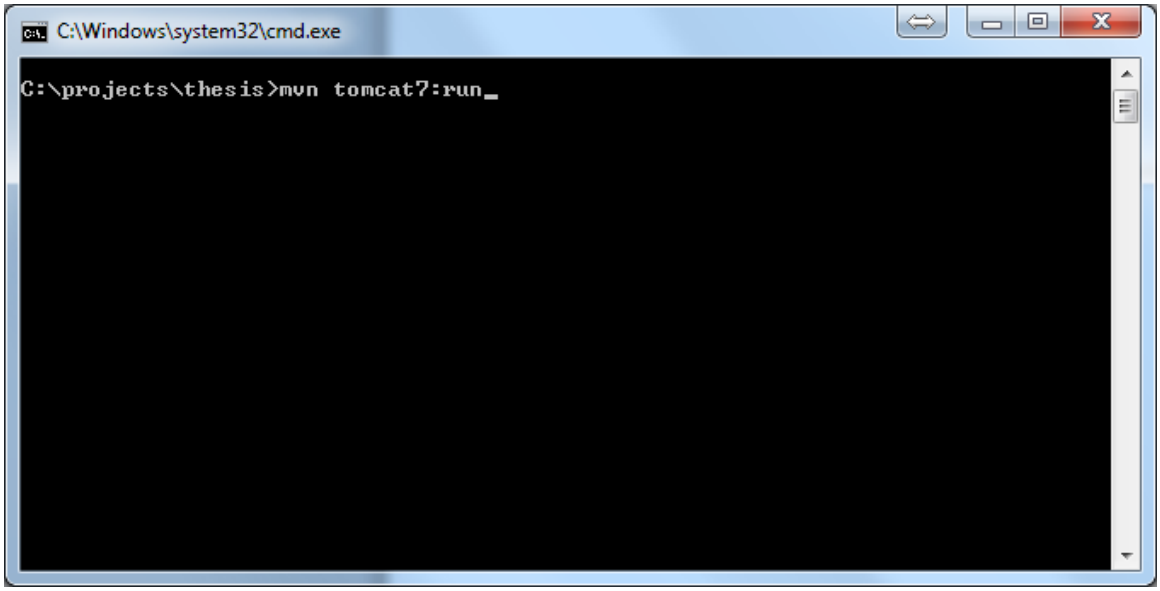
Στο αρχείο **pom.xml**, θα πρέπει να ορίσουμε το πώς γίνεται build το project καθώς και τα dependencies (εξωτερικές βιβλιοθήκες) που θα χρειαστούμε.

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
	xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/maven-v4_0_0.xsd">
	<modelVersion>4.0.0</modelVersion>
	<groupId>thesis</groupId>
	<artifactId>jobfinder</artifactId>
	<packaging>war</packaging>
	<version>0.0.1-SNAPSHOT</version>
	<name>jobFinder Maven Webapp</name>
	<url>http://maven.apache.org</url>
	<properties>
		<spring.version>4.2.0.RELEASE</spring.version>
		<spring-security.version>4.0.2.RELEASE</spring-security.version>
		<spring-data-jpa.version>1.8.2.RELEASE</spring-data-jpa.version>
		<hibernate.version>4.3.11.Final</hibernate.version>
		<hibernate-validator.version>5.2.1.Final</hibernate-validator.version>
		<mysql-connector.version>5.1.36</mysql-connector.version>
		<commons-dbcp.version>1.4</commons-dbcp.version>
		<jstl.version>1.2</jstl.version>
		<junit.version>3.8.1</junit.version>
		<logback.version>1.1.3</logback.version>
		<maven.compiler.source>1.8</maven.compiler.source>
		<maven.compiler.target>1.8</maven.compiler.target>
	</properties>
	<dependencies>
		<dependency>
			<groupId>org.springframework</groupId>
			<artifactId>spring-web</artifactId>
			<version>${spring.version}</version>
		</dependency>
		<dependency>
			<groupId>org.springframework</groupId>
			<artifactId>spring-webmvc</artifactId>
			<version>${spring.version}</version>
		</dependency>
```

```
<dependency>
	<groupId>org.springframework</groupId>
	<artifactId>spring-core</artifactId>
	<version>${spring.version}</version>
</dependency>

<dependency>
	<groupId>org.springframework.security</groupId>
	<artifactId>spring-security-web</artifactId>
	<version>${spring-security.version}</version>
</dependency>
<dependency>
	<groupId>org.springframework.security</groupId>
	<artifactId>spring-security-config</artifactId>
	<version>${spring-security.version}</version>
</dependency>
<dependency>
	<groupId>org.springframework.security</groupId>
	<artifactId>spring-security-taglibs</artifactId>
	<version>${spring-security.version}</version>
</dependency>
<dependency>
	<groupId>org.hibernate</groupId>
	<artifactId>hibernate-validator</artifactId>
	<version>${hibernate-validator.version}</version>
</dependency>
<dependency>
	<groupId>org.springframework.data</groupId>
	<artifactId>spring-data-jpa</artifactId>
	<version>${spring-data-jpa.version}</version>
</dependency>
<dependency>
	<groupId>org.hibernate</groupId>
	<artifactId>hibernate-entitymanager</artifactId>
	<version>${hibernate.version}</version>
</dependency>
<dependency>
	<groupId>mysql</groupId>
	<artifactId>mysql-connector-java</artifactId>
	<version>${mysql-connector.version}</version>
</dependency>
<dependency>
	<groupId>commons-dbcp</groupId>
	<artifactId>commons-dbcp</artifactId>
	<version>${commons-dbcp.version}</version>
</dependency>
<dependency>
	<groupId>javax.servlet</groupId>
	<artifactId>jstl</artifactId>
	<version>${jstl.version}</version>
</dependency>
<dependency>
	<groupId>junit</groupId>
	<artifactId>junit</artifactId>
	<version>${junit.version}</version>
	<scope>test</scope>
</dependency>
<dependency>
	<groupId>ch.qos.logback</groupId>
	<artifactId>logback-classic</artifactId>
	<version>${logback.version}</version>
```

```
        <dependency>
              <groupId>log4j</groupId>
              <artifactId>log4j</artifactId>
              <version>1.2.17</version>
        </dependency>
    </dependencies>
      <build>
            <finalName>/</finalName>
            <plugins>
                  <plugin>
                        <artifactId>maven-compiler-plugin</artifactId>
                        <version>2.3.2</version>
                        <configuration>
                              <source>1.7</source>
                              <target>1.7</target>
                        </configuration>
                  </plugin>
                  <plugin>
                        <groupId>org.apache.tomcat.maven</groupId>
                        <artifactId>tomcat7-maven-plugin</artifactId>
                        <version>2.2</version>
                  </plugin>
            </plugins>
      </build>
</project>
```

Για να τρέξουμε την εφαρμογή, αρκεί να εκτελούμε την εντολή **mvn tomcat7:run** μέσω του command line:



4.4.1.2 web.xml

Δημιουργία αρχείου **web.xml** (src/main/webapp/WEB-INF). Το στοιχείο (element) display-name καθορίζει το όνομα με το οποίο θα εμφανίζεται η εφαρμογή. Χρησιμοποιώντας το servlet element ορίζουμε ένα servlet με όνομα dispatcher(servlet-name) και κλάση(servlet-class) την DispatcherServlet του spring (org.springframework.web.servlet.DispatcherServet). Μέσω του στοιχείου context-param ορίζουμε τους παραμέτρους ενός servlet. Η παράμετρος contextConfigLocation ορίζει την τοποθεσία του αρχείου με τις ρυθμίσεις που χρειάζεται το spring. Στην παρακάτω περίπτωση, ορίζουμε τις ρυθμίσεις στο αρχείο appconfig-root.xml.

Το filter element καθορίζει μια κλάση τύπου φίλτρου μαζί με τα χαρακτηριστικά του (attributes). Χρησιμοποιούμε την κλάση του spring DelegatingFilterProxy για να φιλτράρουμε όλα τα requests της εφαρμογής για λόγους ασφαλείας(url-pattern: /*).

```
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xmlns="http://java.sun.com/xml/ns/javaee"
         xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
       http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
         version="2.5">

    <display-name>Job Finder</display-name>

    <context-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>/WEB-INF/appconfig-root.xml</param-value>
    </context-param>

    <filter>
        <filter-name>springSecurityFilterChain</filter-name>
        <filter-
class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>
    </filter>
    <filter-mapping>
        <filter-name>springSecurityFilterChain</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>

    <servlet>
        <servlet-name>dispatcher</servlet-name>
        <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-
class>
        <init-param>
            <param-name>contextConfigLocation</param-name>
            <param-value></param-value>
        </init-param>
        <load-on-startup>1</load-on-startup>
    </servlet>

    <servlet-mapping>
        <servlet-name>dispatcher</servlet-name>
        <url-pattern>/</url-pattern>
    </servlet-mapping>

    <listener>
        <listener-
class>org.springframework.web.context.ContextLoaderListener</listener-class>
    </listener>
</web-app>
```

4.4.1.3 appconfig-root.xml

Σε αυτό το αρχείο περιλαμβάνονται οι παράμετροι προετοιμασίας(initialization parameters) για το DispatcherServlet του spring. Θα δημιουργήσουμε τρία διαφορετικά αρχεία xml, για τις ρυθμίσεις του mvc, δεδομένων(data) και ασφάλειας(security), τα οποία είναι εισάγονται σε αυτό το κεντρικό αρχείο. Το χαρακτηριστικό(attribute) base-package του element <context:component-scan> ορίζει το πακέτο στο οποίο περιέχονται annotated beans (π.χ @Controller, @Service, @Repository).

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:context="http://www.springframework.org/schema/context"
       xmlns="http://www.springframework.org/schema/beans"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
            http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context.xsd">

    <import resource="appconfig-mvc.xml"/>

    <import resource="appconfig-data.xml"/>

    <import resource="appconfig-security.xml"/>

    <context:component-scan base-package="com.jobfinder.*"/>

</beans>
```

4.4.1.4 Appconfig-mvc.xml

Σε αυτό το αρχείο περιλαμβάνονται ρυθμίσεις σχετικές με το mvc.

Μέσω του στοιχείου <mvc:annotation> παρέχεται υποστήριξη για την χρήση annotated controllers (π.χ @Controller, @RequestMapping).

```
<beans xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:mvc="http://www.springframework.org/schema/mvc"
       xmlns="http://www.springframework.org/schema/beans"
       xsi:schemaLocation="http://www.springframework.org/schema/mvc
http://www.springframework.org/schema/mvc/spring-mvc.xsd
http://www.springframework.org/schema/beans
 http://www.springframework.org/schema/beans/spring-beans.xsd">

    <mvc:annotation-driven/>

    <mvc:resources mapping="/resources/**" location="/resources/"/>

    <bean id="messageSource"
class="org.springframework.context.support.ReloadableResourceBundleMessageSource">
        <property name="basenames">
            <list>
                <value>classpath:validation</value>
            </list>
        </property>
    </bean>

    <bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
        <property name="prefix">
            <value>/WEB-INF/views/</value>
        </property>
        <property name="suffix">
            <value>.jsp</value>
        </property>
    </bean>
</beans>
```

4.4.1.5 appconfig-data.xml

Σε αυτό το xml αρχείο θα ορίσουμε τις ρυθμίσεις που σχετίζονται με τα δεδομένα της εφαρμογής. Θα ορίσουμε ένα Java Bean με id ***transactionManager*** το οποίο χρησιμοποιεί την κλάση ***HibernateTransactionManager*** η οποία περιλαμβάνεται στο package Hibernate4. Μέσω αυτής της κλάσης θα μπορούμε να χρησιμοποιήσουμε το Hibernate ώστε να επικοινωνούμε με την βάση δεδομένων. Το Bean ***sessionFactory*** χρησιμοποιείται για την παραγωγή αντικειμένων τύπου **session** (συνεδρία), μέσω των οποίων γίνεται η ανάκτηση των δεδομένων από την ΒΔ. Μέσω του ***dataSource*** bean ορίζονται τα διαπιστευτήρια (credentials), όπως url, username,password για πρόσβαση στην πηγή δεδομένων, δηλαδή την ΒΔ.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:p="http://www.springframework.org/schema/p"
       xmlns:tx="http://www.springframework.org/schema/tx"
xmlns:context="http://www.springframework.org/schema/context"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
  http://www.springframework.org/schema/beans/spring-beans.xsd
  http://www.springframework.org/schema/context
  http://www.springframework.org/schema/context/spring-context.xsd
  http://www.springframework.org/schema/tx
  http://www.springframework.org/schema/tx/spring-tx.xsd">

    <tx:annotation-driven/>

       <bean id="propertyConfigurer"
class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">
             <property name="locations">
                    <list>
                           <value>classpath:database.properties</value>
                    </list>
             </property>
       </bean>

          <!-- Data source bean configuration -->
    <bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource" destroy-
method="close">
        <property name="driverClassName" value="${jdbc.driverClassName}"/>
        <property name="url" value="${jdbc.url}"/>
        <property name="username" value="${jdbc.username}"/>
        <property name="password" value="${jdbc.password}"/>
    </bean>

       <bean id="sessionFactory"
             class="org.springframework.orm.hibernate4.LocalSessionFactoryBean">
             <property name="dataSource" ref="dataSource" />
             <property name="packagesToScan">
                    <list>
                           <value>com.jobfinder.model</value>
                    </list>
             </property>
             <property name="hibernateProperties">
                    <props>
                           <prop key="hibernate.hbm2ddl.auto">update</prop>
                           <prop
key="hibernate.dialect">${hibernate.dialect}</prop>
                           <prop key="hibernate.show_sql">false</prop>
                    </props>
             </property>
       </bean>
       <bean id="transactionManager"
       class="org.springframework.orm.hibernate4.HibernateTransactionManager">
             <property name="sessionFactory" ref="sessionFactory" />
       </bean>
</beans>
```

4.4.1.6 Appconfig-security.xml

Χρησιμοποιώντας το element **intercept-url,** πριν ο χρήστης μεταβεί στο URL /user/dashboard γίνεται έλεγχος εάν έχει δικαίωμα πρόσβασης. Στην προκειμένη περίπτωση θα πρέπει να έχει ρόλο με τίτλο "ROLE_COMPANY" ή "ROLE_CANDIDATE". Οι λεγόμενοι **interceptors** χρησιμοποιούνται για να διακόψουν την εκτέλεση ενός http request προκειμένου να γίνουν οι απαιτούμενοι έλεγχοι ασφαλείας. Στην παρακάτω περίπτωση, εάν ο χρήστης προσπαθήσει να μεταβεί στον σύνδεσμο /user/dashboard, τότε ο interceptor που έχουμε ορίσει να ελέγξει εάν έχει την κατάλληλη πρόσβαση. Σε αντίθετη περίπτωση, θα μεταβεί στην σελίδα σύνδεσης, η οποία ορίζεται στο element <form-login login-page='login'>.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans:beans xmlns="http://www.springframework.org/schema/security"
             xmlns:beans="http://www.springframework.org/schema/beans"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             xsi:schemaLocation="http://www.springframework.org/schema/beans
             http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
             http://www.springframework.org/schema/security
             http://www.springframework.org/schema/security/spring-
security.xsd">

    <http auto-config="true">
        <intercept-url pattern="/user/dashboard"
access="hasAnyRole('ROLE_COMPANY','ROLE_CANDIDATE')"/>
        <form-login login-page="/login" default-target-url="/user/dashboard"
authentication-failure-url="/login?error" username-parameter="username"
password-parameter="password"/>
        <logout logout-success-url="/login?logout" />
             <csrf disabled="true"/>
    </http>

    <authentication-manager alias="authenticationManager">
        <authentication-provider user-service-ref="userDetailsServiceImpl">
            <password-encoder ref="encoder"></password-encoder>
        </authentication-provider>
    </authentication-manager>

    <beans:bean id="userDetailsServiceImpl"
class="com.jobfinder.service.Impl.UserDetailsServiceImpl"></beans:bean>

    <beans:bean id="encoder"

class="org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder">
        <beans:constructor-arg name="strength" value="11"/>
    </beans:bean>
</beans:beans>
```

Property files

Αρχεία που περιέχουν πληροφορίες για την σύνδεση με την βάση δεδομένων, ρυθμίσεις logger και validation errors. Αποθηκεύονται στον φάκελο src/main/resources

1. database.properties

Σε αυτό το αρχείο ορίζουμε στοιχεία για την πρόσβαση στην MySQL βάση δεδομένων. Στην παρακάτω περίπτωση έχουμε ορίσει ως host το localhost (εφόσον η εφαρμογή τρέχει τοπικά), database name: jobfinder, username: root, password: zoot .

```
jdbc.driverClassName=com.mysql.jdbc.Driver
jdbc.url=jdbc:mysql://localhost/jobfinder?useUnicode=true&characterE
ncoding=utf-8&autoReconnect=true
jdbc.username=root
jdbc.password=zoot

hibernate.dialect=org.hibernate.dialect.MySQL5InnoDBDialect
```

2. log4j.properties

Το log4j είναι ένα εργαλείο που χρησιμοποιείται για την καταγραφή μηνυμάτων σε αρχεία (log files). Σε αυτό το αρχείο ορίζουμε τις ρυθμίσεις του, όπως το επίπεδο καταγραφής(log level), την τοποθεσία που θα αποθηκεύονται τα log files, το format κτλ.

```
# Root logger option
log4j.rootLogger=DEBUG, file,errorlog

# Redirect log messages to a log file, support file rolling.
log4j.appender.file=org.apache.log4j.RollingFileAppender
log4j.appender.file.File=log/debug.log
log4j.appender.file.MaxFileSize=5MB
log4j.appender.file.MaxBackupIndex=10
log4j.appender.file.layout=org.apache.log4j.PatternLayout
log4j.appender.file.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss} %-5p
%c{1}:%L - %m%n

log4j.appender.errorlog=org.apache.log4j.RollingFileAppender
log4j.appender.errorlog.Threshold=error
log4j.appender.errorlog.File=log/error.log
log4j.appender.errorlog.MaxFileSize=5MB
log4j.appender.errorlog.MaxBackupIndex=10
log4j.appender.errorlog.layout=org.apache.log4j.PatternLayout
log4j.appender.errorlog.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss} %-5p
%c{1}:%L - %m%n
```

3. validation.properties

Αρχείο στο οποίο αποθηκεύουμε μηνύματα που έχουν να κάνουν με το validation.

```
NotEmpty=This field is required.
Size.userForm.username=Please use between 6 and 32
characters.
Duplicate.userForm.username=Someone already has that
username.
Size.userForm.password=Try one with at least 8 characters.
Diff.userForm.passwordConfirm=These passwords don't match.
```

4.4.1.7 Repository Layer

Δημιουργούμε το package **com.jobfinder.dao,** στο οποίο θα δημιουργήσουμε και θα υλοποιήσουμε διεπαφές (interfaces) που επικοινωνούν με την βάση δεδομένων. Σκοπός του DAO(Date access object) Layer ή αλλιώς Repository Layer, είναι η πρόσβαση σε μια συγκεκριμένη πηγή δεδομένων (π.χ UserDAO).

Η κλάση **GenericDAO** περιγράφει τις CRUD λειτουργίες (Create – Read – Update - Delete) που μπορούν να εφαρμοσθούν από οποιαδήποτε οντότητα. Η κλάση DAO της κάθε οντότητας που πρόκειται να δημιουργήσουμε(π.χ JobDAO), θα κληρονομεί τις μεθόδους της GenericDAO. Με αυτόν τον τρόπο εξοικονομούμε αρκετό κώδικα, αφού στο μέλλον δεν θα χρειάζεται να ξαναγράψουμε μεθόδους για βασικές λειτουργίες πρόσβασης στα δεδομένα της βάσης δεδομένων.

Το **SessionFactory** είναι ένα εργοστάσιο παραγωγής αντικειμένων τύπου **Session**. Το αντικείμενο Session χρησιμοποιείται από το Hibernate για την φυσική σύνδεση με μία βάση δεδομένων με σκοπό την ανάκτηση δεδομένων.

```
package com.jobfinder.dao;

public interface GenericDAO<E,K> {

	public void add(E entity);

	public void saveOrUpdate(E entity) ;

	public void update(E entity) ;

	public void delete(E entity);

	public E find(K key);

	public List<E> getAll();
}
```

```java
@Repository
public abstract class GenericDAOImpl<E,K extends Serializable> implements
GenericDAO<E, K> {

	@Autowired
	private SessionFactory sessionFactory;

    protected Class<? extends E> daoType;


	protected SessionFactory getSessionFactory() {
		return sessionFactory;
	}

	protected Session getSession() {
		Session sess = getSessionFactory().getCurrentSession();
		if (sess == null) {
			sess = getSessionFactory().openSession();
		}
		return sess;
	}

    protected Session currentSession() {
        return sessionFactory.getCurrentSession();
    }
    public GenericDAOImpl() {
        Type t = getClass().getGenericSuperclass();
        ParameterizedType pt = (ParameterizedType) t;
        daoType = (Class) pt.getActualTypeArguments()[0];
    }

    @Override
    public void add(E entity) {
	   getSession().save(entity);
    }

    @Override
    public void saveOrUpdate(E entity) {
	   getSession().saveOrUpdate(entity);
    }

    @Override
    public void update(E entity) {
	   getSession().merge(entity);
    }

    @Override
    public E find(K key) {
        return (E) getSession().get(daoType, (Serializable) key);
    }

    @Override
    public List<E> getAll(){
        return getSession().createCriteria(daoType).list();
    }

	public void delete(E entity){
		getSession().delete(entity);
	}

}
```

4.4.1.8 Service Layer

Υλοποιώντας το επίπεδο υπηρεσίας (Service Layer) μπορούμε να έχουμε έναν σαφή διαχωρισμό μεταξύ των λειτουργιών του Controller και της επιχειρησιακής λογικής(Business logic). Το service layer είναι ο διαμεσολαβητής μεταξύ του controller και του repository(DAO) layer. Δημιουργούμε το package service, μέσα στο οποίο τοποθετούνται οι κλάσεις στις οποίες υλοποιείται το Business Logic.

```
package com.jobfinder.service;

public interface GenericService<E, K> {

        public void add(E entity);

        public void saveOrUpdate(E entity) ;

        public void update(E entity) ;

        public void delete(E entity);

        public E find(K key);

        public List<E> getAll();

}
```

```java
@Service
public abstract class GenericServiceImpl<E, K> implements GenericService<E, K> {

	private GenericDAO<E, K> genericDAO;

	public GenericServiceImpl(GenericDAO<E, K> genericDAO) {
		this.genericDAO = genericDAO;
	}

	public GenericServiceImpl() {
	}

	@Override
	@Transactional
	public void add(E entity) {
		genericDAO.add(entity);
	}
	@Override
	@Transactional
	public void saveOrUpdate(E entity) {
		genericDAO.saveOrUpdate(entity);
	}
	@Override
	@Transactional
	public void update(E entity) {
		genericDAO.update(entity);
	}
	@Override
	@Transactional
	public void delete(E entity) {
		genericDAO.delete(entity);
	}
	@Override
	@Transactional(readOnly = true)
	public E find(K key) {
		return genericDAO.find(key);
	}
	@Override
	@Transactional(readOnly = true)
	public List<E> getAll() {
		return genericDAO.getAll();
	}
}
```

i. Common Views

Οι όψεις (Views) της εφαρμογής αποθηκεύονται στον φάκελο **src/main/webapp/WEB-INF/views.** Όλες οι σελίδες της εφαρμογής έχουν κάποια κοινά στοιχεία όπως την κεφαλίδα (head), μπάρα περιήγησης(navbar) και υποσέλιδο (footer). Για αυτόν τον λόγο θα δημιουργήσουμε 3 διαφορετικά αρχεία τα οποία θα περιλαμβάνονται(include) από τα άλλα views. Ο φάκελος στον οποίο θα αποθηκεύονται αυτά τα κοινά views είναι ο **views/include**.

1. head.jsp

Στο αρχείο head.jsp περιλαμβάνονται οι δηλώσεις για την εισαγωγή εξωτερικών βιβλιοθηκών.

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<%@ taglib prefix="spring" uri="http://www.springframework.org/tags"%>
<%@ taglib prefix="sec" uri="http://www.springframework.org/security/tags" %>

<c:set var="contextPath" value="${pageContext.request.contextPath}" />

<sec:authorize var="loggedIn" access="isAuthenticated()" />

<meta http-equiv="Content-Type" content="text/html; utf-8">
<meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-
fit=no">

<link rel="icon" href="../../favicon.ico">

<!-- JQuery -->
<script src="${contextPath}/resources/lib/jquery/jquery-1.10.2.js"></script>
<script src="${contextPath}/resources/lib/jquery/jquery-ui-1.12.1.js"></script>

<!-- Custom js -->
<script src="${contextPath}/resources/js/custom.js"></script>

<!-- Bootstrap -->
<link href="${contextPath}/resources/lib/bootstrap/bootstrap.min.css"
rel="stylesheet">
<script src="${contextPath}/resources/lib/bootstrap/bootstrap.min.js"></script>

<!-- Custom styles  -->
<link href="${contextPath}/resources/css/justified-nav.css" rel="stylesheet">
<link href="${contextPath}/resources/css/custom.css" rel="stylesheet">

<link rel="stylesheet" href="${contextPath}/resources/css/jquery-ui.css">
```

2. navbar.jsp

```
<div class="masthead">
       <h3 class="text-muted">JobFinder</h3>
       <nav>
              <ul class="nav nav-justified">
                     <li class="nav-item"><a id="home-link" class="nav-link"
       href="${contextPath}">Home</a></li>
                     <li class="nav-item"><a id="listJobs-link" class="nav-link"
       href="${contextPath}/listJobs">Jobs</a></li>
                     <li class="nav-item"><a class="nav-link"
href="${contextPath}/candidates" id="candidates-link">Candidates</a></li>
                     <li class="nav-item"><a class="nav-link"
href="${contextPath}/companies" id="companies-link">Companies</a></li>
                     <c:choose>
                            <c:when test="${loggedIn}">
                                   <li class="nav-item"><a class="nav-link"
href="${contextPath}/user/dashboard" id="dashboard-link">Dashboard</a></li>
                                   <li class="nav-item"><a class="nav-link"
onclick="document.forms['logoutForm'].submit()" href="#">Logout</a></li>
                            </c:when>
                            <c:otherwise>
                            <li class="nav-item"><a class="nav-link"
href="${contextPath}/login" id="login-link">Login</a></li>
                                   <li class="nav-item"><a class="nav-link"
href="${contextPath}/registration" id="registration-link">Register</a></li>
                            </c:otherwise>
                     </c:choose>
              </ul>
       </nav>
</div>
```

3. footer.jsp

```
        <!-- Site footer -->
      <footer class="footer">
        <p>&copy; Mavroforakis Dimitris</p>
      </footer>
```

4.4.2 Αρχική σελίδα

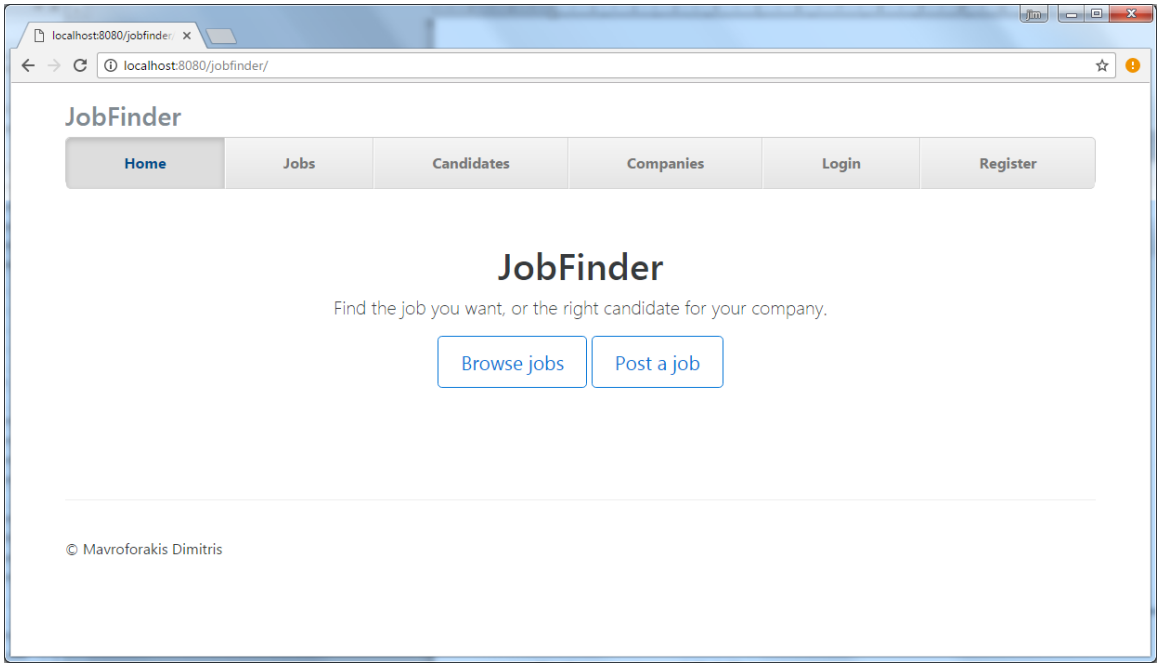
PageController.java

Δημιουργούμε την κλάση PageController.java ή οποία θα διαχειρίζεται την προβολή των σελίδων. Το link http://localhost:8080/jobfinder/ αντιστοιχεί στην μέθοδο showHome() του controller, η οποία επιστρέφει το view home.jsp .

```
@Controller
public class PageController {

	@RequestMapping(value = "/")
	public String showHome() {
		return "home";
	}

}
```

home.jsp

```
<%@ page language="java" contentType="text/html; charset=utf-8"
	pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
	<%@ include file="include/head.jsp"%>
</head>
<body>
	<div class="container">
		<%@ include file="include/navbar.jsp"%>
		<div class="jumbotron">
			<h1>JobFinder</h1>
			<p class="lead">Find the job you want, or the right candidate for your
company.</p>
			<p>
				<a class="btn btn-lg btn-success"
href="${contextPath}/listJobs" role="button">Browse jobs</a>
				<a class="btn btn-lg btn-primary" href="${contextPath}/job/add"
role="button">Post a job</a>
			</p>
		</div>
		<%@ include file="include/footer.jsp"%>
	</div>
	<script type="text/javascript">
		$("#home-link").addClass("active");
	</script>
</body>
</html>
```



Εικόνα 6 – Αρχική Σελίδα

4.4.3 Εγγραφή χρήστη

Αρχικά, δημιουργούμε την κλάση **User.java**, η οποία περιγράφει την οντότητα του χρήστη. Η τοποθεσία της κλάσης, όπως και οι υπόλοιπες κλάσης που περιγράφουν οντότητες (entities) είναι το package **com.jobfinder.model.** Το Entity Annotation (@Entity) υποδηλώνει ότι αυτή η κλάση μπορεί να αντιστοιχίσει με έναν πίνακα στην βάση δεδομένων. Το Table annotation (@Table) υποδηλώνει σε ποιόν πίνακα αντιστοιχεί η συγκεκριμένη οντότητα. Στην προκειμένη περίπτωση, η οντότητα User αντιστοιχεί στον πίνακα users της βάσης δεδομένων. Αυτό σημαίνει ότι ένα αντικείμενο τύπου User μπορεί να αντιστοιχεί σε μια εγγραφή (record) του πίνακα users.

Η κάθε ιδιότητα της κλάσης(property) αντιστοιχεί αυτόματα στο ίδιο πεδίο(field) του πίνακα. Εάν για παράδειγμα έχουμε την ιδιότητα password στην κλάση, τότε το Hibernate θα ψάξει στον πίνακα για το πεδίο password. Το @Transient annotation δηλώνει ότι το συγκεκριμένο property δεν αντιστοιχεί σε κάποιο field του πίνακα.

Η σχέση μεταξύ των οντοτήτων Χρήστης και Ρόλος είναι πολλά προς πολλά, γι αυτόν τον λόγο θα χρησιμοποιήσουμε το @ManyToMany annotation. Χρησιμοποιώντας το όρισμα fetch= FetchType.EAGER έχουμε την δυνατότητα να ανακτούμε αυτόματα τους ρόλους που αντιστοιχούν στον εκάστοτε χρήστη. Σε όλες τις σχέσεις πολλά-προς-πολλά χρησιμοποιούμε έναν ενδιάμεσο πίνακα ο οποίο περιέχει τα πρωτεύοντα κλειδιά των δυο πινάκων. Αυτός ο ενδιάμεσος πίνακας ορίζεται στο @JoinTable μαζί με τα πεδία που αντιστοιχούν στα πρωτεύοντα κλειδιά των οντοτήτων.

```
@Entity
@Table(name = "users")
public class User {
	private int id;
	private String username;
	private String password;
	private String passwordConfirm;
	private String type;

	private Set<Role> roles;

	@Id
	@GeneratedValue(strategy = GenerationType.AUTO)
	public int getId() {
		return id;
	}

	public void setId(int id) {
		this.id = id;
	}
```

```
	public String getUsername() {
		return username;
	}
	public void setUsername(String username) {
		this.username = username;
	}
	public String getPassword() {
		return password;
	}
	public void setPassword(String password) {
		this.password = password;
	}
	@Transient
	public String getPasswordConfirm() {
		return passwordConfirm;
	}

	public void setPasswordConfirm(String passwordConfirm) {
		this.passwordConfirm = passwordConfirm;
	}

	@Transient
	public String getType() {
		return type;
	}

	public void setType(String type) {
		this.type = type;
	}

	@ManyToMany(fetch = FetchType.EAGER)
	@JoinTable(name = "users_roles", joinColumns = @JoinColumn(name =
"user_id"), inverseJoinColumns = @JoinColumn(name = "role_id"))
	public Set<Role> getRoles() {
		return roles;
	}
	public void setRoles(Set<Role> roles) {
		this.roles = roles;
	}

	public String toString() {
		String result = "id: " + id + " username: " + username;
		return result;
	}

	public boolean hasRole(String roleName){
		for(Role role : roles){
			if(role.getName().equals(roleName)){
				return true;
			}
		}
		return false;
	}
```

Στην συνέχεια δημιουργούμε την κλάση **Role.java** η οποία περιγράφει έναν ρόλο χρήστη του συστήματος. Όπως είδαμε και προηγουμένως, η σχέση μεταξύ των οντοτήτων User και Role είναι πολλά προς πολλά, γι αυτόν το λόγο χρησιμοποιήσαμε το @ManyToMany annotation στο entity User. Στο entity Role, θα πρέπει να ξαναχρησιμοποιήσουμε το @ManyToMany για την ιδιότητα users, με όρισμα mappedBy=”roles” , καθώς η παρούσα κλάση αναφέρεται από την ιδιότητα roles στην οντότητα User.

```
@Entity
@Table(name = "roles")
public class Role {
    private int id;
    private String name;
    private Set<User> users;

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    @ManyToMany(mappedBy = "roles")
    public Set<User> getUsers() {
        return users;
    }

    public void setUsers(Set<User> users) {
        this.users = users;
    }
}
```

Δημιουργούμε την κλάση **Company.java** η οποία περιγράφει την οντότητα Εταιρεία. Αυτή η οντότητα περιγράφεται από τον πίνακα **companies** στην βάση δεδομένων.

```
@Entity
@Table(name = "companies")
public class Company{

	private int id;
	private String name;
	private int taxNumber;
	private String telephone;
	private String email;
	private String country;
	private String city;
	private String address;
	private int userId;

	private List<Job> jobs;

	@Id
	@GeneratedValue(strategy = GenerationType.AUTO)
	public int getId() {
		return id;
	}

	public void setId(int id) {
		this.id = id;
	}
	@Column(name="name")
	public String getName() {
		return name;
	}

	public void setName(String name) {
		this.name = name;
	}

	@Column(name = "tax_number")
	public int getTaxNumber() {
		return taxNumber;
	}

	public void setTaxNumber(int taxNumber) {
		this.taxNumber = taxNumber;
	}

	public String getTelephone() {
		return telephone;
	}

	public void setTelephone(String telephone) {
		this.telephone = telephone;
	}
```

```
	public String getEmail() {
		return email;
	}

	public void setEmail(String email) {
		this.email = email;
	}

	public String getCountry() {
		return country;
	}

	public void setCountry(String country) {
		this.country = country;
	}

	public String getCity() {
		return city;
	}

	public void setCity(String city) {
		this.city = city;
	}

	public String getAddress() {
		return address;
	}

	public void setAddress(String address) {
		this.address = address;
	}

	@Column(name = "user_id")
	public int getUserId() {
		return userId;
	}

	public void setUserId(int userId) {
		this.userId = userId;
	}
	@OneToMany(fetch=FetchType.EAGER,mappedBy="company", cascade =
CascadeType.ALL)
	public List<Job> getJobs() {
		return jobs;
	}

	public void setJobs(List<Job> jobs) {
		this.jobs = jobs;
	}

	public String toString(){
		String result = "";
		result +="id: "+id
				+" name: "+name
				+" tax number: "+taxNumber
				+" email: "+email
				+" address: "+address;
		return result;
	}
```

Στην συνέχεια δημιουργούμε την κλάση **Candidate.java,** η οποία περιγράφει τον υποψήφιο εργαζόμενο. Οι υποψήφιοι αποθηκεύονται στον πίνακα **candidates** της βάσης δεδομένων.

```
@Entity
@Table(name = "candidates")
public class Candidate {
	private int id;
	private String fname;
	private String lname;
	private int age;
	private String gender;
	private String address;
	private String email;
	private String telephone;
	private int userId;

	@Id
	@GeneratedValue(strategy = GenerationType.AUTO)
	public int getId() {
		return id;
	}
	public void setId(int id) {
		this.id = id;
	}
	public String getFname() {
		return fname;
	}
	public void setFname(String fname) {
		this.fname = fname;
	}
	public String getLname() {
		return lname;
	}
	public void setLname(String lname) {
		this.lname = lname;
	}
	public int getAge() {
		return age;
	}
	public void setAge(int age) {
		this.age = age;
	}
	public String getGender() {
		return gender;
	}
	public void setGender(String gender) {
		this.gender = gender;
	}

	public String getAddress() {

		return address;
	}
	public void setAddress(String address) {
		this.address = address;
	}
```

```
	@Column(name = "user_id")
	public int getUserId() {
		return userId;
	}

	public void setUserId(int userId) {
		this.userId = userId;
	}
	@OneToMany(fetch=FetchType.EAGER,mappedBy="company", cascade =
CascadeType.ALL)
	public List<Job> getJobs() {
		return jobs;
	}

	public void setJobs(List<Job> jobs) {
		this.jobs = jobs;
	}

	public String toString(){
		String result = "";
		result +="id: "+id
				+" name: "+name
				+" tax number: "+taxNumber
				+" email: "+email
				+" address: "+address;
		return result;
	}
}
```

Στην κλάση **Consts.java** αποθηκεύουμε τις σταθερές που χρειαζόμαστε στην εφαρμογή μας.

```
public class Consts {

	public static final String ROLE_COMPANY = "ROLE_COMPANY";

	public static final String ROLE_CANDIDATE = "ROLE_CANDIDATE";

}
```

Μέσω του UserDAO θα έχουμε πρόσβαση στα δεδομένα που σχετίζονται με την κλάση User. Η κλάση UserDAO κληρονομεί τις μεθόδους της κλάσης GenericDAO, δηλάδη μεθόδους για CRUD λειτουργίες όπως εύρεση οντότητας, επεξεργασία, διαγραφή κτλ. Στην συνέχεια ορίζουμε τις μεθόδους findByUsername και save οι οποίες θα μας χρειαστούν αργότερα.

UserDAO.java:

```
package com.jobfinder.dao;

import com.jobfinder.model.User;

public interface UserDAO extends GenericDAO<User,Integer>{

	public User findByUsername(String uname);

	public User save(User user);

}
```

UserDAOImpl.java:

```
package com.jobfinder.dao.impl;

import org.apache.log4j.Logger;
import org.hibernate.Query;
import org.springframework.stereotype.Repository;

import com.jobfinder.dao.UserDAO;
import com.jobfinder.model.User;

@Repository
public class UserDAOImpl extends GenericDAOImpl<User,Integer> implements UserDAO{

	private static Logger logger = Logger.getLogger(UserDAOImpl.class);

	@Override
	public User findByUsername(String uname) {
		logger.debug("findByUsername: "+uname);
		String hql = "from User where username = :uname";
		Query query = getSession().createQuery(hql);
		query.setParameter("uname",uname);
		User user = (User) query.uniqueResult();
		return user;
	}

	@Override
	public User save(User user) {
		// save Entity and get the generated id
		int generatedId = (int) getSession().save(user);
		//return generated entity
		return find(generatedId);
	}

}
```

Παρόμοια με το UserDAO, στο Service Layer η κλάση UserService κληρονομεί τις μεθόδους της GenericService οι οποίες καλούν τις μεθόδους του GenericDAO για τις βασικές CRUD λειτουργίες.

UserService.java:

```
package com.jobfinder.service;

import com.jobfinder.model.User;

public interface UserService extends GenericService<User,Integer>{
    public void save(User user);

    public User findByUsername(String username);

    public String getLoggedInUserName();

    public User getUserById(int userId);

    public User getUserFromSession();

}
```

UserServiceImpl.java:

```
package com.jobfinder.service.Impl;

@Service
public class UserServiceImpl extends GenericServiceImpl<User,Integer> implements
UserService {

       private UserDAO userDAO;

    @Autowired
    private BCryptPasswordEncoder bCryptPasswordEncoder;

    @Autowired
    private CompanyService companyService;

    @Autowired
    private CandidateService candidateService;

    @Autowired
    private RoleService roleService;

       private static Logger logger = Logger.getLogger(UserServiceImpl.class);

       @Autowired
       public UserServiceImpl(@Qualifier("userDAOImpl") GenericDAO<User,Integer>
genericDAO){
              super(genericDAO);
              this.userDAO = (UserDAO) genericDAO;
       }
```

```
@Override
    @Transactional
    public void save(User user) {
       logger.debug("saving user: "+user.toString());
       // encrypt password
        user.setPassword(bCryptPasswordEncoder.encode(user.getPassword()));
        if(user.getType().equals(Consts.ROLE_COMPANY)){
              Role companyRole = roleService.getRoleByName(Consts.ROLE_COMPANY);
              Set<Role> userRoles = new HashSet<Role>();
              userRoles.add(companyRole);
             user.setRoles(userRoles);
             User savedUser = userDAO.save(user);

              Company company = new Company();
              company.setUserId(savedUser.getId());
              logger.debug("saving new company");
              companyService.add(company);
        }else if(user.getType().equals(Consts.ROLE_CANDIDATE)){
              Role candidateRole = roleService.getRoleByName(Consts.ROLE_CANDIDATE);
              Set<Role> userRoles = new HashSet<Role>();
              userRoles.add(candidateRole);
             user.setRoles(userRoles);
             User savedUser = userDAO.save(user);

             Candidate candidate = new Candidate();
              candidate.setUserId(savedUser.getId());
              logger.debug("saving new candidate");
              candidateService.add(candidate);
        }
    }

    @Override
    @Transactional
    public User findByUsername(String username) {
        User user = this.userDAO.findByUsername(username);
        return user;
    }

       @Override
       @Transactional
       public String getLoggedInUserName() {
              Authentication auth =
SecurityContextHolder.getContext().getAuthentication();
              String name = auth.getName(); // get logged in username
              return name;
       }

    @Override
    @Transactional
    public User getUserFromSession() {
       return findByUsername(getLoggedInUserName());
    }
       @Override
       @Transactional
       public User getUserById(int userId){
              return userDAO.find(userId);
       }

}
```

CandidateDAO.java:

```
package com.jobfinder.dao;

import com.jobfinder.model.Candidate;

public interface CandidateDAO extends GenericDAO<Candidate,Integer>{

	public Candidate findByUserId(Integer id);

}
```

CandidateDAOImpl.java:

```
package com.jobfinder.dao.impl;

import org.springframework.stereotype.Repository;

import com.jobfinder.dao.CandidateDAO;
import com.jobfinder.model.Candidate;

@Repository
public class CandidateDAOImpl extends GenericDAOImpl<Candidate,Integer> implements
CandidateDAO{

	@Override
	public Candidate findByUserId(Integer id) {
		String hql = "from Candidate cand where cand.userId = :userId ";
		return (Candidate)
getSession().createQuery(hql).setParameter("userId", id).uniqueResult();
	}

}
```

CandidateService.java:

```
package com.jobfinder.service;

import com.jobfinder.model.Candidate;

public interface CandidateService extends GenericService<Candidate,Integer> {

	public Candidate findByUserId(Integer id);

}
```

CandidateServiceImpl.java:

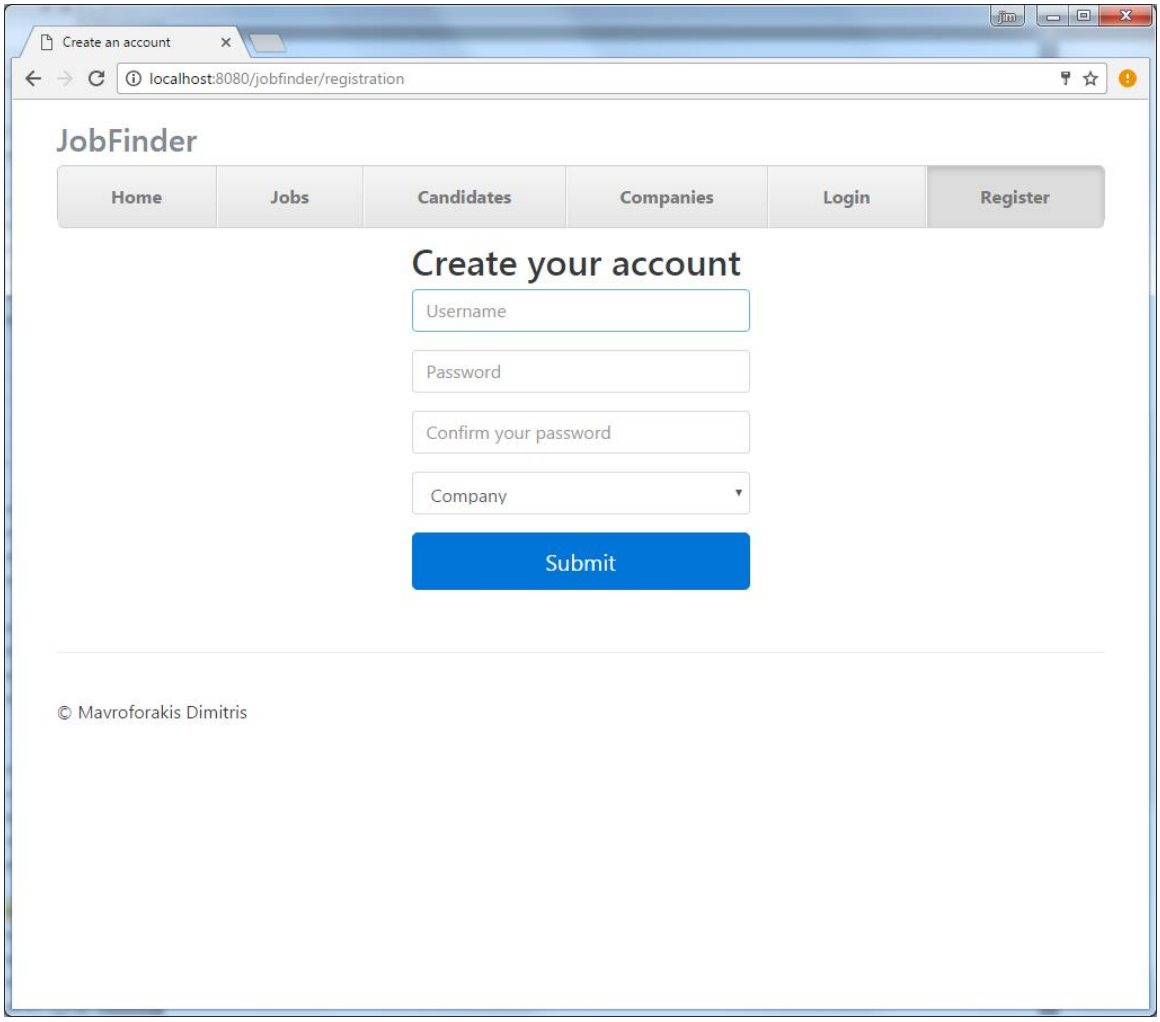
```
package com.jobfinder.service.Impl;


@Service
public class CandidateServiceImpl extends GenericServiceImpl<Candidate,Integer>
implements CandidateService {

	private CandidateDAO candidateDAO;

	public CandidateServiceImpl(){
	}

	@Autowired
	public CandidateServiceImpl(@Qualifier("candidateDAOImpl")
GenericDAO<Candidate,Integer> genericDAO){
		super(genericDAO);
		this.candidateDAO = (CandidateDAO) genericDAO;
	}

	@Override
	@Transactional
	public Candidate findByUserId(Integer id) {
		return candidateDAO.findByUserId(id);
	}

}
```

Πατώντας τον σύνδεσμο Register από την μπάρα περιήγησης, ο χρήστης κατευθύνεται στο URL /registration. Η κλάση του UserController που αντιστοιχεί σε αυτό το URL είναι η παρακάτω:

```
@RequestMapping(value = "/registration", method = RequestMethod.GET)
public String prepareRegistration(Model model) {
	model.addAttribute("userForm", new User());
	return "registration";
}
```

Η registration.jsp είναι η σελίδα που θα επιστραφεί:

```
<form:form method="POST" modelAttribute="userForm" class="form-signin">
	<h2 class="form-signin-heading">Create your account</h2>
	<spring:bind path="username">
	<div class="form-group ${status.error ? 'has-error' : ''}">
		<form:input type="text" path="username" class="form-control"
		placeholder="Username" autofocus="true"></form:input>
		<form:errors path="username"></form:errors>
	</div>
	</spring:bind>

	<spring:bind path="password">
		<div class="form-group ${status.error ? 'has-error' : ''}">
		<form:input type="password" path="password" class="form-control"
					placeholder="Password"></form:input>
		<form:errors path="password"></form:errors>
		</div>
	</spring:bind>

	<spring:bind path="passwordConfirm">
		<div class="form-group ${status.error ? 'has-error' : ''}">
			<form:input type="password" path="passwordConfirm"
		class="form-control" placeholder="Confirm your
	password"></form:input>
			<form:errors path="passwordConfirm"></form:errors>
		</div>
	</spring:bind>
	<spring:bind path="type">
		<div class="form-group ${status.error ? 'has-error' : ''}">
		<form:select type="text" path="type" class="form-control" >
			<form:option value="<%=Consts.ROLE_COMPANY%>" label="Company"/>
		<form:option value="<%=Consts.ROLE_CANDIDATE%>" label="Candidate"/>
		</form:select>
		</div>
	</spring:bind>
	<button class="btn btn-lg btn-primary btn-block"
	type="submit">Submit</button>
</form:form>
```



Εικόνα 7 – Εγγραφή Χρήστη

Όταν ο χρήστης συμπληρώσει την φόρμα και θα πατήσει το κουμπί submit, τότε θα κληθεί η μέθοδος registration του UserController. Η μέθοδος validate του userValidator ελέγχει τα δεδομένα εισόδου και εάν υπάρχουν λάθη τότε επιστρέφεται και πάλι η σελίδα εγγραφής. Εάν δεν υπάρχουν λάθη τότε μέσω της μεθόδου save του UserService θα αποθηκευτεί ο χρήστης στην βάση δεδομένων. Στην συνέχεια πραγματοποιείται αυτόματη σύνδεση μέσω της μεθόδου autologin του SecurityService ώστε να μην χρειαστεί ο χρήστης να εισάγει ξανά τα στοιχεία του. Αυτή η μέθοδος δεν θα επιστρέψει σελίδα jsp, αλλά μέσω του redirect θα γίνει ανακατεύθυνση στο URL user/dashboard, δηλαδή το πάνελ του χρήστη.

```
@RequestMapping(value = "/registration", method = RequestMethod.POST)
public String registration(@ModelAttribute("userForm") User userForm, BindingResult
bindingResult, Model model) {
	userValidator.validate(userForm, bindingResult);
	if (bindingResult.hasErrors()) {
		return "registration";
	}
	userService.save(userForm);
	securityService.autologin(userForm.getUsername(),
	userForm.getPasswordConfirm());

	return "redirect:/user/dashboard";
}
```

Η μέθοδος του UserController που αντιστοιχεί στο request **/user/dashboard**:

```
@RequestMapping(value = "/user/dashboard")
public String showDashboard(Model model){
	User loggedUsr =  userService.getUserFromSession();
	if(loggedUsr != null){
		model.addAttribute("user", loggedUsr);
		if(loggedUsr.hasRole(Consts.ROLE_COMPANY)){
			Company company = companyService.findByUserId(loggedUsr.getId());
			model.addAttribute("company", company);
		}else if(loggedUsr.hasRole(Consts.ROLE_CANDIDATE)){
			Candidate candidate = candidateService.findByUserId(loggedUsr.getId());
			model.addAttribute("candidate", candidate);
		}
	}

	return "user/dashboard";
}
```

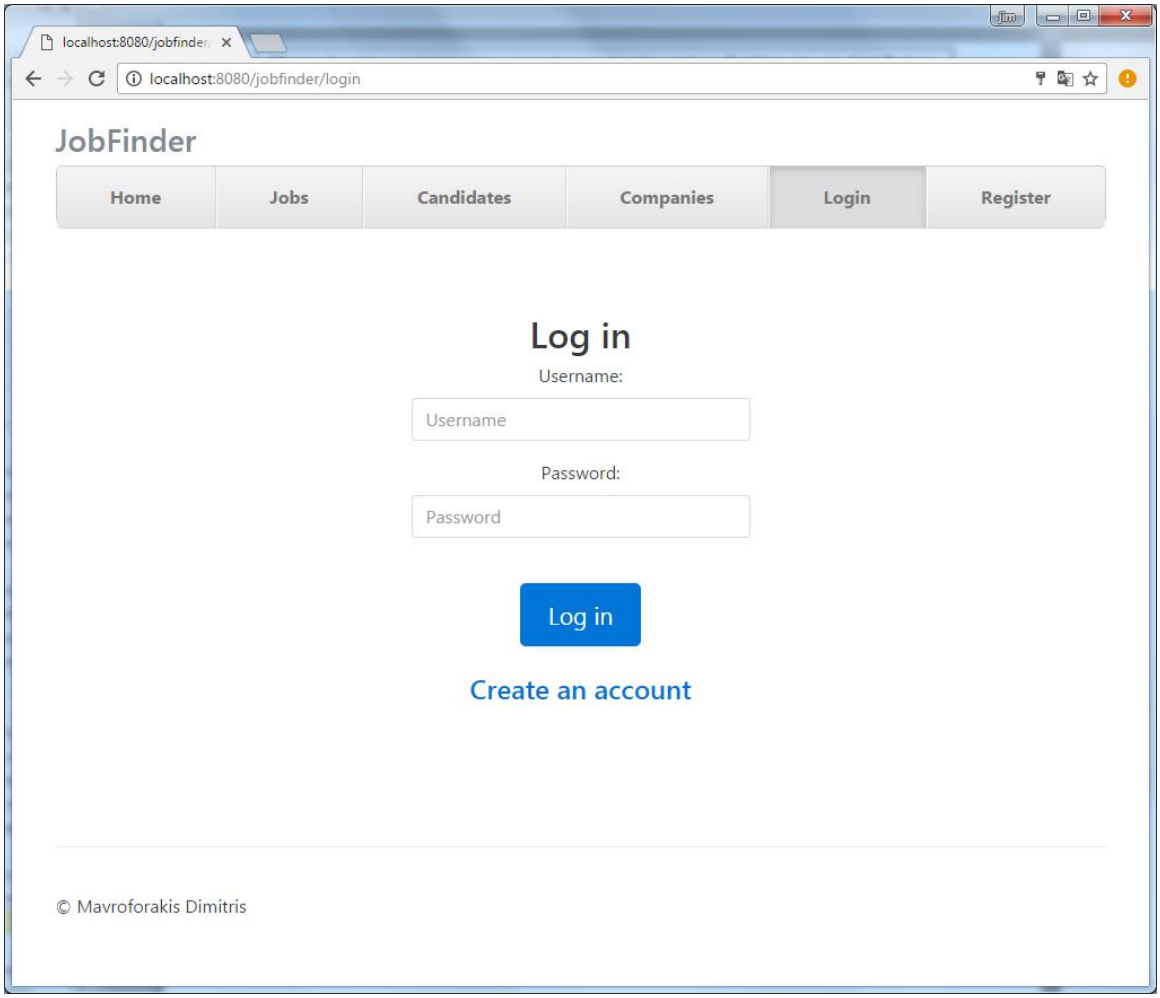
4.4.4 Σύνδεση χρήστη

Επιλέγοντας τον σύνδεσμο /**login** ο χρήστης κατευθύνεται στην σελίδα όπου θα εισάγει τα στοιχεία του λογαριασμού του για να συνδεθεί στο σύστημα.

```
@RequestMapping(value = "/login", method = RequestMethod.GET)
public String login(Model model, String error, String logout) {
	if (error != null)
		model.addAttribute("error", "Your username and password is invalid.");

	if (logout != null)
		model.addAttribute("message", "You have been logged out successfully.");

		return "login";
	}
```

login.jsp:

```
<form method="POST" action="${contextPath}/login"
                        class="form-signin" action="#">
	<h2 class="form-signin-heading">Log in</h2>
	<div class="form-group ${error != null ? 'has-error' : ''}">
		<label>Username:</label>
		<input name="username" type="text" class="form-control"
		placeholder="Username" autofocus="true" />
	</div>
	<div class="form-group">
		<label>Password:</label>
		<input name="password" type="password" class="form-control"
		placeholder="Password" /> <span>${error}</span>
	</div>
	<input type="hidden" name="${_csrf.parameterName}"
                                    value="${_csrf.token}" /> <br>
	<div class='col-md-offset-3'>
		<button type="submit" class="btn btn-primary btn-lg">Log
		in</button>
	</div>
	<br>
	<h4 class="text-center">
		<a href="${contextPath}/registration">Create an account</a>
	</h4>
</form>
```



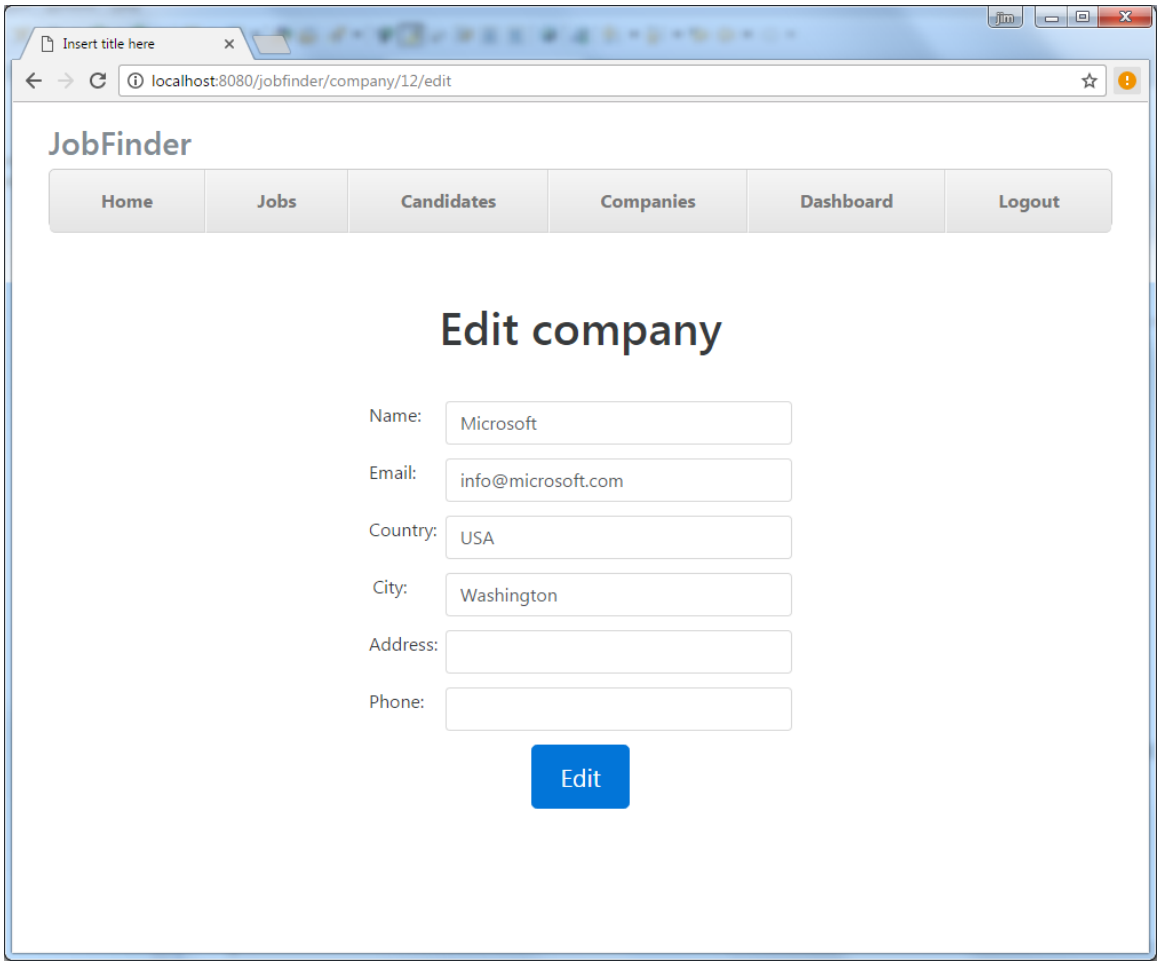
Εικόνα 8 – Σύνδεση χρήστη

4.4.5 Επεξεργασία προφίλ χρήστη

Πατώντας το κουμπί **Edit my profile** από τον πίνακα διαχείρισης, ο χρήστης μπορεί να επεξεργαστεί το προφίλ του. Εάν ο χρήστης είναι εταιρεία, τότε το URL είναι της μορφής **/company/{id}/edit**, οπου id ο κωδικός της εταιρείας. Εάν ο συνδεδεμένος χρήστης είναι υποψήφιος τότε το URL είναι της μορφής **/candidate/{id}/edit**, όπου id ο κωδικός του υποψηφίου.

```
package com.jobfinder.controller;

@Controller
public class CompanyController {

	@Autowired
	private CompanyService companyService;

	@Autowired
	private UserService userService;

	@RequestMapping(value="/company/{id}/edit",method=RequestMethod.GET)
	public String prepareEditCompany(@PathVariable("id") int id, Model model
	){
		Company company = companyService.find(id);
		model.addAttribute("company",company);
		return "company/edit";
	}

	@RequestMapping(value="/company/{id}/edit",method=RequestMethod.POST)
	public String editCompany(@ModelAttribute("company") Company
	company,BindingResult bindingResult, Model model ){
		company.setUserId(userService.getUserFromSession().getId());
		companyService.update(company);
		return "company/edit";
	}

}
```

Τα πεδία του προφίλ του που μπορεί να επεξεργαστεί ένας χρήστης που έχει συνδεθεί ως εταιρεία είναι Ονομα, Email, Χώρα, Πόλη, Διεύθυνση, Τηλέφωνο

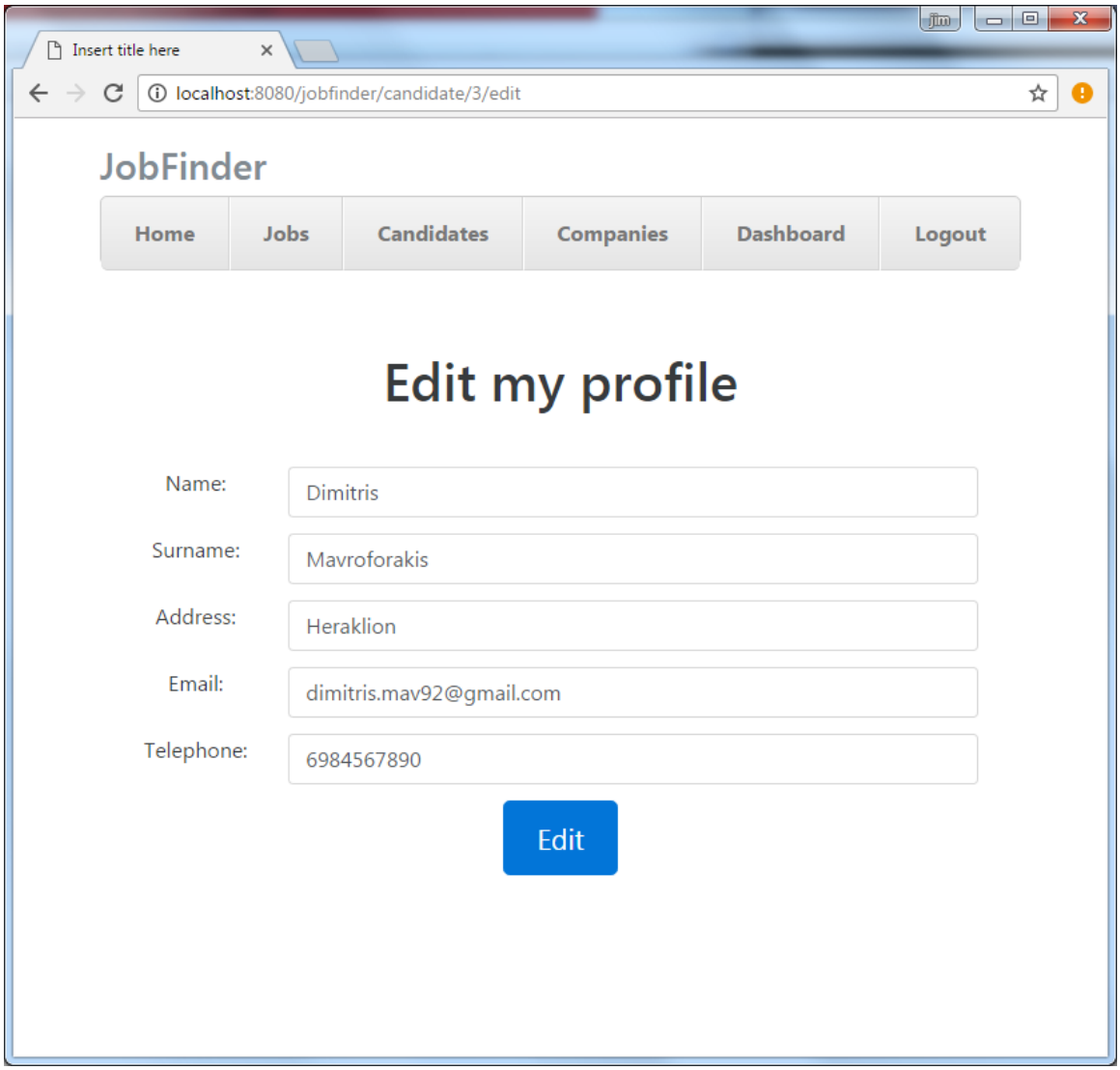


Εικόνα 9 – Επεξεργασία προφιλ εταιρείας

Στην περίπτωση που ο χρήστης έχει εγγραφεί ως υποψήφιος εργαζόμενος, η διαχείρηση των requests θα γίνει μέσω του CandidateController.

```
package com.jobfinder.controller;

@Controller
public class CandidateController {
	@Autowired
	private CandidateService candidateService;

	@Autowired
	private UserService userService;

	@RequestMapping(value = "/candidate/{id}/edit", method = RequestMethod.GET)
	public String prepareEditCandidate(@PathVariable("id") int id, Model model)
	{
		Candidate candidate = candidateService.find(id);
		model.addAttribute("candidate", candidate);
		return "candidate/edit";
	}

	@RequestMapping(value = "/candidate/{id}/edit", method =
RequestMethod.POST)
	public String editCandidate(@ModelAttribute("candidate") Candidate
	candidate, BindingResult result, Model model) {
		candidate.setUserId(userService.getUserFromSession().getId());
		candidateService.update(candidate);
		model.addAttribute("candidate", candidate);
		return "candidate/edit";
	}

}
```

Ένας χρήστης που έχει συνδεθεί ως υποψήφιος, μπορεί να επεξεργαστεί το Ονομα, Επώνυμο, Διεύθυνση, email και τηλέφωνο.



Εικόνα 10 – Επεξεργασία προφιλ υποψηφίου

4.4.6 Δημοσίευση θέσης εργασίας

Εφόσον ο χρήστης έχει συνδεθεί στον λογαριασμό του ως εταιρεία, έχει την δυνατότητα να δημοσιεύσει διαθέσιμες θέσεις εργασίας. Πατώντας το κουμπί **Add New Job** από τον πίνακα του χρήστη, το URL που θα κατευθυνθεί είναι το **/job/add.** Η κλάση η οποία περιγράφει μια θέση εργασίας είναι η **Job** και τοποθετείται στο package model.

Χρησιμοποιώντας το `@Table(name="jobs")` το Hibernate αντιστοιχεί αυτήν την κλάση με τον πίνακα jobs της βάσης δεδομένων. Η κάθε ιδιότητα της κλάσης αντιστοιχεί αυτόματα και σε ένα πεδίο του πίνακα jobs με το ίδιο όνομα.

```
package com.jobfinder.model;


@Entity
@Table(name="jobs")
public class Job {

	private int id;
	private String title;
	private String description;
	private float salary;
	private String employmentType;
	private String experience;
	private JobCategory category;
	private String location;
	private String telephone;
	private String email;
	private Date datePosted;

	private Company company;
	private List<JobApplication> applications;

	/*
	 * Getters and setters
	 * */

}
```

Δημιουργούμε την κλάση **JobController.java** ώστε να διαχειρίζεται τα requests που αφορούν θέσεις εργασίας. Η μέθοδος που καλείται για να μεταβεί ο χρήστης στην σελίδα όπου θα καταχωρήσει μια νέα θέση είναι η **prepareAddJob.** Όταν ο χρήστης συμπληρώσει τα απαιτούμενα πεδία και πατήσει το κουμπί καταχώρησης, τότε θα κληθεί η μέθοδος **addJob**. Και οι δύο μέθοδοι αντιστοιχούν στα ίδιο URL, όμως διαφέρει το είδος του request. Η φόρμα που καταχωρεί ο χρήστης χρησιμοποιεί την POST method, ενώ πατώντας το κουμπί για την προβολή της σελίδας καταχώρησης, χρησιμοποιείται η GET method.

```
package com.jobfinder.controller;

@Controller
public class JobController {

	@Autowired
	private JobService jobService;

	@Autowired
	private JobCategoryService jobCategoryService;

	@RequestMapping(value = "/job/add", method = RequestMethod.GET)
	public String prepareAddJob(Model model) {
		Job job = new Job();
		model.addAttribute("jobForm", job);
		model.addAttribute("jobCategories", jobCategoryService.getAll());
		return "job/add";
	}

	@RequestMapping(value = "/job/add", method = RequestMethod.POST)
	public String addJob(@ModelAttribute("jobForm") Job job, BindingResult result,
	Model model) {
		model.addAttribute("job",job);
		jobService.addJob(job);
		return "redirect:/user/dashboard";
	}
}
```

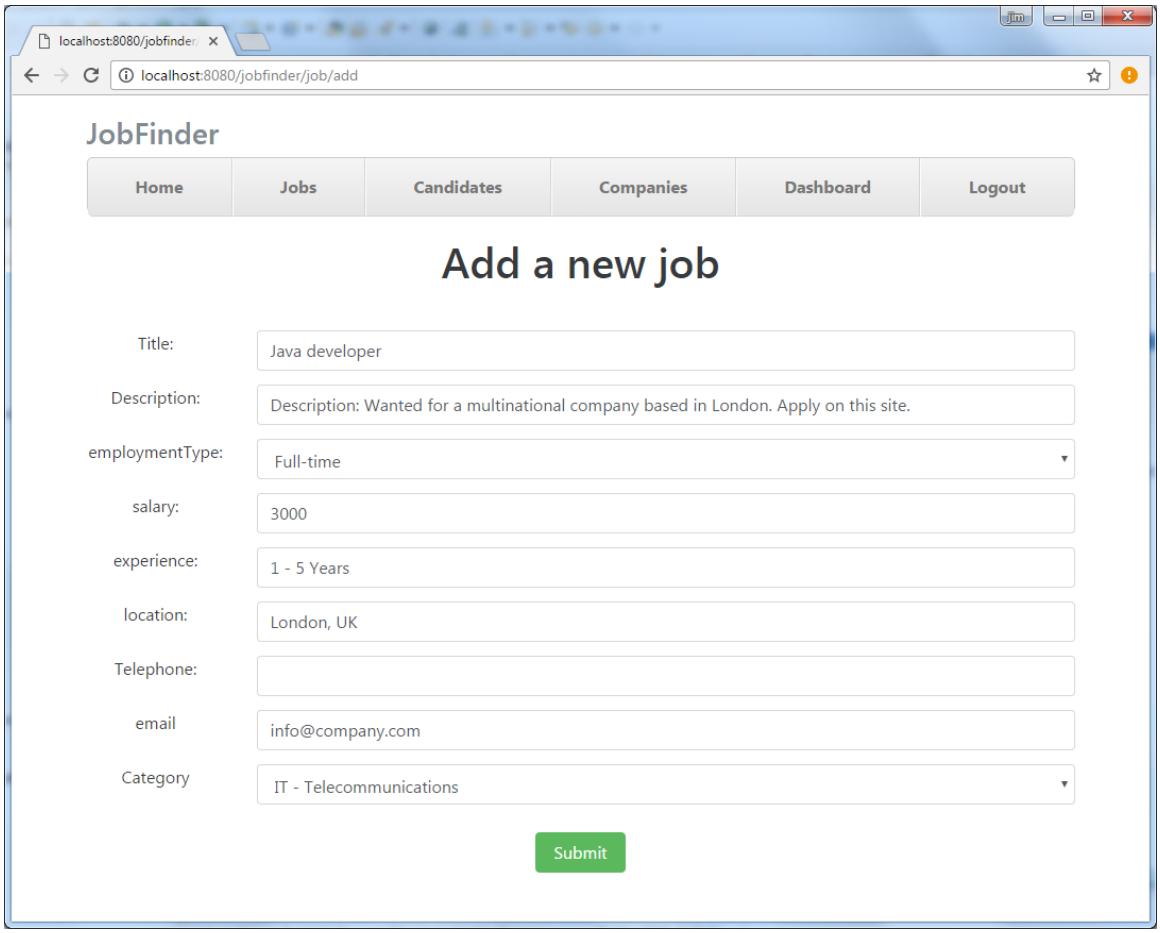
Δημιουργούμε τις κλάσεις JobService και JobServiceImpl, οι οποίες υλοποιούν το business logic που σχετίζεται με τις θέσεις εργασίας. Η ανάκτηση και καταχώρηση δεδομένων γίνεται μέσω του Repository Layer και των κλάσεων JobDAO και JobDAOImpl.

```
package com.jobfinder.dao;

public interface JobDAO extends GenericDAO<Job,Integer>{

	public List<Job> listJobsByCompany(Company company);

}
```

```
package com.jobfinder.dao.impl;

@Repository
public class JobDAOImpl extends GenericDAOImpl<Job,Integer> implements JobDAO{

	@SuppressWarnings("unchecked")
	public List<Job> listJobsByCompany(Company company) {
		String hql = "from Job where company = :company ";
		return getSession().createQuery(hql).setParameter("company",
		company).list();
	}

}
```

```
 package com.jobfinder.service;

 public interface JobService extends GenericService<Job,Integer>{

	public List<Job> listJobsByCompany(Company company);

	public void addJob(Job job);

 }
```

```
 package com.jobfinder.service.Impl;

 @Service
 public class JobServiceImpl extends GenericServiceImpl<Job,Integer>  implements
 JobService{
	private JobDAO jobDAO;

	public JobServiceImpl(){

	}
	@Autowired
	public JobServiceImpl(@Qualifier("jobDAOImpl") GenericDAO<Job,Integer>
 genericDAO){
		super(genericDAO);
		this.jobDAO = (JobDAO) genericDAO;
	}
	@Transactional
	public void addJob(Job job) {
		User user = userService.getUserFromSession();
		Company company = companyService.findByUserId(user.getId());
		job.setCompany(company);
		job.setDatePosted(new Date());
		jobDAO.add(job);
	}

	@Override
	@Transactional
	public List<Job> listJobsByCompany(Company company) {
		return jobDAO.listJobsByCompany(company);
	}

 }
```



Εικόνα 11 – Δημοσίευση θέσης εργασίας

4.4.7 Προβολή εγγεγραμμένων εταιριών

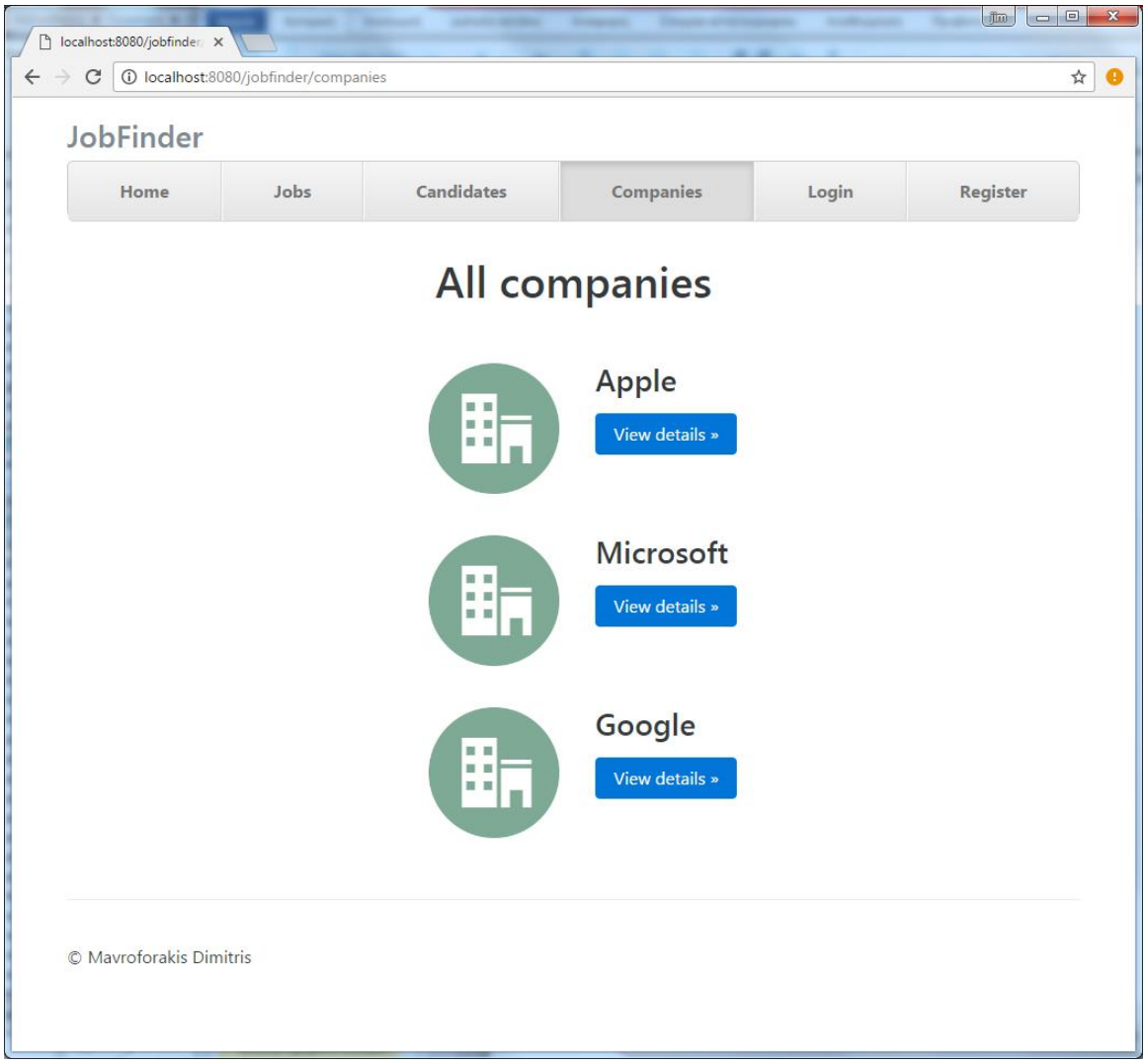
Ο σύνδεσμος που έχουμε ορίσει στην μπάρα περιήγησης για την προβολή των εταιριών είναι ο **/companies,** οπότε στην κλάση **CompanyController.java** θα προσθέσουμε μια μέθοδο που θα αντιστοιχεί σε αυτό το request**.** Η μέθοδος **getCompanies()** ανακτά όλες τις εγγεγραμμένες εταιρίες που υπάρχουν στην βάση δεδομένων μέσω του CompanyService και της μεθόδου **getAll()**, η οποία κληρονομείται από την κλάση GenericService, και στην συνέχεια τις προσθέτει σε μια λίστα αντικειμένων τύπου Company. Αυτή η λίστα θα προστεθεί ως χαρακτηριστικό στο αντικείμενο model ώστε να είναι διαθέσιμη στο view. Η μέθοδος του Controller επιστρέφει το όνομα του view το οποίο στην συνέχεια αναλύεται από τον view resolver. Σε αυτήν την περίπτωση το αρχείο jsp που θα αναπαραστήσει τα δεδομένα είναι το **list.jsp.**

```
package com.jobfinder.controller;

@Controller
public class CompanyController {

	@Autowired
	private CompanyService companyService;

	@RequestMapping(value="/companies")
	public String getCompanies(Model model){
		List<Company> companies = companyService.getAll();
		model.addAttribute("companies",companies);
		return "company/list";
	}

}
```

/src/main/webapp/WEB-INF/views/company/list.jsp:

```
<%@ include file="../include/head.jsp"%>
<body>
	<div class="container">
		<%@ include file="../include/navbar.jsp"%>
		<div style="padding-top: 36px; text-align: center">
			<h1>All companies</h1>
		</div>
		<br>
		<div class="row">
			<div class="col-lg-4"></div>
			<div class="col-lg-8">
				<c:forEach items="${companies}" var="company">
					<div class="job-container">
						<div class="job-image-container">
							<img
src="${contextPath}/resources/images/company-icon.png"	width="120"
height="120" />
						</div>
						<div class="job-info-container">
```

```
                                        <h3>${company.name }</h3>
                                        <p>${company.address }</p>
                                        <p>${company.telephone }</p>
                                        <p>${company.email }</p>
                                        <p>
                                        <a class="btn btn-primary"
href="${contextPath}/company/${company.id}" role="button">View details
&raquo;</a></p>
                                    </div>
                                </div>
                                <br>
                            </c:forEach>
                        </div>
                </div>
                <!-- footer -->
                <%@ include file="../include/footer.jsp"%>
        </div>
        <script type="text/javascript">
                $("#companies-link").addClass("active");
        </script>
```



Εικόνα 12 – Προβολή εγγεγραμμένων εταιριών

Προβολή Εταιρείας

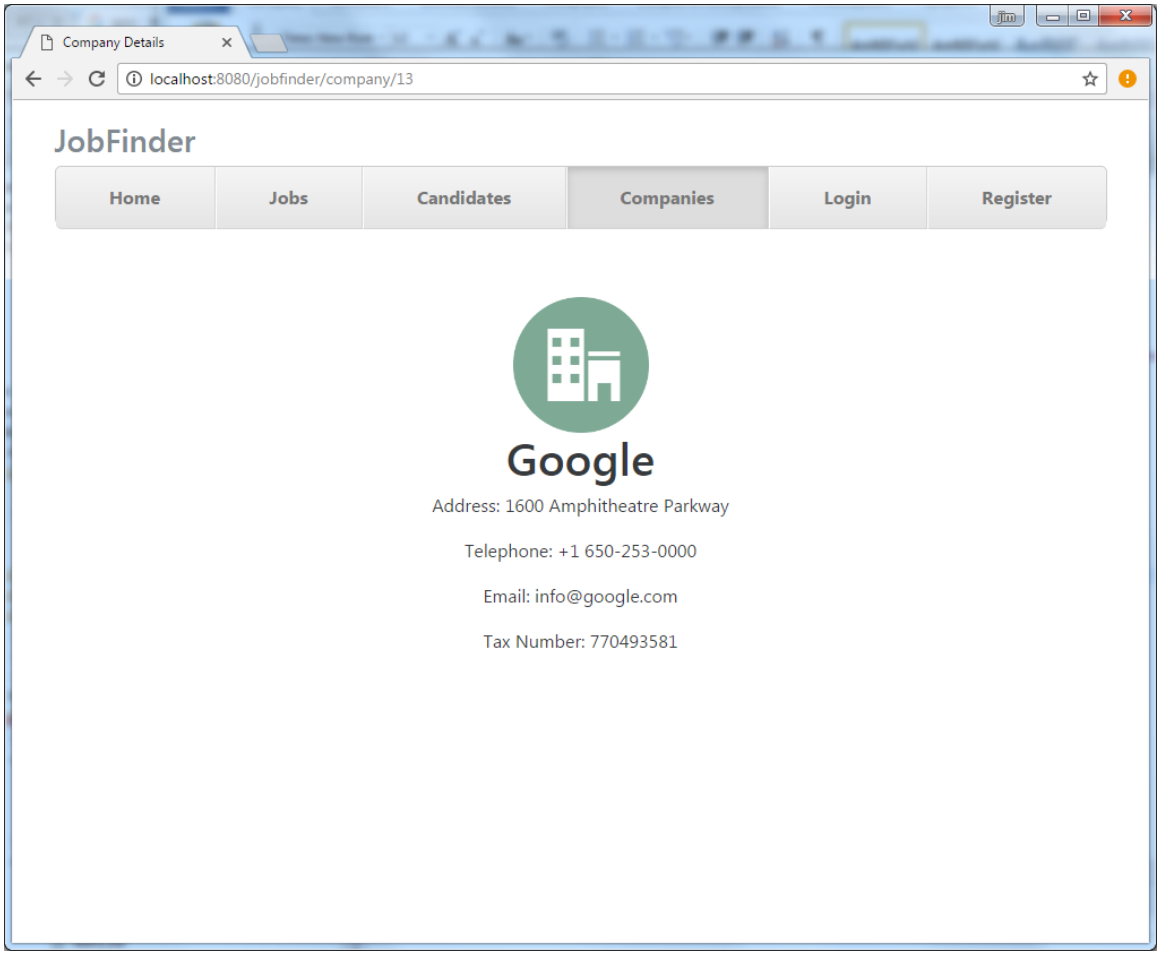
Πατώντας το κουμπί **View details**, ο χρήστης θα μπορεί να δει λεπτομέρειες σχετικά με την συγκεκριμένη εταιρεία. Τα urls που αντιστοιχούν στην προβολή του προφίλ μιας εταιρείας είναι της μορφής **/company/{id}** , όπου το **id** είναι ο κωδικός της εταιρείας όπως αναφέρεται στην βάση δεδομένων. Δημιουργούμε την μέθοδο **getCompany()** στην κλάση CompanyController.java , όπου θα αντιστοιχούν τα urls αυτού του τύπου. Χρησιμοποιόντας το @PathVariable μπορούμε να αντιστοιχίσουμε μια μεταβλητή τιμή από το Url σε μια ορισμένη μεταβλητή. Στην προκειμένη περίπτωση, αν έχουμε ένα url της μορφής **/company/12 ,** τότε η μεταβλητή **id** θα πάρει την τιμή **12.**

```
@RequestMapping(value="/company/{id}")
public String getCompany(@PathVariable("id") int id, Model model){
	Company company = companyService.find(id);
	model.addAttribute("company",company);
	return "company/show";
}
```

Η ανάκτηση μιας εγγραφής (π.χ η εταιρεία με κωδικό 12) γίνεται καλώντας την μέθοδο **find** με όρισμα το id, από το CompanyService, η οποία είναι μια γενική μέθοδος ανάκτησης αντικειμένου που κληρονομείται από την κλάση GenericService. Το jsp αρχείο που θα αναπαραστήσει τα δεδομένα είναι το **/views/company/show.jsp**.

show.jsp:

```
<html>
<head>
<%@ include file="../include/head.jsp"%>
<title>Company Details</title>
</head>
	<body>
	<div class="container">
		<%@ include file="../include/navbar.jsp"%>
			<div class="row">
				<div class="jumbotron">
					<img
src="${contextPath}/resources/images/company-icon.png" width="120" height="120"
/>
					<h1>${company.name }</h1>
					<p>Address: ${company.address}</p>
					<p>Telephone: ${company.telephone}</p>
					<p>Email: ${company.email}</p>
					<p>Tax Number: ${company.taxNumber}</p>
				</div>
			</div>
		</div>
		<script type="text/javascript">
			$("#companies-link").addClass("active");
		</script>
	</body>
</html>
```



Εικόνα 13 – Προβολή εταιρείας

4.4.8 Προβολή εγγεγραμμένων υποψηφίων

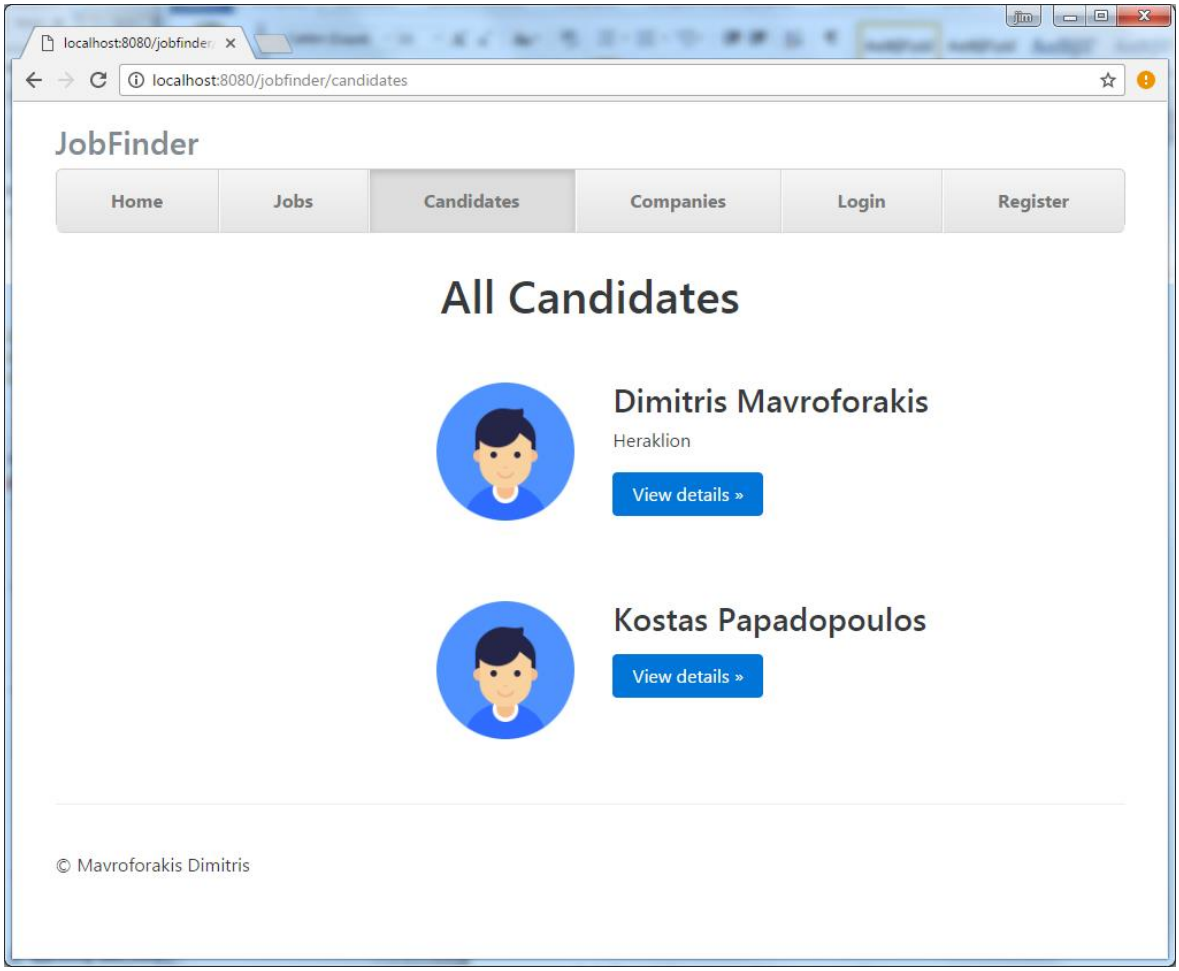
Παρόμοια με την προβολή των εταιριών, θα δημιουργήσουμε μέθοδο που θα διαχειρίζεται το url για την προβολή των υποψηφίων (**/candidates**), δημιουργώντας την μέθοδο **getCandidates** στην κλάση **CandidateController.java.** Η ανάκτηση των εγγεγραμμένων υποψηφίων γίνεται καλώντας την μέθοδο **getAll()** από την κλάση CandidateService.

```
@Controller
public class CandidateController {
	@Autowired
	private CandidateService candidateService;

	@RequestMapping(value = "/candidates")
	public String showCandidates(Model model) {
		List<Candidate> candidates = candidateService.getAll();
		model.addAttribute("candidates", candidates);
		return "candidate/list";
	}

}
```

Το view που θα αναπαραστήσει τα δεδομένα είναι το **/candidate/list.jsp:**

```
<html>
<head>
<%@ include file="../include/head.jsp"%>
</head>
<body>
	<div class="container">
		<%@ include file="../include/navbar.jsp"%>
		<div style="padding-top: 36px; text-align: center">
			<h1>All Candidates</h1>
		</div>
		<br>
		<div class="row">
			<div class="col-lg-4"></div>
				<div class="col-lg-8">
					<c:forEach items="${candidates}" var="candidate">
						<div class="job-container">
							<div class="job-image-container">
				<img src="${contextPath}/resources/images/candidate-
				icon.png" width="120" height="120" />
							</div>
							<div class="job-info-container">
								<h3>${candidate.fname}
								${candidate.lname}</h3>
								<p>${candidate.address}</p>
```

```
<p>
<a class="btn btn-primary" href="${contextPath}/candidate/${candidate.id}"
role="button">View details &raquo;</a>
</p>
                                                    </div>
                                            </div>
                                            <br>
                                        </c:forEach>
                                    </div>
                            </div>
                                    <!-- footer -->
                            <%@ include file="../include/footer.jsp"%>
                    </div>
                    <script type="text/javascript">
                            $("#candidates-link").addClass("active");
                    </script>
            </body>
            </html>
```



Εικόνα 14 – Προβολή εγγεγραμμένων υποψηφίων

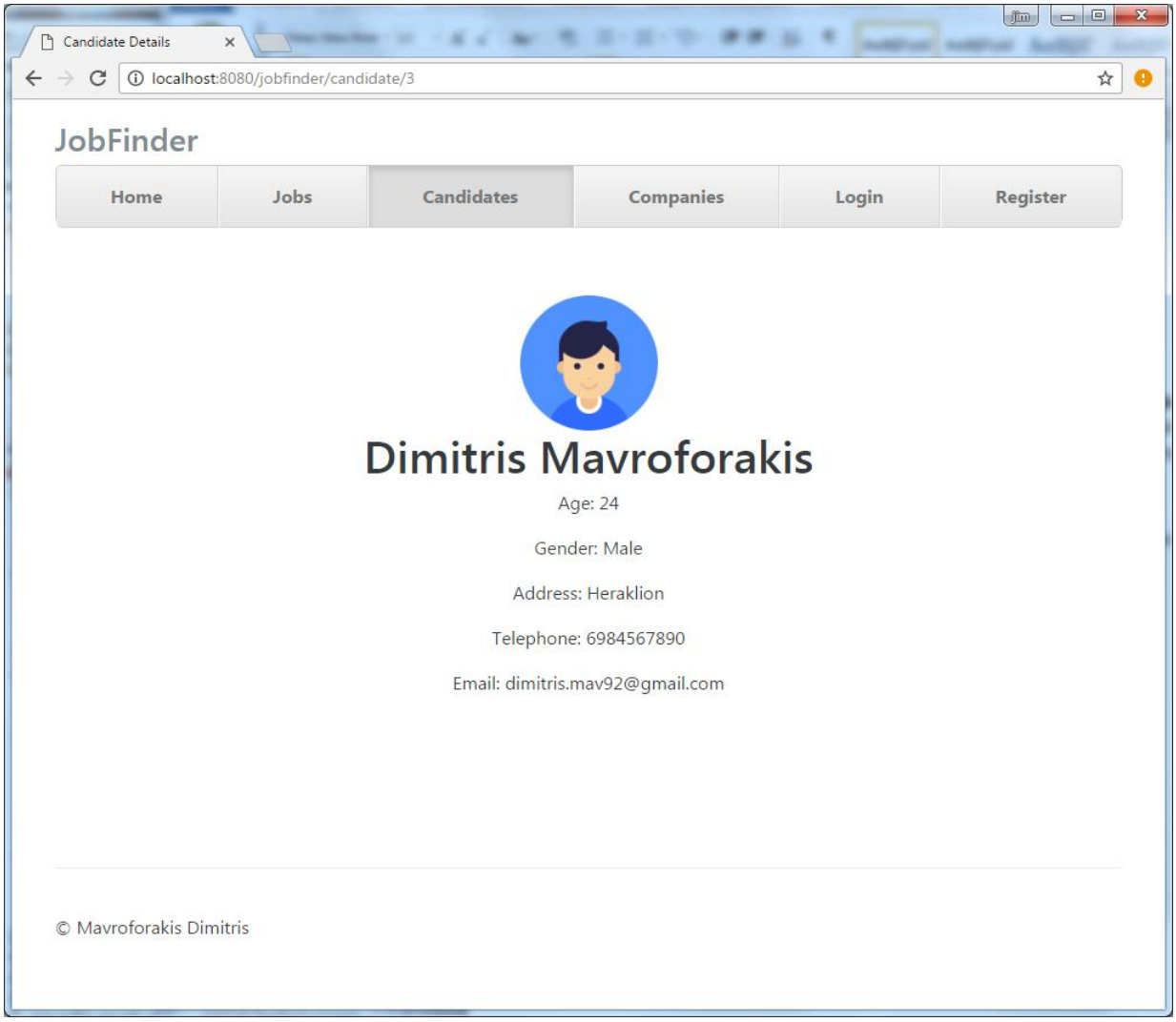
Προβολή υποψηφίου

Το url για την προβολή του προφίλ ενός υποψηφίου είναι της μορφής **/candidate/{id}**, όπου **id** είναι ο κωδικός του υποψηφίου όπως έχει αποθηκευθεί στην βάση δεδομένων. Η παρακάτω μέθοδος στην κλάση **CandidateController.java** διαχειρίζεται requests τέτοιου τύπου.

```
@RequestMapping(value = "/candidate/{id}")
public String showCandidate(@PathVariable("id") int id, Model model) {
	Candidate candidate = candidateService.find(id);
	model.addAttribute("candidate", candidate);
	return "candidate/show";
}
```

/candidate/show.jsp:

```
<html>
<head>
<%@ include file="../include/head.jsp"%>
<title>Candidate Details</title>
</head>
	<body>
	<div class="container">
		<%@ include file="../include/navbar.jsp"%>
			<div class="row">
				<div class="jumbotron">
					<img
src="${contextPath}/resources/images/candidate-icon.png" width="120" height="120"
/>
					<h1>${candidate.fname} ${candidate.lname}</h1>
					<p>Age: ${candidate.age}</p>
					<p>Gender: ${candidate.gender}</p>
					<p>Address: ${candidate.address}</p>
					<p>Telephone: ${candidate.telephone}</p>
					<p>Email: ${candidate.email}</p>
				</div>
			</div>
		<!-- footer -->
		<%@ include file="../include/footer.jsp"%>
	<script type="text/javascript">
		$("#candidates-link").addClass("active");
	</script>
	</body>
</html>
```



Εικόνα 15 – Προβολή υποψηφίου

4.4.9 Προβολή θέσεων εργασίας

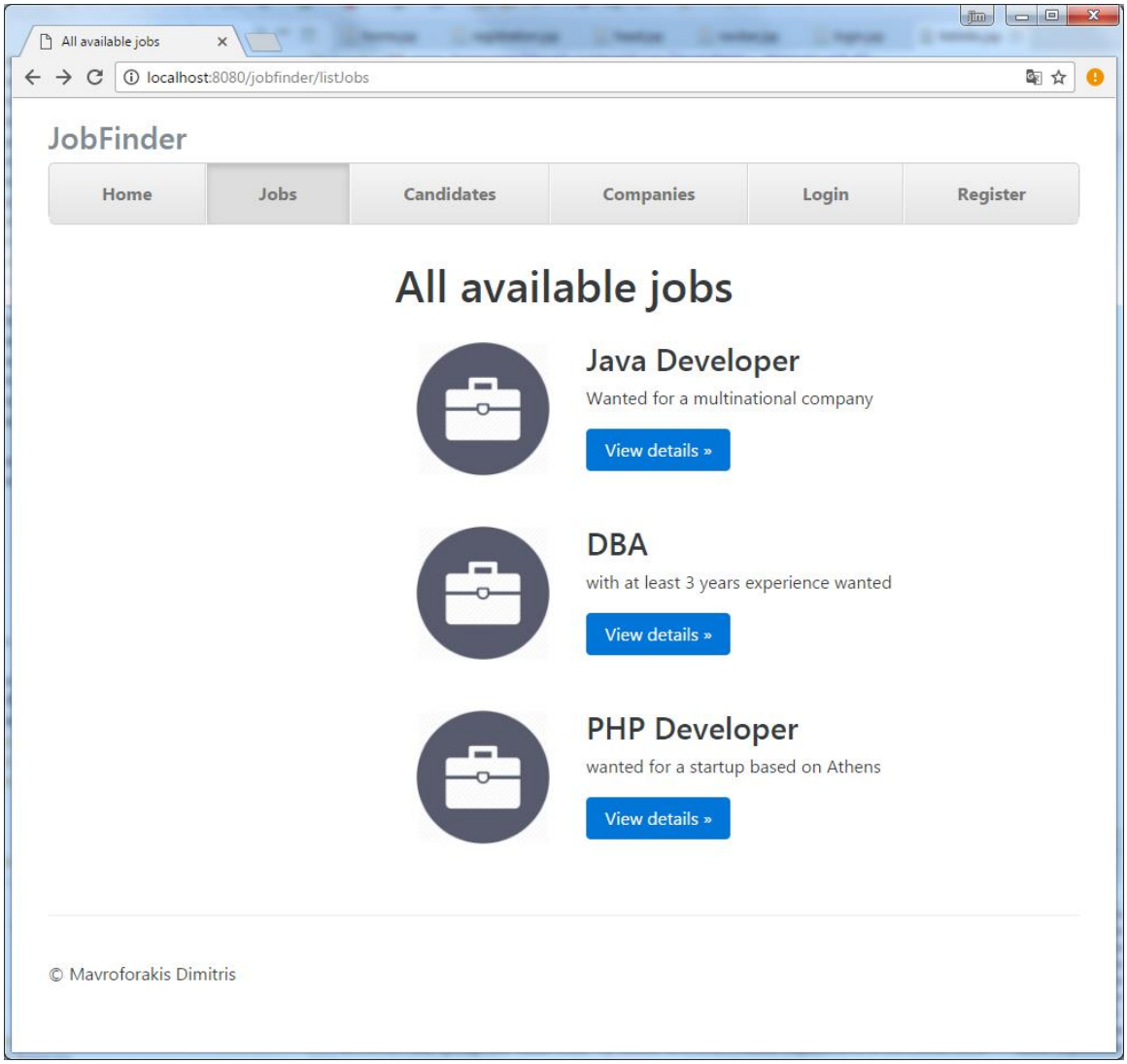
Η λίστα με τις διαθέσιμες θέσεις εργασίας προβάλλονται στον σύνδεσμο **/listJobs.** Η μέθοδος **listJobs** της κλάσης **JobController.java** ανακτά όλες τις θέσεις εργασίας μέσω του JobService καλώντας την μέθοδο getAll.

```
@RequestMapping(value = "/listJobs")
public String listJobs(Model model) {
	List<Job> jobs = jobService.getAll();
	model.addAttribute("jobList",jobs);
	return "/job/listJobs";
}
```

Η προβολή των θέσεων εργασίας γίνεται από το **listJobs.jsp**:

```
<html>
<head>
<title>All available jobs</title>
<%@ include file="../include/head.jsp"%>
</head>
<body>
	<div class="container">
		<%@ include file="../include/navbar.jsp"%>
		<div style="padding-top: 36px; text-align: center">
			<h1>All available jobs</h1>
		</div>
		<div class="row">
			<div class="col-lg-4"></div>
			<div class="col-lg-8">
				<c:forEach items="${jobList}" var="job"
				varStatus="loopCounter">
				<div class="job-container">
					<div class="job-image-container">
						<img
		src="${contextPath}/resources/images/job-icon.png"
							width="120"
							height="120" />
					</div>
					<div class="job-info-container">
						<h3>${job.title}</h3>
						<p>${job.description}</p>
						<p>
		<a class="btn btn-
		primary" href="${contextPath}/job/${job.id}"
		role="button">View
		details &raquo;</a>
						</p>
					</div>
				</div>
			</c:forEach>
		</div>
	</div>
```

```
<!-- footer -->
		<%@ include file="../include/footer.jsp"%>
		<!-- container -->
	</div>
	<script type="text/javascript">
		$("#listJobs-link").addClass("active");
	</script>
</body>

</html>
```



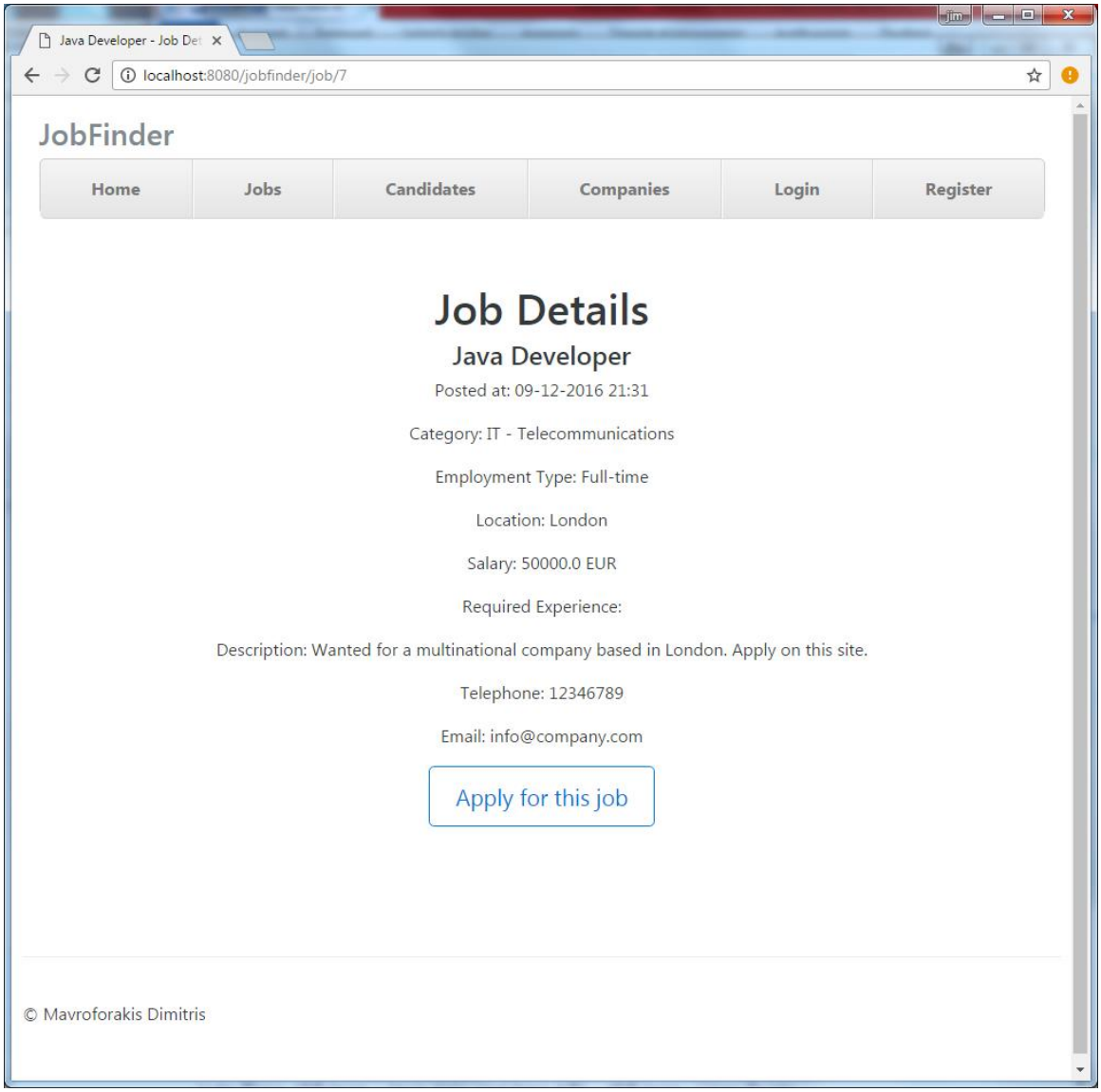
Εικόνα 16 – Προβολή θέσεων εργασίας

Προβολή λεπτομερειών θέσης εργασίας

```
@RequestMapping(value="/job/{jobId}")
public String showJob(@PathVariable("jobId") int jobId, Model model){
	Job job = jobService.find(jobId);
	model.addAttribute("job",job);
	return "job/showDetails";
}
```

showDetails.jsp:

```
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>

<html>
<head>
	<%@ include file="../include/head.jsp"%>
	<title>${job.title} - Job Details</title>
</head>
<body>
	<div class="container">
		<%@ include file="../include/navbar.jsp"%>
		<div class="row">
		<div class="jumbotron">
			<h1>Job Details</h1>
			<h4>${job.title}</h4>

			<p>Posted at: <fmt:formatDate value="${job.datePosted}"
			pattern="dd-M-Y H:m"/></p>

			<p>Category: ${job.category.name }</p>
			<p>Employment Type: ${job.employmentType }</p>
			<p>Location: ${job.location }</p>
			<p>Salary: ${job.salary} EUR</p>
			<p>Required Experience: ${job.experience }</p>
			<p>Description: ${job.description }</p>
			<p>Telephone: ${job.telephone }</p>
			<p>Email: ${job.email }</p>
			<p><a class="btn btn-lg btn-outline-primary"
href="${contextPath}/job/${job.id}/apply">Apply for this job</a>
			</p>
		</div>
			<%@ include file="../include/footer.jsp"%>
		</div>
	</div>
</body>
</html>
```



Εικόνα 17 – Λεπτομέρειες θέσης εργασίας

4.4.10 Αίτηση σε θέση εργασίας

Πατώντας το κουμπί **Apply**, ο χρήστης οδηγείται στην σελίδα όπου θα συμπληρώσει την φόρμα υποβολής βιογραφικού**.** Στο προηγούμενο παράδειγμα είδαμε την θέση με id 7, άρα ο σύνδεσμος για την υποβολή αίτησης θα είναι ο **/job/7/apply.** Δημιουργούμε την κλάση **JobApplicationController** η οποία θα είναι ο ελεγκτής που θα διαχειρίζεται τα αιτήματα που αφορούν αιτήσεις εργασίας. Η μέθοδος **prepareApply** καλείται για να προβάλει την σελίδα όπου ο χρηστής θα εισάγει τα στοιχεία του προτού αυτά καταχωρηθούν.

```
package com.jobfinder.controller;

@Controller
public class JobApplicationController {

	@Autowired
	private JobApplicationService jobApplicationService;

	@RequestMapping(value = "/job/{jobId}/apply", method = RequestMethod.GET)
	public String prepareApply(@PathVariable("jobId") int jobId, Model model)
	{

		Job job = jobService.find(jobId);
		JobApplication jobApp = new JobApplication();
		List<Language> languages = languageService.getAll();
		model.addAttribute("job", job);
		model.addAttribute("jobApplicationForm", jobApp);
		model.addAttribute("languages", languages);

		return "job/apply";
	}


}
```

Η κλάση **JobApplication.java** περιγράφει την αίτηση εργασίας και περιέχει όλα τα πεδία που απαιτούνται για την υποβολή μιας νέας αίτησης, όπως προσωπικά στοιχεία, εργασιακή εμπειρία, εκπαίδευση, γνώσεις ξένων γλωσσών. Χρησιμοποιούμε το `@Table(name = "job_applications")` ώστε το Hibernate να αντιστοιχεί αυτήν την κλάση με τον πίνακα job_applications της βάσης δεδομένων.

```
package com.jobfinder.model;

@Entity
@Table(name = "job_applications")
public class JobApplication {

	private int id;
	private String fname;
	private String lname;
	private String email;
	private String address;
	private String phone;
	private String coverLetter;
	private Date createdAt;

	private List<WorkingExperience> workingExperience;
	private List<Education> education;
	private List<JobApplicationLanguage> languages;


	/*
	 * Getters and setters
	 * */

}
```

Η κλάση **WorkExperience.java** περιγράφει την εργασιακή εμπειρία που μπορεί να έχει ένας υποψήφιος. Περιέχει πεδία όπως τίτλος, περιγραφή, εταιρεία, ημερομηνία έναρξης, ημερομηνία λήξης, βιομηχανία.

```
package com.jobfinder.model;

@Entity
@Table(name = "job_applications_working_experiences")
public class WorkingExperience {

	private int id;
	private String title;
	private String description;
	private String company;
	private Date startDate;
	private Date endDate;
	private String industry;

	private JobApplication jobApplication;

	/*
	 * Getters and setters
	 * */

}
```

Οι δύο οντότητες **JobApplication(Αίτηση Εργασίας)** και **WorkingExperience(Εργασιακή Εμπειρία)** έχουν μια σχέση ένα-προς-πολλά, καθώς σε μία αίτηση μπορούν να αναφέρονται πολλές εργασιακές εμπειρίες. Για να ορίσουμε αυτήν την σχέση μέσω του Hibernate θα χρησιμοποιήσουμε το OneToMany annotation στην κλάση JobApplication και το ManyToOne annotation στην κλάση WorkingExperience.

JobApplication.java:

```
@OneToMany(mappedBy = "jobApplication",
fetch=FetchType.EAGER,cascade = CascadeType.ALL)
public List<WorkingExperience> getWorkingExperience() {
	return workingExperience;
}
```

WorkExperience.java:

```
@ManyToOne
@JoinColumn(name = "job_application_id")
public JobApplication getJobApplication() {
	return jobApplication;
}
```

Στην οντότητα WorkingExperience, το πεδίο του πίνακα της ΒΔ όπου αποθηκεύεται ο κωδικός της αίτησης ονομάζεται application_id, γι αυτό χρησιμοποιήσαμε το `@JoinColumn(name = "application_id")`.

Με ανάλογο τρόπο, θα προσθέσουμε OneToMany annotations και για τις άλλες ιδιότητες της κλάσης **JobApplication.java**:

```
	@OneToMany(mappedBy = "jobApplication",fetch=FetchType.EAGER, cascade =
	CascadeType.ALL)
	public List<Education> getEducation() {
		return education;
	}

	@OneToMany(mappedBy = "jobApplication",fetch=FetchType.LAZY, cascade =
	CascadeType.ALL)
	public List<JobApplicationLanguage> getLanguages() {
		return languages;
	}
```

Η κλάση που περιγράφει την οντότητα εκπαίδευση είναι η **Education.java:**

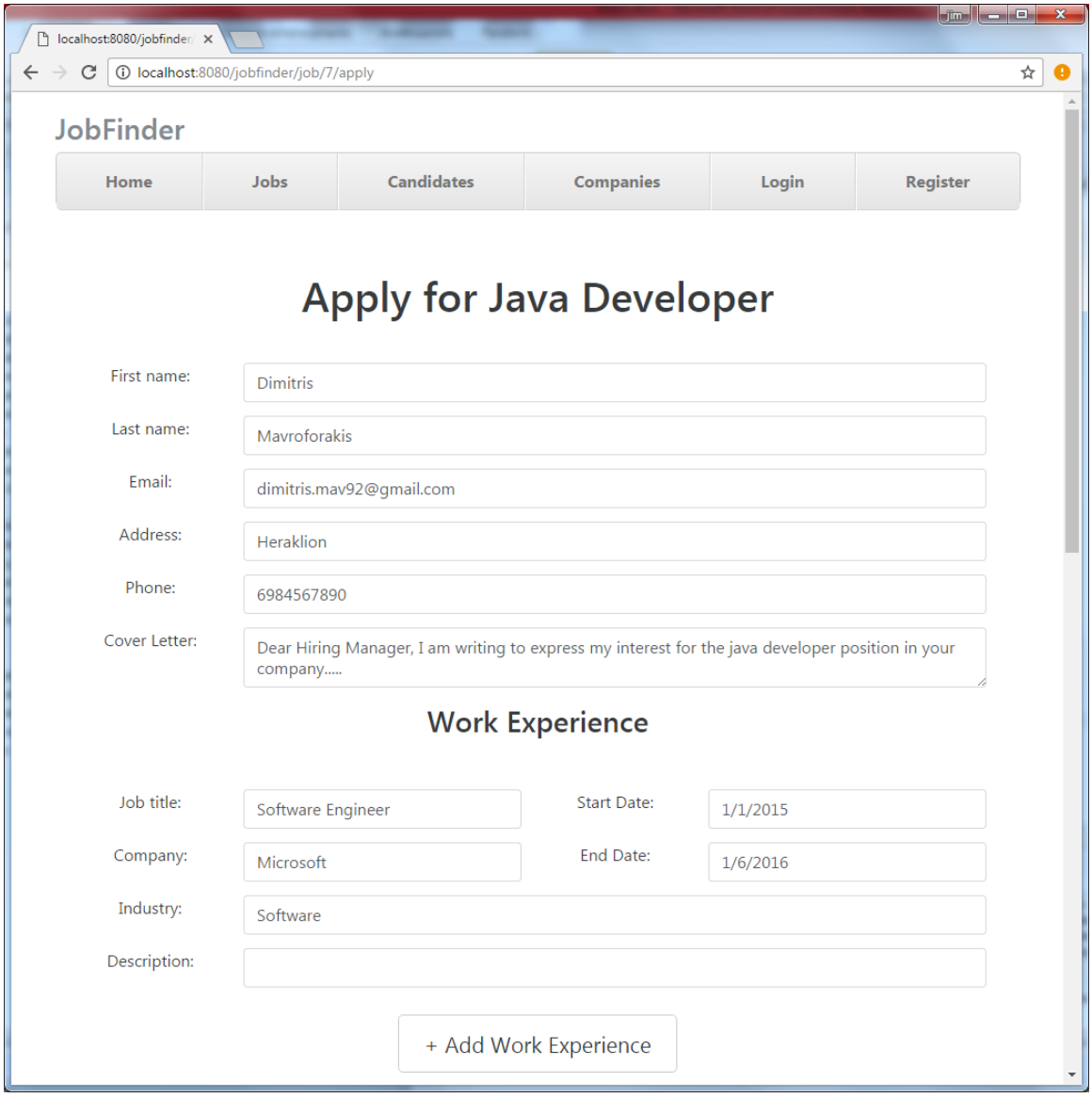
```
package com.jobfinder.model;

@Entity
@Table(name = "job_applications_education")
public class Education {

	private int id;
	private String degree;
	private String school;
	private String fieldOfStudy;
	private String description;
	private float grade;
	private Date startDate;
	private Date endDate;

	private JobApplication jobApplication;
	private EducationLevel educationLevel;

	/*
	 * Getters and setters
	 * */

	@ManyToOne
	@JoinColumn(name = "job_application_id")
	public JobApplication getJobApplication() {
		return jobApplication;
	}

	@ManyToOne
	@JoinColumn(name = "education_level_id")
	public EducationLevel getEducationLevel() {
		return educationLevel;
	}

}
```

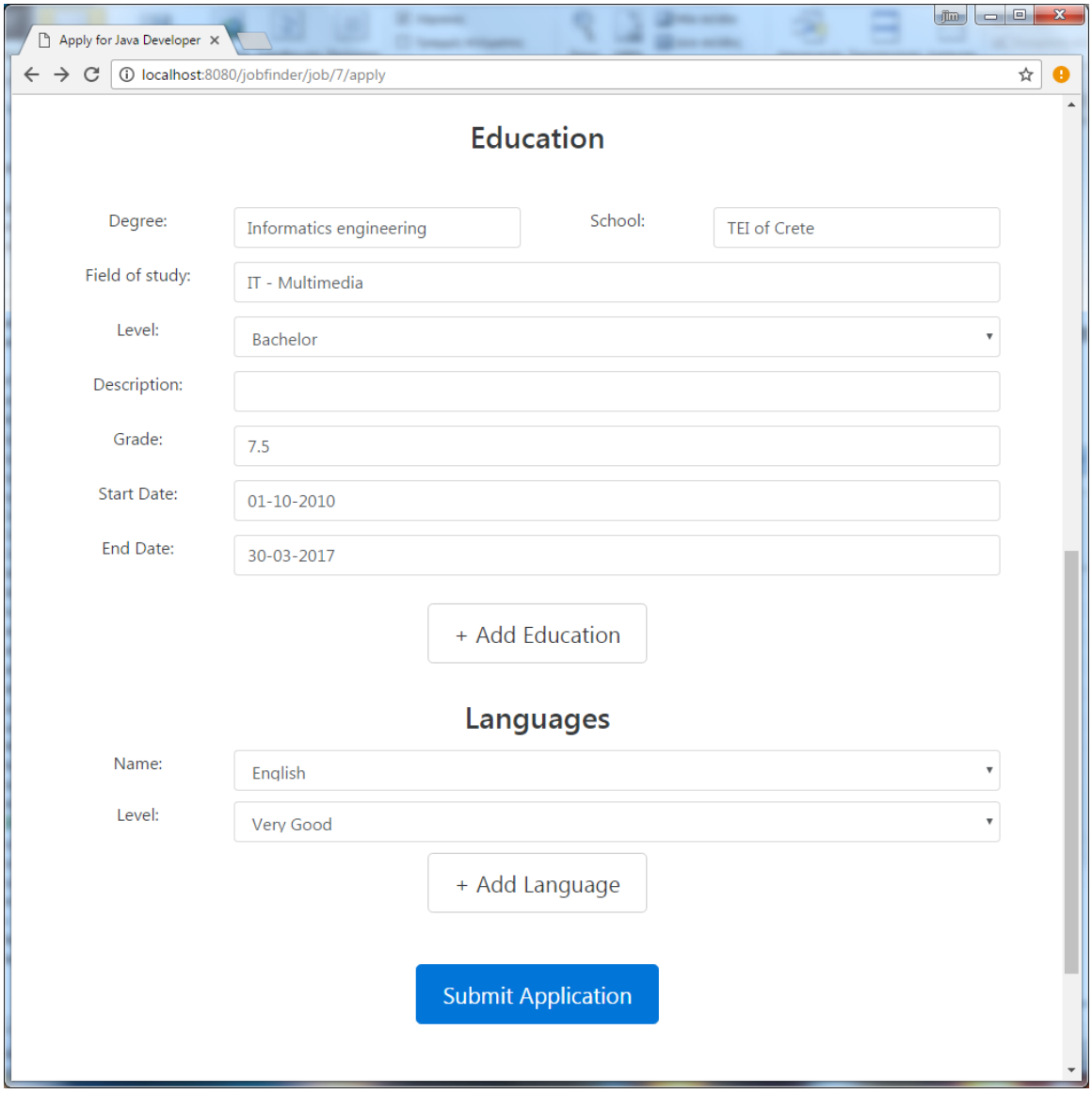
Ένας υποψήφιος μπορεί να γνωρίζει πολλές γλώσσες, άρα σε μια αίτηση μπορεί να αναφέρεται η γνώση πολλών γλωσσών. Η σχέση μεταξύ των οντοτήτων **JobApplication**(Αίτηση εργασίας) και **Language**(Γλώσσα) είναι πολλά προς πολλά, γι αυτό και στην βάση δεδομένων έχουμε τον πίνακα job_applications_languages όπου αποθηκεύονται τα πρωτεύοντα κλειδιά των δυο οντοτήτων καθώς και ο βαθμό γνώσης(level). Επειδή έχουμε μια σχέση πολλά-προς-πολλά με χαρακτηριστικό (attribute) δεν μπορούμε να χρησιμοποιήσουμε ManyToMany annotations και στις δύο κλάσεις. Αυτό θα γινόταν εάν στην βάση δεν χρειαζόταν να αποθηκεύσουμε τον βαθμό γνώσης (πεδίο level). Γι αυτόν τον λόγο θα πρέπει να «σπασουμε» την σχέση πολλα-προς-πολλά δημιουργώντας μια ενδιάμεση κλάση η οποία θα έχει ιδιότητες(properties) με σχέσεις ένα-προς-πολλά με τις δύο συσχετιζόμενες οντότητες.

```
package com.jobfinder.model;

@Entity
@Table(name="languages")
public class Language {

	private int id;
	private String name;
	private List<JobApplicationLanguage> jobApplicationLanguages;

	/*
	 * Getters and setters
	 * */

	@OneToMany(mappedBy = "language",fetch=FetchType.LAZY, cascade =
	CascadeType.ALL)
	public List<JobApplicationLanguage> getJobApplicationLanguages() {
		return jobApplicationLanguages;
	}

}
```

Η ενδιάμεση οντότητα που χρησιμοποιείται για να συσχετίσει τις οντότητες **JobApplication** και **Language**:

```
package com.jobfinder.model;

@Entity
@Table(name="job_applications_languages")
public class JobApplicationLanguage implements Serializable{

	private static final long serialVersionUID = -2027125152606566874L;

	private JobApplication jobApplication;
	private Language language;
	private int level;

	/*
	 * Getters and setters
	 * */

	@Id
	@ManyToOne(cascade = CascadeType.ALL)
	@JoinColumn(name="job_application_id")
	public JobApplication getJobApplication() {
		return jobApplication;
	}
	public void setJobApplication(JobApplication jobApplication) {
		this.jobApplication = jobApplication;
	}
	@Id
	@ManyToOne(cascade = CascadeType.ALL)
	@JoinColumn(name="language_id")
	public Language getLanguage() {
		return language;
	}

}
```


localhost:8080/jobfinder
localhost:8080/jobfinder/job/7/apply
JobFinder
Home
Jobs
Candidates
Companies
Login
Register
Apply for Java Developer
First name:
Dimitris
Last name:
Mavroforakis
Email:
dimitris.mav92@gmail.com
Address:
Heraklion
Phone:
6984567890
Cover Letter:
Dear Hiring Manager, I am writing to express my interest for the java developer position in your company.....
Work Experience
Job title:
Software Engineer
Start Date:
1/1/2015
Company:
Microsoft
End Date:
1/6/2016
Industry:
Software
Description:
+ Add Work Experience



Εικόνα 18 - Αίτηση σε θέση εργασίας

Όταν ο χρήστης πατήσει το κουμπί Submit application τότε θα κληθεί η μέθοδος **apply** του JobApplicationController.Στην φόρμα υποβολής η μέθοδος που χρησιμοποιείται είναι η POST γι αυτό έχουμε ορίσει ως παράμετρο του @RequestMapping το method = RequestMethod.POST. Εάν η μέθοδος ήταν GET, τότε θα είχε κληθεί η prepareApply, που όπως είδαμε έχει @RequestMapping(value="/job/{jobId}/apply", method=RequestMethod.GET).

Ένα από τα βασικά πλεονεκτήματα του Spring MVC είναι η χρήση του @ModelAttribute μέσω του οποίου το spring αναγνωρίζει και αντιστοιχεί αυτόματα τις παραμέτρους εισόδου σε ένα μοντέλο. Χωρίς την χρήση του θα έπρεπε να αντιστοιχούμε μία-μία τις παραμέτρους του request σε ιδιότητες του μοντέλου JobApplication. Για παράδειγμα, χωρίς το @ModelAttribute θα έπρεπε να πάρουμε τις τιμές που εστάλησαν μέσω της φόρμας με τον παρακάτω τρόπο:

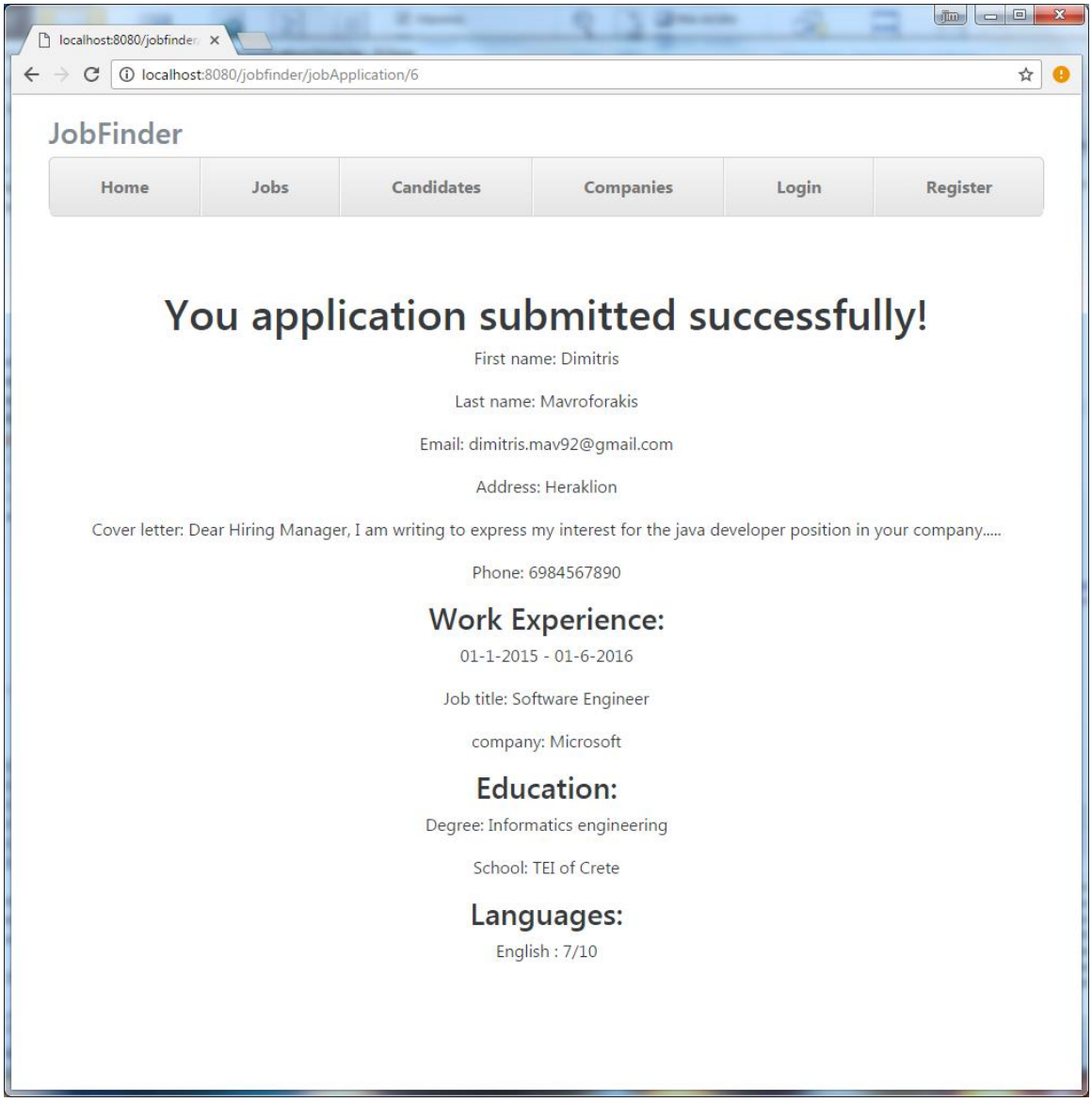
```
String fname = request.getParameter("fname");
String email = request.getParameter("email");
```

Ή

```
@PathVariable("fname") String fname;
@PathVariable("email") String email;


jobApp.setFname(fname);
jobApp.setEmail(email);
.
.
.
```

Χρησιμοποιώντας όμως το @ModelAttribute αυτή η διαδικασία γίνεται αυτόματα, όπως σε αυτήν την περίπτωση.

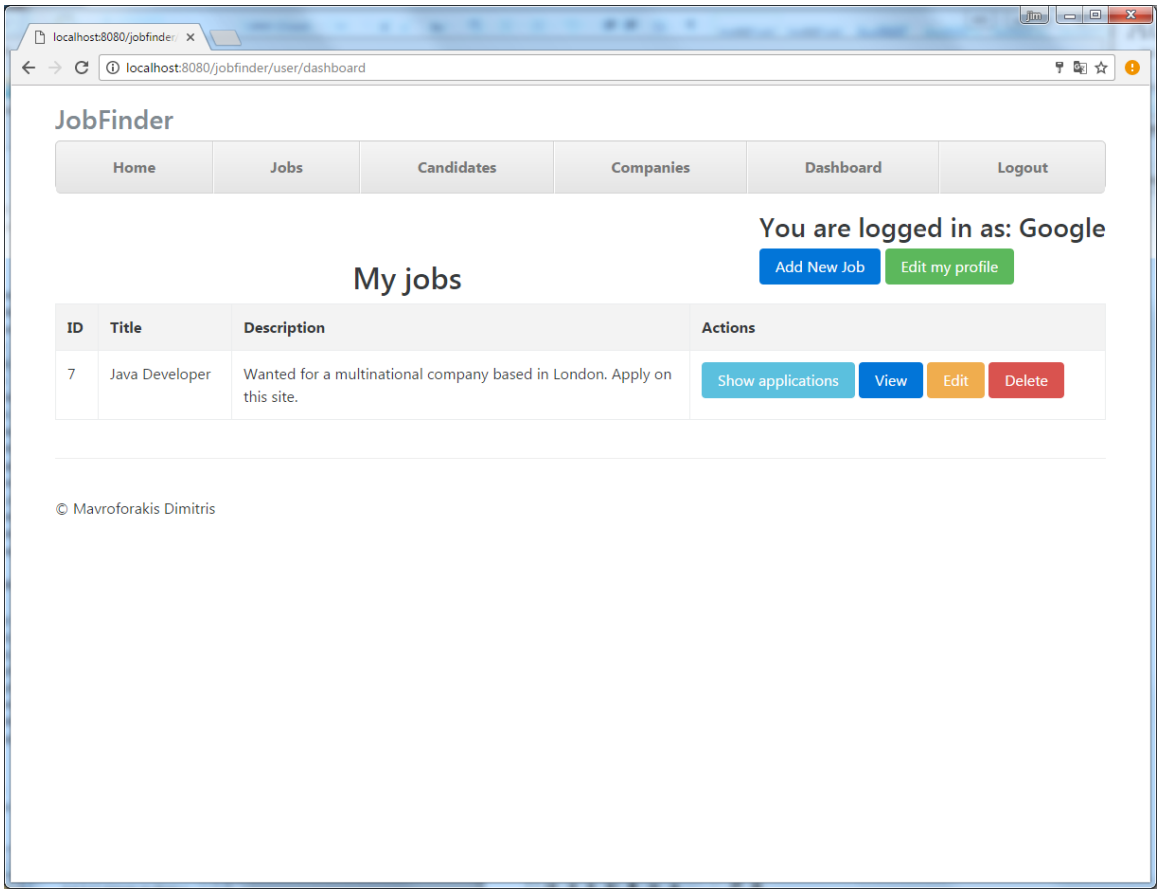
```
@RequestMapping(value = "/job/{jobId}/apply", method = RequestMethod.POST)
public String apply(@ModelAttribute("jobApplicationForm") JobApplication jobApp,
            @PathVariable("jobId") int jobId, BindingResult result, Model
            model,final RedirectAttributes redirectAttributes) {
      logger.debug("INSERTING NEW JOB APPLICATION ");

      jobApp.setJob(jobService.find(jobId));

      jobApp.setCreatedAt(new Date());

      //Submit job application
      jobApplicationService.addJobApplication(jobApp);
      String msg = "You application submitted successfully!";
      redirectAttributes.addFlashAttribute("msg",msg);
      return "redirect:/jobApplication/"+jobApp.getId();
}
```



Εικόνα 19 - Επιτυχής καταχώρηση αίτησης

4.4.11 Προβολή αιτήσεων

Εφόσον ο χρήστης έχει συνδεθεί στο λογαριασμό του, θα μπορεί να δει τις αιτήσεις που έχουν σταλεί για τις θέσεις εργασίας που έχει δημοσιεύσει. Στο πάνελ του χρήστη(dashboard) εμφανίζονται όλες οι θέσεις που έχει δημοσιεύσει ο χρήστης.



Εικόνα 20 – Πανελ χρήστη

Πατώντας το κουμπί Show applications ο χρήστης κατευθύνεται στην σελίδα όπου εμφανίζονται οι αιτήσεις που έχουν γίνει. Τα urls για την προβολή των αιτήσεων είναι της μορφής **/job/{jobId}/applications**. Δημιουργούμε την μέθοδο **showJobApplications()** στην κλάση JobApplicationController ώστε να διαχειρίζεται αυτά τα urls.

```
@RequestMapping(value = "/job/{jobId}/applications", method =
RequestMethod.GET)
public String showJobApplications(@PathVariable("jobId") int jobId, Model
model) {
	Job job = jobService.find(jobId);
	List<JobApplication> jobApplications =
	jobApplicationService.getApplicationsByJobId(jobId);

	model.addAttribute("job", job);
	model.addAttribute("criteria", new EvaluationCriteria());
	model.addAttribute("jobApplications", jobApplications);
	model.addAttribute("availableLanguages", languageService.getAll());
	return "job/applications";
}
```

Η μέθοδος **getApplicationsByJobId(jobId)** του JobApplicationService επιστρέφει τις αιτήσεις που έχουν γίνει για μια συγκεκριμένη θέση

```
@Service
public class JobApplicationServiceImpl extends GenericServiceImpl<JobApplication,
Integer> implements JobApplicationService {

	@Transactional
	public List<JobApplication> getApplicationsByJobId(int jobId) {
		return jobApplicationDAO.getApplicationsByJobId(jobId);
	}

}
```

Η ανάκτηση των αντικειμένων γίνεται χρησιμοποιώντας HQL (Hibernate Query Language) δηλαδή την γλώσσα του Hibernate και όχι με την κλασσική SQL. Η HQL λειτουργεί με αντικείμενα και τις ιδιότητες τους, και όχι απευθείας με τους πίνακες της ΒΔ. Το αντικείμενο JobApplication περιέχει το property job, το οποίο είναι αντικείμενο τύπου Job, και αφορά την θέση εργασίας στην οποία έχει γίνει η αίτηση. Οπότε θα ψάξουμε στο πεδίο id του property job, του αντικειμένου JobApplication.

JobApplicationDAOImpl.java:

```
@Repository
public class JobApplicationDAOImpl extends GenericDAOImpl<JobApplication,Integer>
implements JobApplicationDAO {

	@SuppressWarnings("unchecked")
	public List<JobApplication> getApplicationsByJobId(int jobId) {
		String hql = "from JobApplication WHERE job.id = :jobId";
		List<JobApplication> jobApplications =
		getSession().createQuery(hql).setParameter("jobId", jobId).list();

		return jobApplications;
		}

}
```

Δημιουργούμε την κλάση **EvaluationCriteria.java** η οποία περιγράφει τα κριτήρια αξιολόγησης. Τα δεδομένα που εισάγει ο χρήστης είναι της μορφής:

Εργασιακή εμπειρία: Ελάχιστη - 2 χρόνια – 50% Βαρύτητα
Εκπαίδευση: Τουλάχιστον – Τριτοβάθμια – 30 % Βαρύτητα
Γνώση Αγγλικών: Εως – Αριστα – 10% Βαρύτητα
Γνώση Γερμανικών: Εως – Πολύ καλά – 10% Βαρύτητα

Άρα οι ιδιότητες της κλάσης θα είναι: Τελεστής εργασιακής εμπειρίας(workExperienceOperator), απαιτούμενη εργασιακή εμπειρία (workExperienceRequired), βαρύτητα εργασιακής εμπειρίας (workExperienceWeight).

Για την εκπαίδευση επίσης έχουμε: τελεστή εκπαίδευσης(educationOperator), επίπεδο εκπαίδευσης(educationLevel), βαρύτητα εκπαίδευσης(educationWeight).

Για τις γλώσσες επειδή δεν είναι σταθερός ο αριθμός τους, θα δημιουργήσουμε νέα κλάση για τα κριτήρια αξιολόγησης τους. Μέσω του πεδίου minTotalScore ο χρήστης μπορεί να δώσει το ελάχιστο σκορ που απαιτείται από μια αίτηση για να θεωρηθεί ότι πληροί τα κριτήρια(qualified).

```
public class EvaluationCriteria {

	private String workExperienceOperator;
	private String workExperienceRequired;
	private float  workExperienceWeight = 0;

	private String educationOperator;
	private int educationLevel;
	private float educationWeight = 0;

	private List<LanguageEvaluationCriteria> languages;

	private float minTotalScore;

	/*
	 * Getters and setters
	 * */

}
```

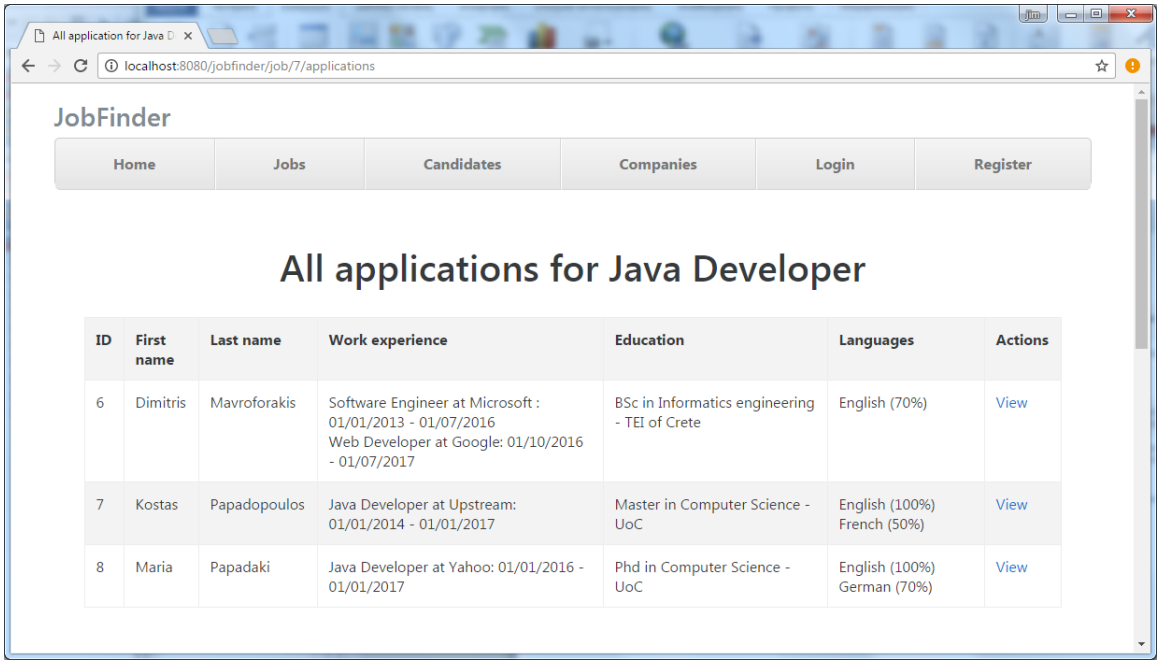
Η αξιολόγηση γνώσης μιας γλώσσας γίνεται δίνοντας τον κωδικό γλώσσας(languageId), τελεστή(operator), επίπεδο(level), βαρύτητα(weight).

```
public class LanguageEvaluationCriteria {

	private int languageId;
	private String operator;
	private int level;
	private float weight = 0;
	/*
	 * Getters and setters
	 * */

}
```

Jsp σελίδα για την προβολή των αιτήσεων:

Applications.jsp:

```
<h1>All applications for ${job.title }</h1>
<br>
<table class="table table-bordered table-striped">
	<tr>
		<th>ID</th>
		<th>First name</th>
		<th>Last name</th>
		<th>Work experience</th>
		<th>Education</th>
		<th>Languages</th>
		<th>Actions</th>
	</tr>

<c:forEach items="${jobApplications}" var="jobApp" varStatus="loopCounter">

		<tr>

		<td>${jobApp.id }</td>
		<td>${jobApp.fname}</td>
		<td>${jobApp.lname}</td>
		<td>

		<c:forEach items="${jobApp.workingExperience}" var="workExp"
		varStatus="workExpCounter">
		${workExp.title} at ${workExp.company}:
		<fmt:formatDate value="${workExp.startDate}" pattern="dd/MM/yyyy" /> -
		<fmt:formatDate value="${workExp.endDate}" pattern="dd/MM/yyyy" /><br>
		</c:forEach></td>

		<td><c:forEach items="${jobApp.education}" var="education">
		${education.educationLevel.title} in ${education.degree} -
		${education.school}
		</c:forEach></td>

		<td><c:forEach items="${jobApp.languages}" var="language">
		${language.language.name } (${language.level*10}%)
		</c:forEach></td>

		<td><a href="${contextPath}/jobApplication/${jobApp.id}">View</a></td>
		</tr>
</c:forEach>
</table>
```



Εικόνα 21 – Προβολή αιτήσεων

4.4.12 Αξιολόγηση αιτήσεων

Όταν ο χρήστης εισάγει τιμές στα κριτήρια αξιολόγησης και καταχωρήσει την φόρμα, θα κληθεί η μέθοδος **evaluateApplications()** του JobApplicationController.Αρχικά θα γίνει η ανάκτηση της θέσης εργασίας μέσω της μεθόδου find() του JobService και στην συνέχεια η ανάκτηση των αιτήσεων μέσω του JobApplicationService και της μεθόδου getApplicationsByJobId().

```
@RequestMapping(value = "/job/{jobId}/applications", method = RequestMethod.POST)
public String evaluateApplications(@PathVariable("jobId") int jobId,
@ModelAttribute("criteria") EvaluationCriteria criteria, Model model) {
	Job job = jobService.find(jobId);

	List<JobApplication> jobApplications =
	jobApplicationService.getApplicationsByJobId(jobId);


	jobApplications =
	jobApplicationService.evaluateJobApplications(jobApplications, criteria);

	model.addAttribute("job", job);
	model.addAttribute("jobApplications", jobApplications);
	model.addAttribute("availableLanguages", languageService.getAll());
	return "job/applications";
}
```

Η αξιολόγηση των αιτήσεων γίνεται καλώντας την μέθοδο **evaluateJobApplications** της JobApplicationService. Δέχεται ως ορίσματα τις αιτήσεις και τα κριτήρια αξιολόγησης και επιστρέφει τις αιτήσεις έχοντας αξιολογηθεί με τα κριτήρια εισόδου.

Για κάθε αίτηση, υπολογίζεται η συνολική εργασιακή εμπειρία σε χρόνια. Ένα π.χ κάποιος υποψήφιος έχει εργασθεί από 1/1/2015 έως 1/7/2016 στην εταιρεία Α και από 1/10/2016 έως 1/7/2017 στην εταιρεία Β, τότε η συνολική εργασιακή εμπειρία του είναι 2,25 χρόνια. Το σύνολο της εργασιακής εμπειρίας αποθηκεύεται στην μεταβλητή totalWorkExp.

Στην συνέχεια αξιολογείται σε τι ποσοστό το σύνολο της προϋπηρεσίας καλύπτει το κριτήριο της εργασιακής εμπειρίας. Εάν ο τελεστής είναι **exacly** τότε για να πληρείται κατά 100% το κριτήριο, θα πρέπει το σύνολο της εργασιακής εμπειρίας του υποψηφίου να είναι μεγαλύτερο ή ίσο του απαιτούμενου. Εάν ο τελεστής είναι **min**, τότε σε περίπτωση που το σύνολο καλύπτει το απαιτούμενο το score είναι 100%. Η έξτρα εμπειρία βαθμολογείται θετικά και αποθηκεύεται στην μεταβλητή workExperienceBonusScore. Εάν το τελεστής είναι **max** τότε διαιρείται η συνολική εμπειρία με την απαιτούμενη και πολλαπλασιάζεται επι 100.

Παραδείγματα:

Αιτήσεις (JobApplications)
Υποψήφιος Α: Συνολική εργασιακή εμπειρία (σε χρόνια): 2,25

Υποψήφιος Β: Συνολική εργασιακή εμπειρία (σε χρόνια): 3

Υποψήφιος Γ: Συνολική εργασιακή εμπειρία (σε χρόνια): 5

1) Κριτήριο αξιολόγησης: Προϋπηρεσία τουλάχιστον 3 χρόνια

Τιμές μεταβλητών
workExperienceOperator: **min**
workExperienceRequired: **3**

Αξιολόγηση:

Υποψήφιος Α: workExperienceScore: **0%**

Υποψήφιος Β: workExperienceScore: **100%**, workExperienceBonusScore: **0**

Υποψήφιος Γ: workExperienceScore: **100%**, workExperienceBonusScore: **66%**

Σε αυτήν την περίπτωση εάν ο υποψήφιος δεν έχει την ελάχιστη απαιτούμενη προϋπηρεσία τότε το σκορ του είναι 0. Εάν έχει ακριβώς την απαιτούμενη εμπειρία τότε το σκορ είναι 100%. Σε περίπτωση που έχει μεγαλύτερη από την απαιτούμενη εμπειρία, το σκορ είναι 100% και η έξτρα εμπειρία μετράει σαν bonus. Εάν π.χ έχει διπλάσια εμπειρία από την απαιτούμενη το bonus score είναι 100%.

2) Κριτήριο αξιολόγησης Προϋπηρεσία έως 4 χρόνια

Τιμές μεταβλητών
workExperienceOperator: **max**
workExperienceRequired: **4**

Υποψήφιος Α: workExperienceScore: 2,25/4*100 = **56,25%**

Υποψήφιος Β: workExperienceScore: 3/4*100 = **75%**, workExperienceBonusScore: 0

Υποψήφιος Γ: workExperienceScore: **100%**, workExperienceBonusScore: 0

Σε αυτή την περίπτωση, εάν ο υποψήφιος υπερκαλύπτει την απαιτούμενη προϋπηρεσία δεν λαμβάνει bonus score.

3) Κριτήριο αξιολόγησης Προϋπηρεσία ακριβώς 3 χρόνια

Τιμές μεταβλητών
workExperienceOperator: **exacly**
workExperienceRequired: **3**

Υποψήφιος Α: workExperienceScore: **0%**

Υποψήφιος Β: workExperienceScore: **100%**, workExperienceBonusScore: **0**

Υποψήφιος Γ: workExperienceScore: **100%,** workExperienceBonusScore: **0**

Η μέθοδος του **JobApplicationService** που αξιολογεί τις αιτήσεις είναι η **evaluateJobApplications:**

```
@Transactional
public List<JobApplication> evaluateJobApplications(List<JobApplication>
jobApplications, EvaluationCriteria criteria) {

	for (JobApplication jobApp : jobApplications) {
		float workExperienceBonusScore = 0;
		float educationScoreBonus = 0;
		float totalWorkExp = 0;

		for (WorkingExperience workExp : jobApp.getWorkingExperience()) {

			Calendar startCalendar = new GregorianCalendar();
			startCalendar.setTime(workExp.getStartDate());
			Calendar endCalendar = new GregorianCalendar();
			endCalendar.setTime(workExp.getEndDate());

			int diffYear = endCalendar.get(Calendar.YEAR) -
			startCalendar.get(Calendar.YEAR);
			int diffMonth = diffYear * 12 +
			endCalendar.get(Calendar.MONTH) -
			startCalendar.get(Calendar.MONTH);


		       float workExpInYears = (float) diffMonth / 12;
		       totalWorkExp += workExpInYears;

		}
```

```
if (!criteria.getWorkExperienceRequired().isEmpty()) {
switch(criteria.getWorkExperienceOperator()){
	case "exacly":
	if (totalWorkExp >= Float.parseFloat(criteria.getWorkExperienceRequired())) {
		jobApp.setWorkingExperienceScore(100);
	}
	break;
	case "min":
	if (totalWorkExp < Float.parseFloat(criteria.getWorkExperienceRequired())) {
		jobApp.setWorkingExperienceScore(0);
	} else {
		jobApp.setWorkingExperienceScore(100);
		workExperienceBonusScore += totalWorkExp/
		Float.parseFloat(criteria.getWorkExperienceRequired()) * 100 - 100;
	}
	break;
	case "max":
	if (totalWorkExp >= Float.parseFloat(criteria.getWorkExperienceRequired())) {
		jobApp.setWorkingExperienceScore(100);
	} else {
		jobApp.setWorkingExperienceScore(totalWorkExp /
	Float.parseFloat(criteria.getWorkExperienceRequired()) * (float) 100);
	}
	break;
```

Αξιολόγηση εκπαίδευσης

Αρχικά δημιουργούμε την κλάση EducationLevel η οποία περιγράφει την βαθμίδα εκπαίδευσης. Τα πεδία της κλάσης είναι κωδικός(id), τίτλος (title) και κατάταξη(ranking).

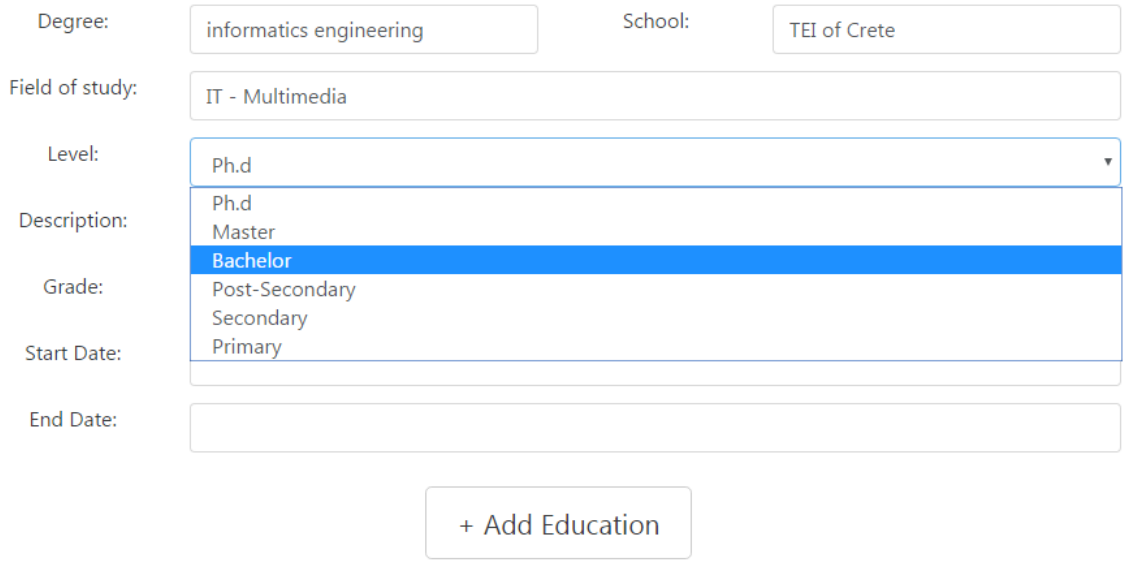
```
package com.jobfinder.model;

@Entity
@Table(name="education_levels")
public class EducationLevel {

	private int id;
	private String title;
	private int ranking;

	/*
	 * Getters and setters
	 * */

}
```

Οι βαθμίδες της εκπαίδευσης αποθηκεύονται στον πίκακα education_levels της βάσης δεδομένων. Για να εισάγουμε στην Β.Δ όλες τις πιθανές βαθμίδες, εκτελούμε τα παρακάτω queries:

```
INSERT INTO education_levels VALUES (1,'Phd',10);
INSERT INTO education_levels VALUES (2,'Master',9);
INSERT INTO education_levels VALUES (3,'BSc',8);
INSERT INTO education_levels VALUES (4,'Post-Secondary',5);
INSERT INTO education_levels VALUES (5,'Secondary',4);
INSERT INTO education_levels VALUES (6,'Primary',2);
```

Η ανώτατη βαθμίδα θεωρείται το διδακτορικό (Phd) με κατάταξη 10, στην συνέχεια το μεταπτυχιακό(Master) με κατάταξη 9, προπτυχιακό (BSc) με 8 κτλ.

Κατά την καταχώρηση της αίτησης, ο χρήστης εισάγει τον τίτλο σπουδών του καθώς και την βαθμίδα στην οποία ανήκει ο τίτλος σπουδών που κατέχει.



Γνωρίζοντας λοιπόν το επίπεδο της εκπαίδευσης του κάθε υποψηφίου μπορούμε να συγκρίνουμε και να υπολογίσουμε σε τι ποσοστό πληροί τα απαιτούμενα κριτήρια. Παρόμοια με την προϋπηρεσία, σε κάθε αίτηση υπάρχουν οι μεταβλητές educationScore και educationBonusScore στις οποίες αποθηκεύονται το σκορ και το bonus που συγκεντρώνει η κάθε αίτηση μετά την αξιολόγηση της.

Η αξιολόγηση της εκπαίδευσης γίνεται με βάση το παρακάτω παράδειγμα:

Υποψήφιος Α: Επίπεδο εκπαίδευσης διδακτορικό (Phd - Ranking 10)

Υποψήφιος Β: Επίπεδο εκπαίδευσης μεταπτυχιακό (Master - Ranking 9)

Υποψήφιος Γ: Επίπεδο εκπαίδευσης προπτυχιακό (BSc - Ranking 8)

1) Κριτήριο 1: τουλάχιστον προπτυχιακό

Τιμές μεταβλητών:
educationOperator: **min**
educationLevel: 8

Αξιολόγηση:

Υποψήφιος Α: educationScore: **100%**
Υποψήφιος Β: educationScore: **100%**, educationBonusScore: **100**
Υποψήφιος Γ: educationScore: **100%**, workExperienceBonusScore: **200**
Όλοι όσοι έχουν πτυχίο μεγαλύτερο ή ίσο του προπτυχιακού πληρούν το κριτήριο κατά 100%. Σε περίπτωση που κάποιος έχει πτυχίο ανώτερο του προπτυχιακού λαμβάνει bonus. Εάν κάποιος έχει εκπαίδευση με ranking μικρότερη του 8, τότε το educationScore είναι 0.

2) Κριτήριο 2: έως Διδακτορικό

Τιμές μεταβλητών:
educationOperator: **max**
educationLevel: 10

Αξιολόγηση:

Υποψήφιος Α: educationScore: **8/10 = 80%**
Υποψήφιος Β: workExperienceScore: 9/10 = **90%**, workExperienceBonusScore: **0**
Υποψήφιος Γ: workExperienceScore: 10/10 = **100%**, workExperienceBonusScore: **0**

3) Κριτήριο 3: ακριβώς μεταπτυχιακό

Τιμές μεταβλητών:
educationOperator: **exacly**
educationLevel: 9

Αξιολόγηση:

Υποψήφιος Α: educationScore: **0%**
Υποψήφιος Β: workExperienceScore: **100%**, workExperienceBonusScore: **0**
Υποψήφιος Γ: workExperienceScore: **100%**, workExperienceBonusScore: **0**

Σε αυτήν την περίπτωση εάν το επίπεδο εκπαίδευσης του υποψηφίου είναι χαμηλότερο από το απαιτούμενο τότε το score είναι 0%. Επίσης εάν είναι μεγαλύτερο τότε δεν βαθμολογείται με bonus.

Υλοποίηση:

```
for (Education education : jobApp.getEducation()) {
if (criteria.getEducationOperator().equals("exacly")) {
	if (education.getEducationLevel().getRanking() >= criteria.getEducationLevel()) {
		jobApp.setEducationScore(100);
	break;
	}
} else if (criteria.getEducationOperator().equals("min")) {
	if (education.getEducationLevel().getRanking() >= criteria.getEducationLevel()) {
		jobApp.setEducationScore(100);
		educationScoreBonus = (education.getEducationLevel().getRanking()
					- criteria.getEducationLevel()) * 100;
		break;
	}
} else if (criteria.getEducationOperator().equals("max")) {
	if (education.getEducationLevel().getRanking() >= criteria.getEducationLevel()) {
	jobApp.setEducationScore(100);
	break;
	} else {
	jobApp.setEducationScore(((float)education.getEducationLevel().getRanking()
				/ (float) criteria.getEducationLevel()) * 100);
	break;
	}
}

}
jobApp.setEducationBonusScore(educationScoreBonus);
```

Αξιολόγηση ξένων γλωσσών

Παρόμοια λογική ακολουθούμε και για την αξιολόγηση των γλωσσών που γνωρίζει ο υποψήφιος.

```
for (JobApplicationLanguage candidatelanguage : jobApp.getLanguages()) {
	for (LanguageEvaluationCriteria languageCriterio : criteria.getLanguages()) {
	if (candidatelanguage.getLanguage().getId() == languageCriterio.getLanguageId()) {
		if (languageCriterio.getOperator().equals("exacly")) {
			if (candidatelanguage.getLevel() >= languageCriterio.getLevel()) {
				candidatelanguage.setScore(100);
			}
		} else if (languageCriterio.getOperator().equals("min")) {
			if (candidatelanguage.getLevel() < languageCriterio.getLevel()) {
				candidatelanguage.setScore(0);
			} else {
				candidatelanguage.setScore(100);
			}
		} else if (languageCriterio.getOperator().equals("max")) {
			if (candidatelanguage.getLevel() >= languageCriterio.getLevel()) {
				candidatelanguage.setScore(100);
			} else {
				float languageScore = (float) (candidatelanguage.getLevel())
							/ languageCriterio.getLevel() * 100;

				candidatelanguage.setScore(languageScore);
			}
		}
	}
	}
}
```

Για τον υπολογισμό του τελικού σκορ της κάθε αίτησης, πολλαπλασιάζουμε το σκορ της Εργασιακής εμπειρίας, εκπαίδευσης και ξένων γλωσσών με τον αντίστοιχο συντελεστή.

```
float totalScore = 0;

if (criteria.getWorkExperienceWeight() > 0) {
	totalScore +=
	jobApp.getWorkingExperienceScore()*criteria.getWorkExperienceWeight();
}
if (criteria.getEducationWeight() > 0) {
	totalScore += jobApp.getEducationScore() * criteria.getEducationWeight();
}

for (LanguageEvaluationCriteria languageCriterio : criteria.getLanguages()) {
	if (languageCriterio.getWeight() > 0) {
		float languageScore = 0;
	for (JobApplicationLanguage lang : jobApp.getLanguages()) {
		if (lang.getLanguage().getId() == languageCriterio.getLanguageId()) {
			languageScore = lang.getScore();
		}
	}
	totalScore += languageScore * languageCriterio.getWeight();
	}
}
jobApp.setTotalScore(totalScore);

if (totalScore >= criteria.getMinTotalScore()) {
	jobApp.setQualified("Yes");
} else {
	jobApp.setQualified("No");
}
```

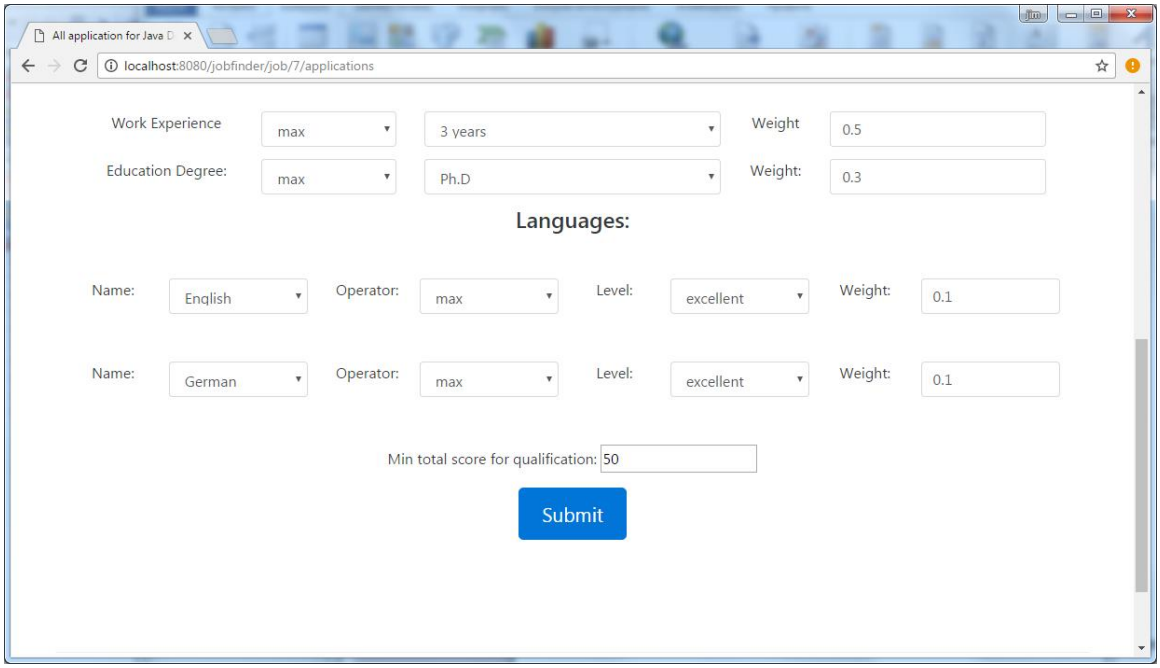
Στην συνέχεια κατατάσσουμε τις αιτήσεις ανάλογα το συνολικό σκορ με τον αλγόριθμο bubble sort.

```
  for (int i = 0; i < jobApplications.size() - 1; i++) {
		for (int j = 1; j < jobApplications.size() - i; j++) {
			if (jobApplications.get(j - 1).getTotalScore() <
			jobApplications.get(j).getTotalScore()) {
				JobApplication temp = jobApplications.get(j - 1);
				jobApplications.set(j - 1, jobApplications.get(j));
				jobApplications.set(j, temp);
			}

		}
  }
```

Παράδειγμα 1, δίνουμε ως κριτήρια αξιολόγησης:

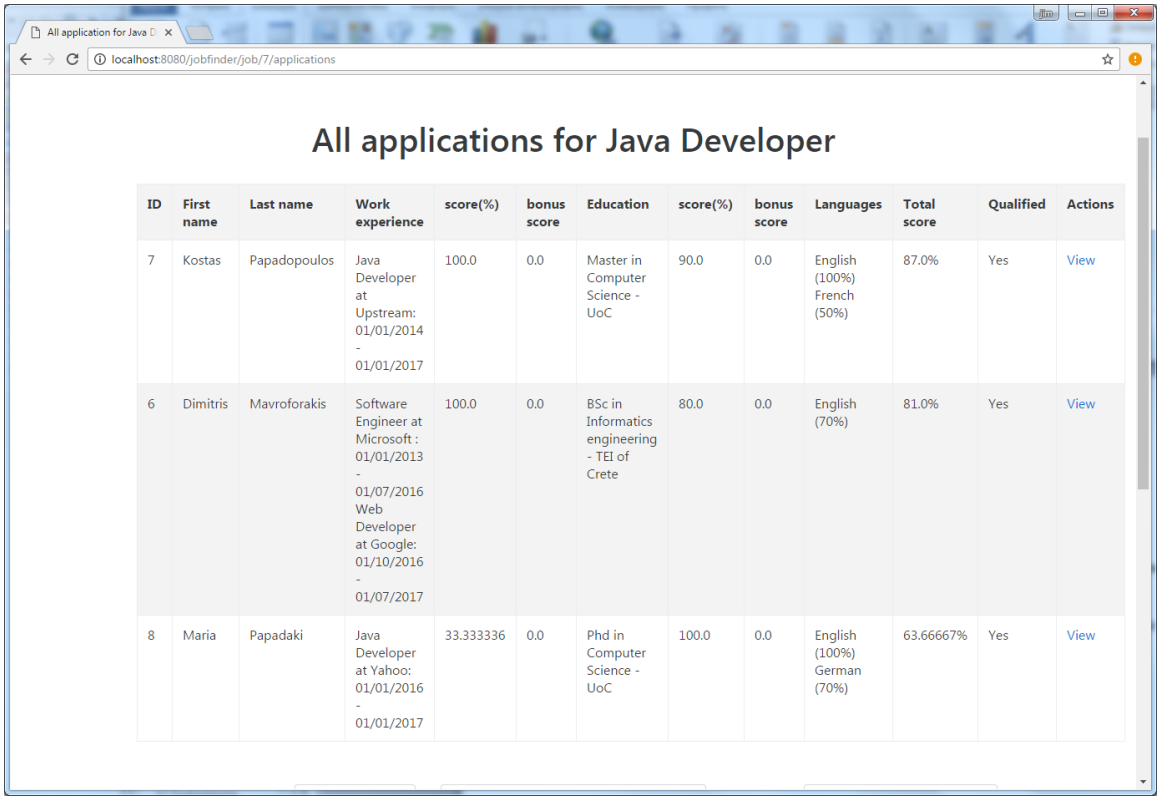
Προϋπηρεσία έως 3 χρόνια, 50% βαρύτητα
Εκπαίδευση έως διδακτορικό, 30% βαρύτητα
Αγγλικά έως άριστα(10%), γερμανικά έως άριστα(10%)

Για να θεωρηθεί κάποιος υποψήφιος κατάλληλος θα πρέπει να συγκεντρώνει σκορ τουλάχιστον 50%.



Εικόνα 22 – Εισαγωγή κριτηρίων αξιολόγησης

Τα αποτελέσματα που παίρνουμε με την παραπάνω αναζήτηση είναι τα παρακάτω:



ID	First name	Last name	Work experience	score(%)	bonus score	Education	score(%)	bonus score	Languages	Total score	Qualified	Actions
7	Kostas	Papadopoulos	Java Developer at Upstream: 01/01/2014 - 01/01/2017	100.0	0.0	Master in Computer Science - UoC	90.0	0.0	English (100%) French (50%)	87.0%	Yes	View
6	Dimitris	Mavroforakis	Software Engineer at Microsoft : 01/01/2013 - 01/07/2016 Web Developer at Google: 01/10/2016 - 01/07/2017	100.0	0.0	BSc in Informatics engineering - TEI of Crete	80.0	0.0	English (70%)	81.0%	Yes	View
8	Maria	Papadaki	Java Developer at Yahoo: 01/01/2016 - 01/01/2017	33.333336	0.0	Phd in Computer Science - UoC	100.0	0.0	English (100%) German (70%)	63.66667%	Yes	View

Εικόνα 23 – Παράδειγμα 1 – Αποτελέσματα αξιολόγησης

Η αίτηση με id 7, συγκεντρώνει 100% για την προϋπηρεσία, καθώς έχει 3 χρόνια εμπειρία (3/3*100 = 100%). Εκπαίδευση επιπέδου μεταπτυχιακού άρα 9/10*100 = 90%.
Γνώση Αγγλικών 10/10*100 = 100%. Γνώσεις γερμανικών δεν αναφέρονται άρα 0%.
Το συνολικό σκορ προκύπτει για αυτήν την αίτηση είναι:
100 * 0.5 + 90 * 0.3 + 100*0.1 + 0*0.1 = 87%

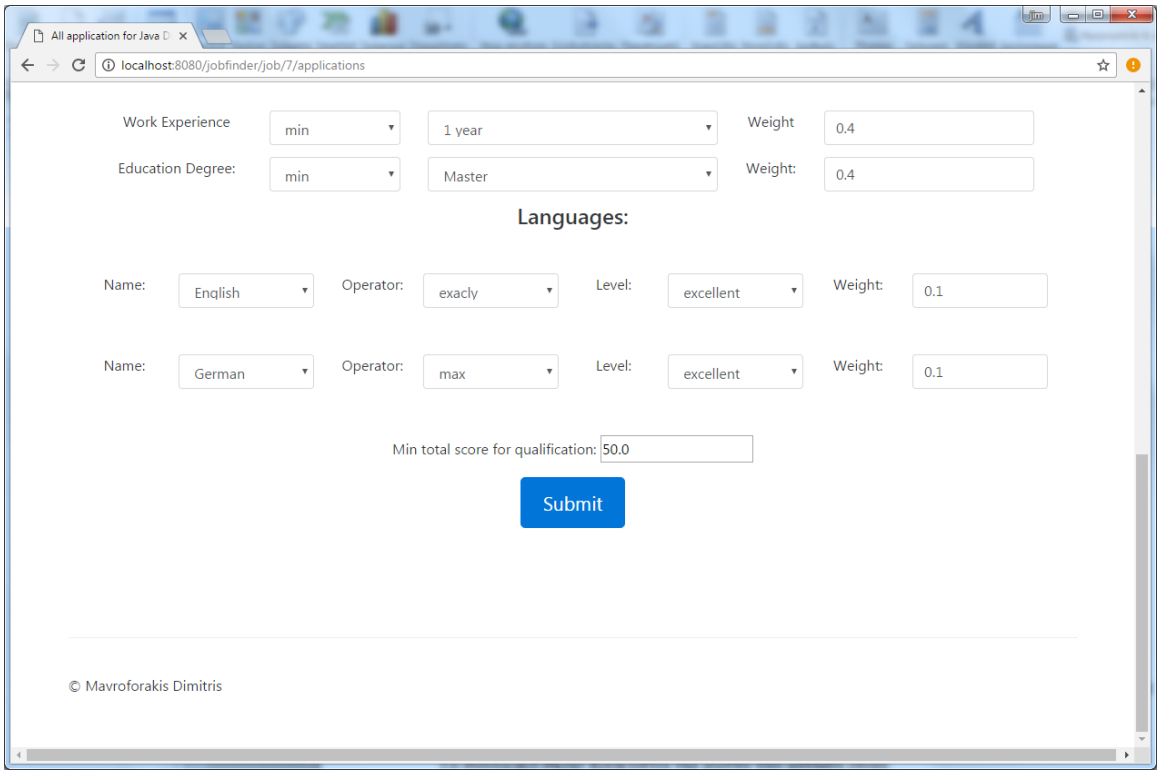
Η αίτηση με id 6, έχει πάνω από 3 χρόνια εμπειρία άρα συγκεντρώνει 100% για την προϋπηρεσία. Η εκπαίδευση είναι επιπέδου προπτυχιακού άρα το σκορ για την εκπαίδευση είναι 8/10*100 = 80%. Για την γνώση ξένων γλωσσών θα βαθμολογηθεί ως εξής:
7/10*100=70% για τα αγγλικά και 0 % για γερμανικά. Το συνολικό σκορ θα είναι:
100*0.5 + 80*0.3 + 70*0.1 = 81%

Η αίτηση με id 8 έχει 1 χρόνο προϋπηρεσία, άρα το σκορ για την εργασιακή εμπειρία θα είναι 1/3*100 = 33,3%. Εκπαίδευση επιπέδου διδακτορικού άρα 10/10 *100 = 100%. Αγγλικά 100% και γερμανικά 70%. Άρα το συνολικό σκορ θα είναι
33,3 *0.5 + 100 * 0.3 + 100*0.1 + 70*0.1 = 63.66%

Παράδειγμα 2:

Προϋπηρεσία τουλάχιστον 1 χρόνο, 40% βαρύτητα
Εκπαίδευση τουλάχιστον μεταπτυχιακό, 40% βαρύτητα
Αγγλικά ακριβώς άριστα(10%), γερμανικά έως άριστα(10%)

Για να θεωρηθεί κάποιος υποψήφιος κατάλληλος θα πρέπει να συγκεντρώνει σκορ τουλάχιστον 50%.



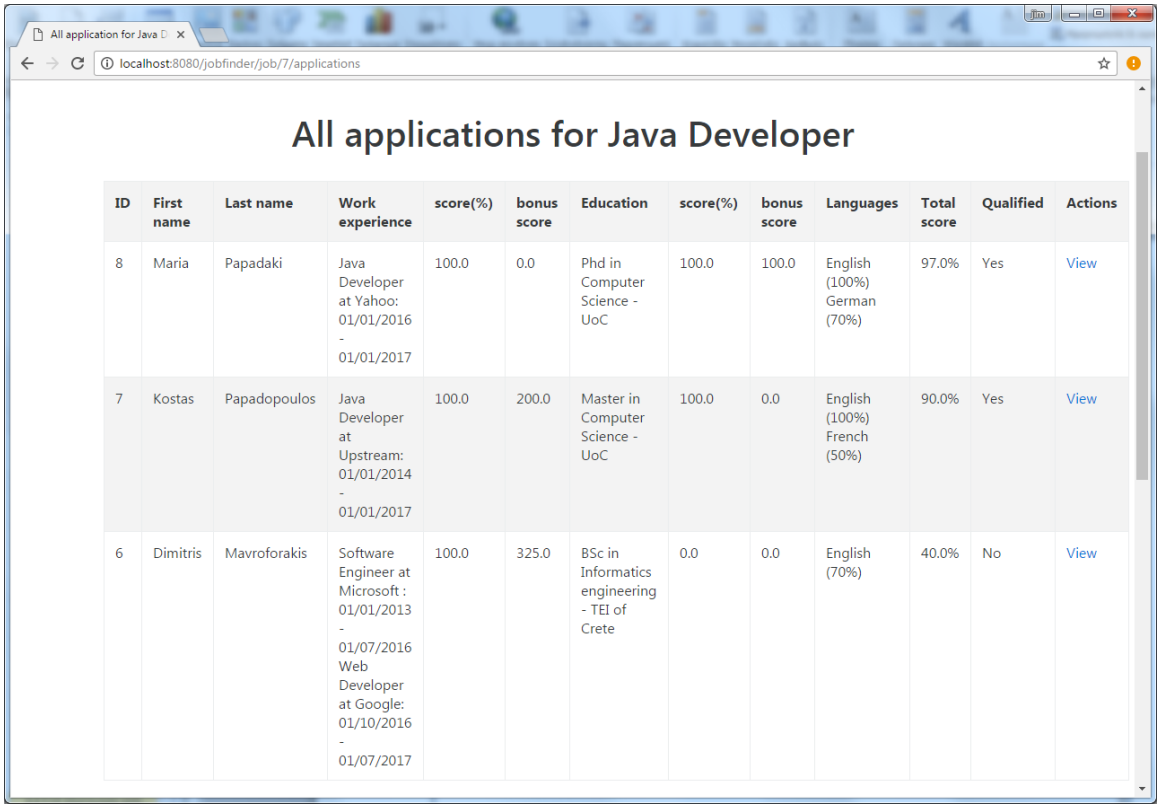
Εικόνα 24 - Παράδειγμα 2 – Εισαγωγή κριτηριών αξιολόγησης

Αποτελέσματα:

Η αίτηση με ID 8 έχει τουλάχιστον 1 χρόνο προϋπηρεσία άρα σε αυτόν τον τομέα θα συγκεντρώσει 100%. Στην εκπαίδευση θα βαθμολογηθεί με 100% καθώς έχει τουλάχιστον μεταπτυχιακό και επίσης θα λάβει bonus 100% επειδή έχει διδακτορικό. Οι συντελεστές εργασιακής εμπειρίας, εκπαίδευσης, γνώσης αγγλικών, γνώσης γερμανικών είναι 40, 40, 10 και 10 % αντίστοιχα, άρα το σκορ που προκύπτει είναι:
100*0.4 + 100*0.4 + 100*0.1 + 70*0.1 = 97%

Αντίστοιχα για την αίτηση με ID 7 το σκορ που προκύπτει είναι
100*0.4 + 100*0.4 + 100*0.1 + 0*0.1 = 90%

Η αίτηση με ID 6 δεν έχει μεταπτυχιακό, άρα για την εκπαίδευση το σκορ θα είναι 0.
Συνολικό σκορ: 100*0.4 + 0*0.4 + 0*0.1 + 0*0.1 = 40% άρα δεν θεωρείται qualified.



Παράδειγμα 2 – Αποτελέσματα Αξιολόγησης

Παράδειγμα 3:

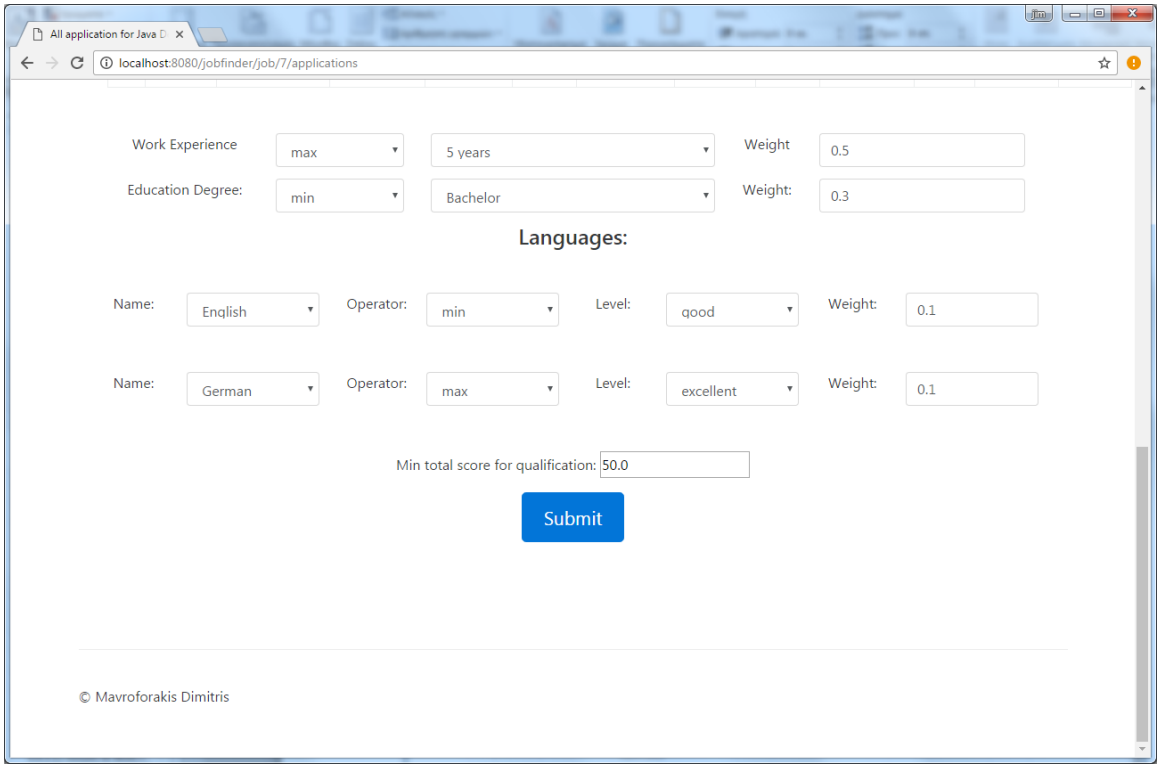
Προϋπηρεσία έως 5 χρόνια, 50% βαρύτητα
Εκπαίδευση τουλάχιστον προπτυχιακό, 30% βαρύτητα
Αγγλικά τουλάχιστον καλά(10%), γερμανικά έως άριστα(10%)

Αποτελέσματα:

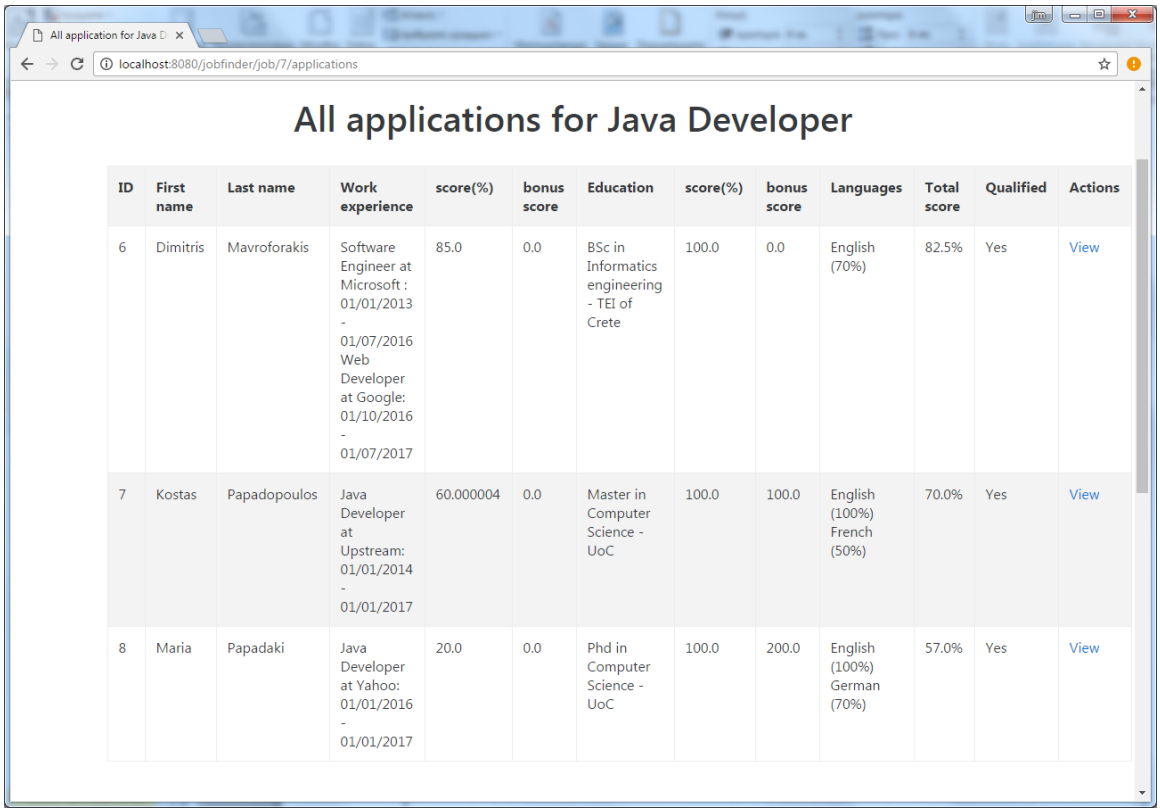
Αίτηση με ID 8:
Προϋπηρεσία συνολικά 4,25 χρόνια άρα Work experience score = 4,25/5 = 85%
Εκπαίδευση ίση με προπτυχιακό άρα Education score = 100%
Η γνώση αγγλικών είναι πολύ καλή (70%) άρα το κριτήριο για ελάχιστη γνώση «καλα» πληρείται κατά 100%.
Συνεπώς το συνολικό σκορ είναι 85*0.5 + 100*0.3 + 100*0.1 + 0 * 0.1 = 82.5%

Αίτηση με ID 7:
Προϋπηρεσία 3 χρόνια άρα Work experience score = 3/5 = 60%
Εκπαίδευση επιπέδου μεταπτυχιακού, Education score = 100% και bonus = 100%,
Αγγλικά άριστα άρα το κριτήριο πληρείται κατά 100%
Σύνολο: 60*0.5 + 100*0.3 + 100*0.1 + 0 * 0.1 = 70%

Αίτηση με ID 6:
Προϋπηρεσία 1 χρόνο άρα Work experience score = 1/5 = 20%
Κατέχει διδακτορικό άρα Education score = 100% και bonus = 200%,
Αγγλικά άριστα άρα το κριτήριο πληρείται κατά 100%, γερμανικά πολύ καλά άρα 70%.
Σύνολο: 20*0.5 + 100*0.3 + 100*0.1 + 70 * 0.1 = 57%

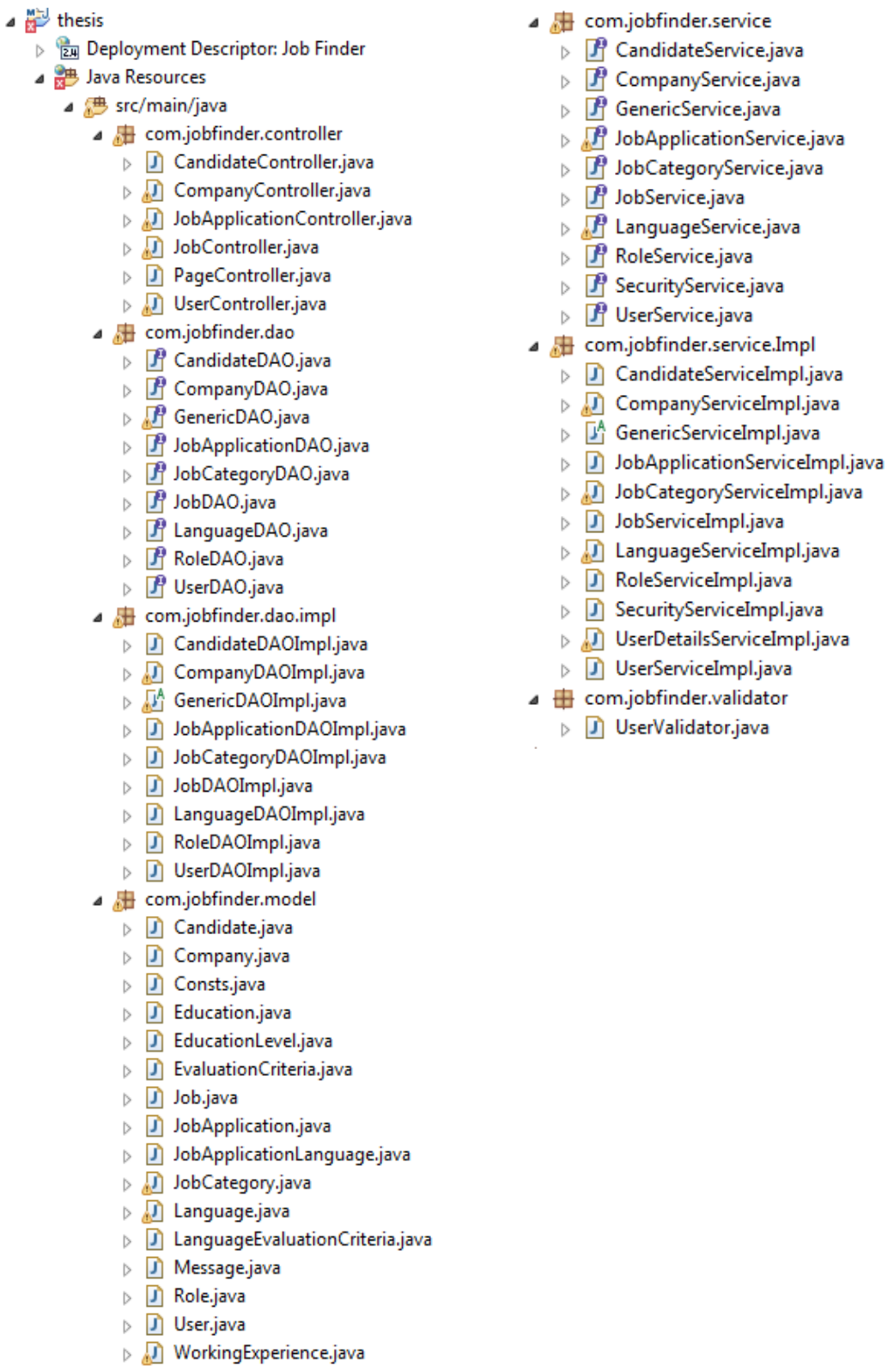


Εικόνα 26 – Παράδειγμα 3 – Εισαγωγή κριτηρίων αξιολόγησης

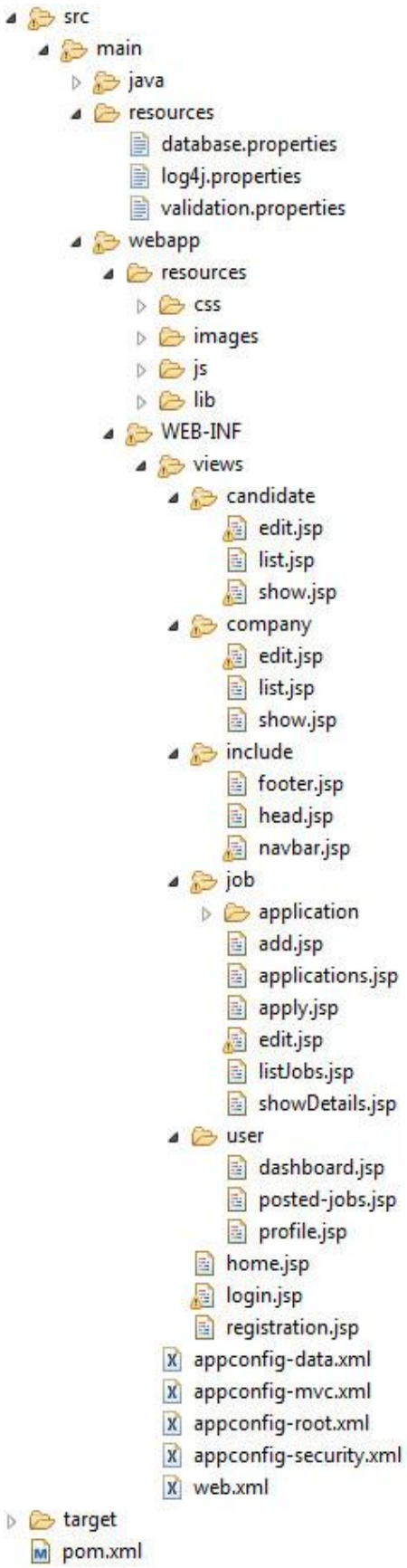


ID	First name	Last name	Work experience	score(%)	bonus score	Education	score(%)	bonus score	Languages	Total score	Qualified	Actions
6	Dimitris	Mavroforakis	Software Engineer at Microsoft : 01/01/2013 - 01/07/2016 Web Developer at Google: 01/10/2016 - 01/07/2017	85.0	0.0	BSc in Informatics engineering - TEI of Crete	100.0	0.0	English (70%)	82.5%	Yes	View
7	Kostas	Papadopoulos	Java Developer at Upstream: 01/01/2014 - 01/01/2017	60.000004	0.0	Master in Computer Science - UoC	100.0	100.0	English (100%) French (50%)	70.0%	Yes	View
8	Maria	Papadaki	Java Developer at Yahoo: 01/01/2016 - 01/01/2017	20.0	0.0	Phd in Computer Science - UoC	100.0	200.0	English (100%) German (70%)	57.0%	Yes	View

Εικόνα 27 – Παράδειγμα 3 – Αποτελέσματα αξιολόγησης



Εικόνα 28 - Τελική μορφή του project – java resources



Εικόνα 29 - Τελική μορφή του project – jsp, xml files

5. Αποτελέσματα

5.1 Συμπεράσματα

Μέσω της τεχνολογικής ανάπτυξης και των υπολογιστικών συστημάτων, δίνεται η δυνατότητα στον άνθρωπο να διεκπεραιώσει χρονοβόρες διαδικασίες μέσα σε ελάχιστο χρονικό διάστημα. Αυτό συνεπάγεται με μείωση του κόστους εργασίας και αύξηση της παραγωγικότητας. Επίσης, η μετατροπή ενός εγγράφου κειμένου σε δομημένη πληροφορία, η οποία στην συνέχεια μπορεί να καταχωρηθεί σε μια βάση δεδομένων, δίνει την δυνατότητα σε ένα υπολογιστικό σύστημα να εκτελέσει πράξεις και διαδικασίες με ανάλογη αποτελεσματικότητα με αυτή ενός ανθρώπου.

5.2 Μελλοντική εργασία και επεκτάσεις

Σε συνέχεια της παρούσας εργασίας, θα μπορούσαμε να προσθέσουμε λειτουργίες όπως προσωπική συνέντευξη με έναν υποψήφιο εργαζόμενο μέσω κάμερας, χρησιμοποιώντας τεχνολογία τύπου WebRTC. Επίσης θα ήταν δυνατή η δημιουργία αυτοματοποιημένων τεστ γνώσεων υποψηφίων, στα οποία θα συμμετείχαν οι υποψήφιοι πρωτού υποβάλλουν αίτηση σε μια θέση εργασίας, με σκοπό η εταιρεία να γνωρίζει σε τι επίπεδο βρίσκονται οι γνώσεις κάποιου υποψηφίου.

Βιβλιογραφία

[1] Deitel, Paul. Deitel, Harvey (2015). Java Προγραμματισμός (Ελληνική Μετάφραση) (10η έκδοση). Αθήνα: Μ. Γκιούρδας, σελ. 19. ISBN 978-960-512-681-0.

[2] Δημοτικότητα Java frameworks: https://zeroturnaround.com/rebellabs/java-tools-and-technologies-landscape-2016/
[3] Πλεονεκτήματα χρήσης framework: http://www.stigmahost.com/why-you-should-be-using-a-php-framework/
[4] Γιαννακός Κωνσταντίνος, Ανάλυση και σχεδίαση ενός πρότυπου διαδικτυακού πληροφοριακού συστήματος διαχείρισης ροής εργασιών, σελ.76. Μεταπτυχιακή Διατριβή, Πανεπιστήμιο Πειραιώς
[5] Βασικά πλεονεκτήματα MVC: http://webapptester.com/mvc-framework-first-impression/
[6] To Spring Framework:
https://www.tutorialspoint.com/spring/spring_web_mvc_framework.htm
[7] Hibernate: http://www.hibernate.org
[8] Bauer C., King G. (2005) Hibernate in Action. Manning. ISBN 1-932394-15-X.
[9] HTML: https://www.w3schools.com/html/
[10] Css: https://www.w3.org/Style/CSS/
[11] David Flanagan, JavaScript: The Definitive Guide (1996)
[12] Apache Maven: https://maven.apache.org/