

Τμήμα Ηλεκτρολόγων
Μηχανικών & Μηχανικών
Υπολογιστών
Ελληνικό Μεσογειακό Πανεπιστήμιο

Πτυχιακή Εργασία

Διαδικτυακή πλατφόρμα εύρεσης και ανάρτησης προγραμματιστικών θέσεων εργασίας

Περπιράκης Ιωάννης - ΑΜ 3991

Επιβλέπων
Αϊβαλής Κωνσταντίνος

ΗΡΑΚΛΕΙΟ 2024

Ευχαριστίες

Ευχαριστώ όλους πίστευαν ότι όντως δούλευα αυτή την πτυχιακή εργασία όλα αυτά τα χρόνια, και ότι δεν τους κορόιδευα.

Σύνοψη

Σκοπός αυτής της πτυχιακής εργασίας, είναι η δημιουργία μιας διαδικτυακής εφαρμογής και η εξοικείωση με σύγχρονα εργαλεία και τεχνολογίες τα οποία χρησιμοποιούνται από τη βιομηχανία σήμερα.

Abstract

The goal of this thesis is the creation of a web application and the familiarization of modern tools and technologies used in the industry today.

Πίνακας Περιεχομένων

Ευχαριστίες 2
Σύνοψη 3
Abstract 4
Πίνακας Περιεχομένων 5
1. Εισαγωγή 7
1.1 Περίληψη 7
1.2 Εστίαση και Στόχοι 8
1.3 Οργάνωση Αναφοράς 8
2. Τεχνολογίες και Αρχιτεκτονική 9
2.1 Τεχνολογίες 9
2.1.1 Java 11
2.1.2 Maven 11
2.1.3 Spring Framework 11
2.1.4 Lombok 12
2.1.5 Bootstrap 12
2.1.6 Thymeleaf 12
2.1.7 Postgresql 13
2.1.8 HTML - CSS - Javascript 13
2.2 Αρχιτεκτονική 14
2.2.1 Εισαγωγή 14
2.2.2 MVC 15
2.2.3 Δομή Κώδικα 20
3. Ανάλυση Κώδικα 21
3.1 Εισαγωγή 21
3.2 Model 22
3.2.1 Entities 22
3.2.2 Repositories 28
3.3 View 31

3.3.1 Layout 31
3.3.2 Reusable Components 32
3.3.3 Security 34
3.4 Controller 35
3.4.1 Controllers 35
3.4.2 Services 42
3.4.3 Dtos 47
3.4.5 Security 49
4. Λειτουργία Εφαρμογής 50
4.1 Αναζήτηση και προβολή 50
4.2 Εγγραφή και σύνδεση 54
4.3 Διαχείριση Αγγελιών 55
5. Σύνοψη 59
5.1 Συμπεράσματα 59
5.2 Μελλοντικές επεκτάσεις 59
6. Βιβλιογραφία 60

1. Εισαγωγή

1.1 Περίληψη

Η έμπνευση αυτής της εφαρμογής προέρχεται το Linkedin. Το Linkedin είναι ένα κοινωνικό δίκτυο το οποίο εστιάζει περισσότερο σε επαγγελματίες. Φέρνει σε επαφή εκατομμύρια χρήστες, από διάφορες εταιρείες και θέσεις εργασίας με σκοπό τη δικτύωση, την ανταλλαγή απόψεων και ιδεών αλλά και την εύρεση εργασίας. Αυτό το τελευταίο κομμάτι - η εύρεση εργασίας - είναι ο σκοπός αυτής της εφαρμογής, με τη διαφοροποίηση ότι απευθύνεται συγκεκριμένα σε προγραμματιστές.

Στην εφαρμογή αυτή, θα δίνεται η ευκαιρίας στους χρήστες να κάνουν εγγραφή, και ανάλογα τον σκοπό τους, δηλαδή αν ψάχνουν δουλεία ή αν ψάχνουν υποψήφιους για δουλειά, να κάνουν τις ανάλογες ενέργειες.

Οι δύο κύριοι ρόλοι κάποιο χρήστη είναι Developer και Recruiter. Και οι δύο ρόλοι έχουν κοινά αλλά έχουν και διαφορές. Για παράδειγμα και οι δύο μπορούν να κάνουν αναζήτηση για κάποια αγγελία, αλλά μόνο ο Recruiter μπορεί να ανεβάσει κάποια αγγελία και να μπορεί να την επεξεργαστείτε ή να δει και να διαχειριστεί τις αιτήσεις σε κάθε αγγελία. Σε αντίθεση, μόνο ο Developer μπορεί να κάνει αίτηση σε κάποια αγγελία ή να φτιάξει το προφίλ του.

1.2 Εστίαση και Στόχοι

Ο σκοπός αυτής της εργασίας είναι η εξοικείωση του φοιτητή με τη δημιουργία δυνατών διαδικτυακών εφαρμογών, χρησημοποιώντας industry standard εργαλεία και το μοντέλο αρχιτεκτονικής MVC.

1.3 Οργάνωση Αναφοράς

Το 1ο κεφάλαιο αναφέρεται στην περίληψη και το κίνητρο αυτής της εργασίας.

Το 2ο κεφάλαιο αναφέρεται στις τεχνολογίες και εργαλεία που χρησιμοποιήθηκαν αλλα και στην αρχιτεκτονική της εφαρμογής.

Το 3ο κεφάλαιο αναφέρεται στην εξέταση και ανάλυση του κώδικα της εφαρμογής.

Το 4ο κεφάλαιο αναφέρεται στην λειτουργία της εφαρμογής περιληπτικά με εικόνες.

Το 5ο κεφάλαιο αναφέρεται στα συμπεράσματα και στις μελλοντικές επεκτάσεις της εφαρμογής.

Το 6ο κεφάλαιο αναφέρεται στη βιβλιογραφία.

2. Τεχνολογίες και Αρχιτεκτονική

2.1 Τεχνολογίες

Δυστυχώς, στο web development δεν μπορείς να διαλέξεις ένα εργαλείο, σε αυτή την περίπτωση γλώσσα προγραμματισμού, και να μείνεις με αυτό από την αρχή έως το τέλος του development.

Η εφαρμογή προορίζεται να χρησιμοποιείται από κανονικούς χρήστες, άρα θέλουμε κάποιο είδος διεπαφής με το οποίο θα αλληλεπιδρά ο χρήστης. Θέλουμε να υπάρχει κάποιο είδος ελέγχου και λογικής, για να μπορεί ο χρήστης να βρίσκει αυτά που θέλει και να μπορεί να κάνει τις ενέργειες που του έχουν οριστεί, και τέλος, θέλουμε όλα αυτά να αποθηκεύονται με δομημένο τρόπο

Αυτές είναι οι 3 βασικές προδιαγραφές της εφαρμογής μας. Οπότε κάποιος θα έλεγε ότι αφού δεν μπορούμε να τα κάνουμε όλα με ένα εργαλείο, τότε μπορούμε να διαλέξουμε ένα για το καθένα. Κατα κάποιο τρόπο ναι, πρακτικά το ηλεκτρικό κατσαβίδι είναι ένα εργαλείο, όταν όμως του αλλάξεις την κεφαλή μπορείς να το κάνεις και τρυπάνι.

Στον κόσμο του προγραμματισμού, αυτα τα εργαλεία λέγονται ***βιβλιοθήκες*** και ***frameworks***.

Οι ***βιβλιοθήκες*** συνήθως προσφέρουν κάποιες επεκτάσεις και συντομεύσεις που συνήθως απευθύνονται σε πιο συγκεκριμένα προβλήματα. Ένα παράδειγμα βιβλιοθηκης θα μπορούσε να είναι για τη διαχείριση μιας λίστας δεδομένων που προσφέρει, έτοιμες μεθόδους για προσθήκη, αφαίρεση ή ταξινόμηση, οπου η εναλλακτική θα ήταν να

γραφτούν αυτές οι μέθοδοι απο την αρχή, πράγρα που θα ήταν χρονοβόρο και ίσως επίφοβο για λάθη.

Το ***framework*** από την άλλη προορίζεται για πιο γενική χρήση για την επίλυση κάποιου πιο μεγαλύτερου προβλήματος, όπως για παράδειγμα για τη δημιουργία μιας διαδικτυακής εφαρμογής. Το framework μπορει να συμπεριλαμβάνει πολλές βιβλιοθήκες μέσα σε αυτό. Τα frameworks συνήθως, σε αντίθεση με τις βιβλιοθήκες, είναι πιο δομημένα και προσφέρουν περισσότερη καθοδήγηση [1].

Και τα δύο μπορούν να μειώσουν δραματικά τον χρόνο που θα χρειαστεί η δημιουργία μιας εφαρμογής ,και επειδή συνήθως συντηρούνται απο πολύ κόσμο ή ακόμα και εταιρίες, που πολλές φορές όλοι αυτοί μπορεί να είναι πιο ειδικοί στο θέμα που απευθύνεται η βιβλιοθήκη ή το framework, να είναι πιο αξιόπιστα τα αποτελέσματα από κάτι το οποίο θα φτιάξει κάποιος από την αρχή.

Παρακάτω ακολουθεί μια λίστα με τις κύριες τεχνολογίες, βιβλιοθήκες και frameworks που χρησιμοποιήθηκαν για τη δημιουργία αυτης της εφαρμογής. Όλες οι τεχνολογίες, βιβλιοθήκες και frameworks που χρησιμοποιήθηκαν είναι ανοιχτού κώδικα (open source).

2.1.1 Java

Η ***Java*** είναι μία υψηλού επιπέδου αντικειμενοστραφής γλώσσα προγραμματισμού. Με τη βοήθεια του ***JVM*** (Java Virtual Machine) μπορεί να τρέξει παντού. Είναι μία ώριμη γλώσσα που βλέπει εφαρμογές σε σχεδόν όλους τους τομείς (web, desktop, cloud, mobile, embedded) και αυτός είναι ένας από τους λόγους που την κάνει πολύ ελκυστική σε μεγάλες εταιρίες με πολλές απαιτήσεις [2].

Στη δική μας περίπτωση, θα χρησιμοποιηθεί από την αρχή μέχρι το τέλος, δηλαδή από τη διεπαφή του χρήστη μέχρι και τη βάση δεδομένων.

2.1.2 Maven

Το ***Maven*** είναι ένα εργαλείο αυτοματισμού κατασκευής που χρησιμοποιείται κυρίως για έργα Java άλλα μπορεί επίσης να χρησιμοποιηθεί για τη δημιουργία και τη διαχείριση έργων γραμμένων και άλλες γλώσσες. Το Maven εξετάζει δύο πτυχές της κατασκευής λογισμικού: τον τρόπο ***κατασκευής*** (build) του λογισμικού και τις ***εξαρτήσεις*** (dependencies) του [3].

2.1.3 Spring Framework

Το ***Spring Framework*** είναι ένα πλήρες framework της Java. Αυτό το καταφέρνει με την προώθηση καλών πρακτικών σχεδιασμού (good design practices) και τη ***μείωση του boilerplate*** κώδικα. Στη δική μας περίπτωση, θα βοηθήσει στη ***ταχύτερη υλοποίηση*** της web εφαρμογής μας με τη μείωση του κώδικα που χρειάζεται, στην ασφάλεια με εργαλεία που παρέχει για ***authentication &***

authorization, και στην ***δημιουργία & επικοινωνία με τη βάση δεδομένων*** [4].

2.1.4 Lombok

Η ***Lombok*** είναι μία βιβλιοθήκη της Java που έχει σκοπό να μειώσει κατα πολύ τον boilerplate κώδικα που χρειάζεται η Java, με την χρήση decorators. Μία από τις πιο κοινές επικρίσεις για την Java είναι η πολυλογία της — και δικαίως. Υπάρχουν πολλά απλά πράγματα τα οποία γράφουμε πολλές φορές όπως ***getters, setters*** και ***constructors*** [5]***.***

2.1.5 Bootstrap

Η ***Bootstrap*** είναι η πιο γνωστή βιβλιοθήκη ***HTML/CSS*** με pre-styled, responsive και cross-platform UI components όπως cards, tables και πολλά άλλα. Επίσης, με τη χρήση της ***Javascript***, μπορεί να προσφέρει components με πιο σύνθετες λειτουργίες όπως accordions, tabs και άλλα. Η χρήση της μπορεί να μειώση σημαντικά το development time και να παρέχει στην εφαρμογή μια ομοιόμορφη εικόνα. Με τη βοήθεια του ***Bootswatch***, που παρέχει διαφορετικά themes πάνω στη Bootstrap, η εφαρμογή πας έχει ένα πιο ιδιαίτερο look & feel [6].

2.1.6 Thymeleaf

Το Thymeleaf είναι μια σύγχρονη μηχανή προτύπων (***template engine***) απο την πλευρά του διακομιστή (server) για αυτόνομες (standalone/desktop) και για διαδικτυακές εφαρμογές. Θας μας

βοηθήσει στην προβολή περιεχομένου στον χρήστη κατευθείαν απο τον server άλλα επίσης μπορεί να διατηρήσει και ακόμα να αποκρύψει περιεχόμενο από τον χρήστη, προσφέροντας έτσι ένα ακόμα επίπεδο ασφάλειας. Επίσης, θα μας βοηθήσει να φτιάξουμε δικά μας πρότυπα για επαναχρησιμοποιούμενα κομμάτια κώδικα (reusable components) και μπορεί να προσφέρει μια καλή δομή για τις σελίδες μας [7].

2.1.7 Postgresql

Η ***Postgres*** είναι ένα πάρα πολύ ισχυρό σύστημα διαχείρισης βάσεων δεδομένων ή αλλιως ***RDBMS***. Συμμορφώνεται με το πρότυπο ***SQL*** (structured query language) αλλα και με το πρότυπο ***ACID*** (Atomicity, Consistency, Isolation, Durability) με το οποίο εγγυάται την αξιοπιστία των συναλλαγών.

Κάποια άλλα από τα δυνατά της χαρακτηριστικά που την κάνουν να διαφέρει από άλλες SQL βάσεις, είναι η υποστήριξη ***προηγμένων τύπων δεδομένων*** όπως XML,JSON αλλα και geometric και network addresses. Επίσης έχει ένα πολύ δυνατό ***Concurrency Control*** που επιτρέπει να γίνεται ταυτόχρονα πολλαπλές συναλλαγές στη βάση χωρίς να «κλειδώνει» ολόκληρη τη βάση.

Πέρα από αυτά, υπάρχει δυνατή υποστήριξη από την κοινότητα μιας και είναι ***open source*** και είναι και ***cross platform*** [8].

2.1.8 HTML - CSS - Javascript

The bread and butter of the web. Οι μόνες γλώσσες που καταλαβαίνει κάποιος browser. Ακόμα και όλα τα μοντέρνα frontend framework όπως React,Angular,Svelte κτλ, στο τέλος θα γίνουν compile σε αυτές τις 3 γλώσσες. Η ***HTML*** (Hypertext Markup

Language) με την βοήθεια των tags ή elements, είναι υπεύθυνη για τη δομή της σελίδας, για το πού πάει τι δηλαδή. Η ***CSS*** (Cascading Style Sheets) με την βοήθεια των κλάσεων (css classes) παρέχει εργαλεία για τη μορφοποίηση όλως αυτων των HTML tags όπως για παράδειγμα χρώματα, spacing, fonts κτλ. Και τέλος η ***Javascript*** είναι υπεύθυνη για την αλληλεπίδραση της σελίδας. Και οι 3 αυτές τεχνολογίες μπορούν να δουλέψουν μεταξύ εξαιρετικά, αλλά επειδή μεταξύ τους μόνο η Javascript είναι γλώσσα προγραμματισμού, μόνο αυτή μπορεί να παρέχει λογική μεταξύ αυτών [9].

2.2 Αρχιτεκτονική

2.2.1 Εισαγωγή

Η επιλογή του αρχιτεκτονικού προτύπου για την ανάπτυξη λογισμικού είναι κομβικό, διαμορφώνοντας τη δομή και τη συντηρησιμότητα ολόκληρου του έργου. Το 95%, αν όχι παραπάνω, των διαδικτυακών εφαρμογών είναι ***CRUD*** (create read update and delete), οπότε χρειαζόμαστε έναν απλό τρόπο για να μπορούμε να διαχειριστούμε αυτές τις ενέργειες, αλλα επεκτάσιμο αρκετά ώστε να μπορούμε να προσθέσουμε περισσότερες και πιο σύνθετες ενέργειες.

Κάνοντας μία σύντομη αναζήτηση στο internet, γρήγορα θα συμπεράνουμε οτι η πιο «hot» μέθοδος δημιουργίας μιας διαδικτυακής εφαρμογής σήμερα είναι ένα Rest API για το backend και κάποιο frontend framework για το τη διεπαφή. Ενώ αυτή η προσέγγιση έχει τα καλά, όπως για παράδειγμα η ευρεία γκάμα εργαλείων που μπορούν να χρησιμοποιηθούν, έχει και τα κακά της, και στη συγκεκριμένη περίπτωση δεν είναι τόσο απλή, γρήγορα και εύκολα επεκτάσιμη.

Για αυτόν το λόγο κατέληξα σε μια πιο παλιότερη μεν προσέγγιση, άλλα αυτό την καθιστά περισσότερο δοκιμασμένη.

2.2.2 MVC

Το ***MVC*** (Model-View-Controller) είναι ένα μοτίβο στο σχεδιασμό λογισμικού που χρησιμοποιείται συνήθως για την υλοποίηση διεπαφών χρήστη. Δίνει έμφαση στον διαχωρισμό μεταξύ της επιχειρηματικής λογικής (business logic) του λογισμικού και της οθόνης/διεπαφής. Αυτός ο διαχωρισμός των ανησυχιών (separation of concerns) προβλέπει καλύτερο καταμερισμό εργασίας και βελτιωμένη συντήρηση και επεκτασιμότητα.

Όπως φαίνεται και απο την ονομασία, διαιρεί μια εφαρμογή σε τρία διασυνδεδεμένα στοιχεία:

1) Ο ***Controller*** μπορεί να στέλνει εντολές στο model και να ενημερώνει την κατάσταση του model . Μπορεί επίσης να στέλνει εντολές ώστε να γίνει η αντίστοιχη αναπαράσταση των δεδομένων του model μέσω του view.

2) Το ***Model*** ενημερώνει τις αντίστοιχες αναπαραστάσεις views και τους controllers όταν υπάρχει αλλαγή στα δεδομένα. Αυτή η

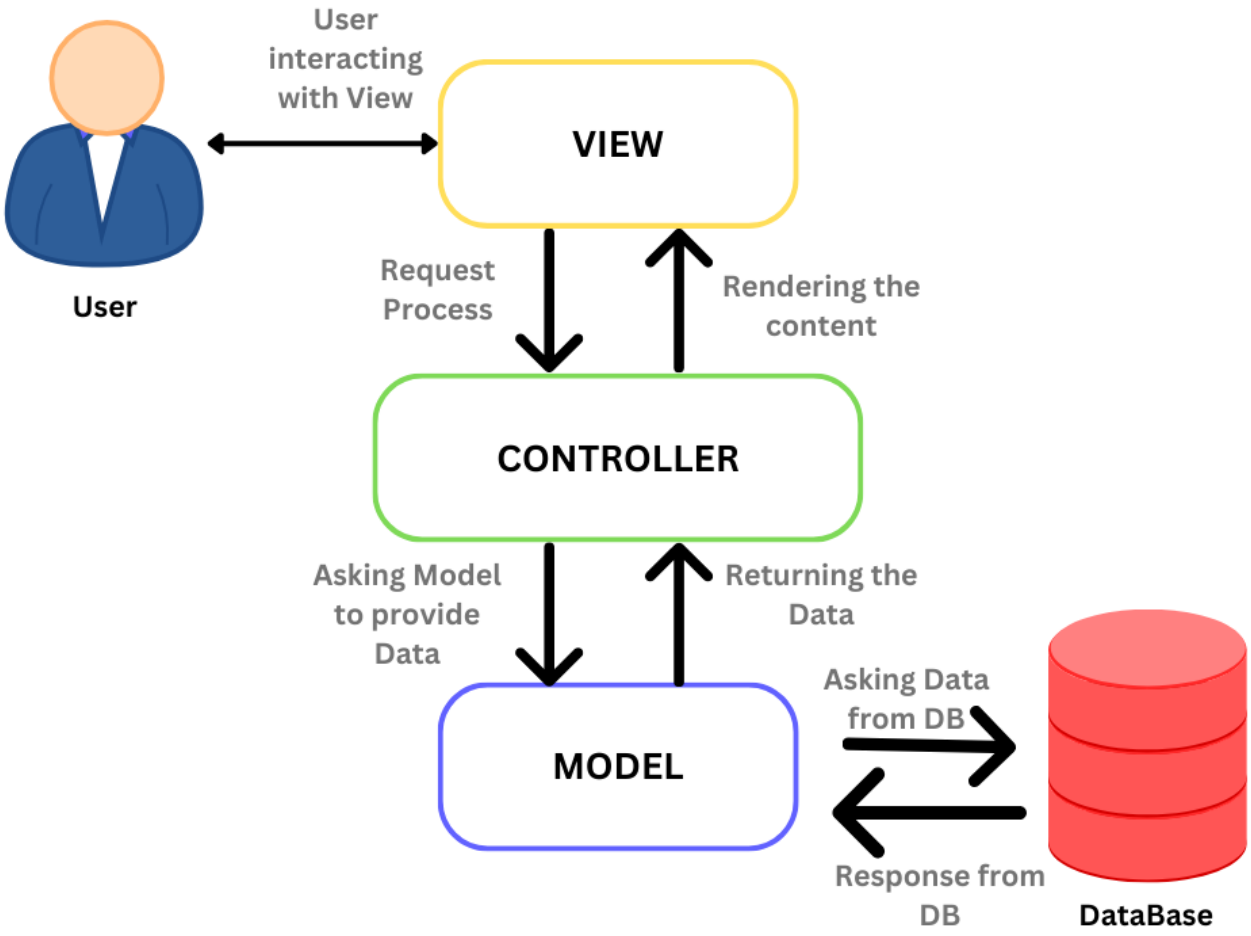
ενημέρωση επιτρέπει στα views να ενημερώνουν τη γραφική απεικόνιση.

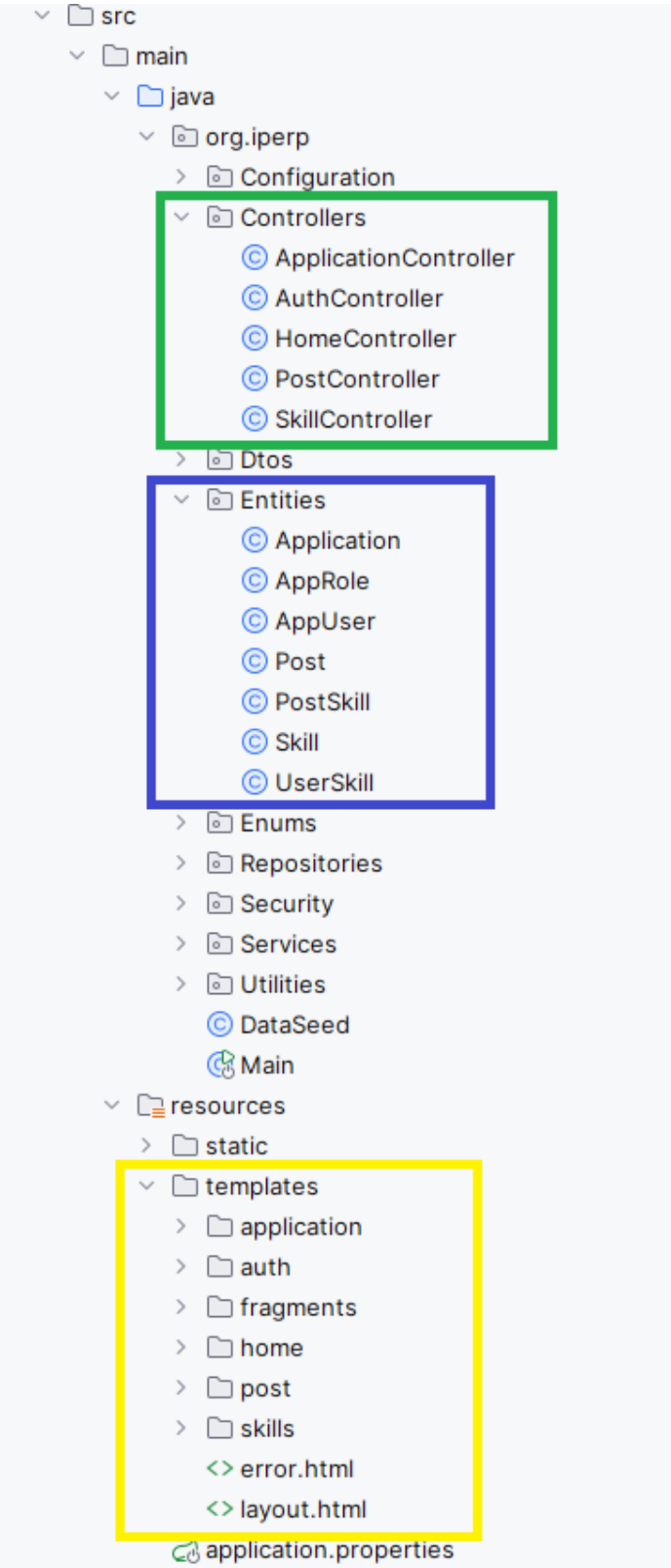
3) Το ***View*** αναπαριστά με γραφικό τρόπο την πληροφορία που περιέχει το model δημιουργώντας γραφική παρουσίαση στο χρήστη.

Με το να διαχωρίζονται έτσι τα κύρια στοιχεία, μας παρέχεται μια δομή και μια καθοδήγηση για το πως μπορούμε να επεκτείνουμε και να συντηρούμε την εφαρμογή μας. Οπότε πολύ γρήγορα βλέπουμε πως έχουμε βρει 3 πολύ καλούς λόγους για την επιλογή αυτής της προσέγγισης:

- Separation of Concerns
- Maintainability
- Scalability

Παρακάτω ακολουθεί ένας γενικό διάγραμμα απεικόνισης της ***MVC*** αρχιτεκτονικής μαζί με μία φωτογραφία της δομής του τελικού κώδικα, με τα αντίστοιχα στοιχεία κυκλωμένα **[10]**.


User
interacting
with View
VIEW
User
Request
Process
Rendering the
content
CONTROLLER
Asking Model
to provide
Data
Returning the
Data
MODEL
Asking Data
from DB
Response from
DB
DataBase


src
main
java
org.iperp
Configuration
Controllers
ApplicationController
AuthController
HomeController
PostController
SkillController
Dtos
Entities
Application
AppRole
AppUser
Post
PostSkill
Skill
UserSkill
Enums
Repositories
Security
Services
Utilities
DataSeed
Main
resources
static
templates
application
auth
fragments
home
post
skills
error.html
layout.html
application.properties

Γρήγορα κάποιος θα παρατηρήσει τις ονομασίες αρχείων, ακόμα και τα Services και τα Repositories ακολουθούν την ίδια μέθοδο ονομασίας, κοινώς ***Naming Conventions***. Αυτό δεν είναι τυχαίο και είναι κάτι που ακολουθείται από σχεδόν όλα τα MVC frameworks. Το πώς θα επιλέξουμε να ονομάσουμε κάτι είναι αρκετά σημαντικό για 2 βασικούς λόγους:

1. Για να μπορεί κάποιος άλλος πέρα από εμάς, όταν διαβάζει τον κώδικα μας να καταλαβαίνει μια μία γρήγορη ματιά τι είναι.

2. Για να μπορεί να καταλάβει ο ίδιος ο κώδικας τι είναι το καθένα. Απο σχεδιασμού τα MVC frameworks ακολουθούν αυτή την τακτική και για λόγους λειτουργίας άλλα και για λόγους ευκολίας. Για παράδειγμα ο ***PostController*** ξέρει ότι όλα του τα views βρίσκονται στον ***φάκελο post*** απο default. Εκεί θα είναι το πρώτο μέρος που θα πάει να ψάξει αν δεν λάβει διαφορετικές οδηγίες.

Έτσι λοιπών βλέπουμε πως επαληθεύονται και τα 3 θετικά της αρχιτεκτονικής ***MVC.*** Αν θέλουμε να διορθώσουμε ή να επεκτείνουμε μια λειτουργία της εφαρμογής μας, έχουμε πολύ γρήγορα μια γενική εικόνα και έτσι μπορούμε να κατευθυνθούμε πιο εύκολα και γρήγορα στο σωστό σημείο ή αν θέλουμε να προσθέσουμε νέες λειτουργίες, έχουμε ήδη τα σχέδια έτοιμα.

2.2.3 Δομή Κώδικα

Τώρα θα δούμε πως η παραπάνω αρχιτεκτονική αντιστοιχίζεται στον κώδικά μας.

Θα ξεκινήσουμε από το τέλος και θα φτάσουμε στην αρχή, δηλαδη θα ξεκινήσουμε από τη βάση δεδομένων και θα φτάσουμε στη διεπαφή του χρήστη.

- Για το ***Model***, έχουμε τους φακέλους ***Entities*** και ***Repositories***.
- Για το ***Controller*** έχουμε τους φακέλους ***Controllers*** και ***Services***.
- Για το ***View***, έχουμε τον φάκελο ***Dtos*** και ***templates.***

Όλοι οι υπόλοιποι φάκελοι είτε περιέχουν πράγματα που χρησιμοποιούνται σε διάφορα σημεία στης εφαρμογή μας είτε περιέχουν configuration settings για όλη μας την εφαρμογή

Για κάθε διαφορετικό στοιχείο της εφαρμογής μας, υπάρχει και το αντίστοιχο αρχείο σε κάθε ένα από τους παραπάνω φακέλους.

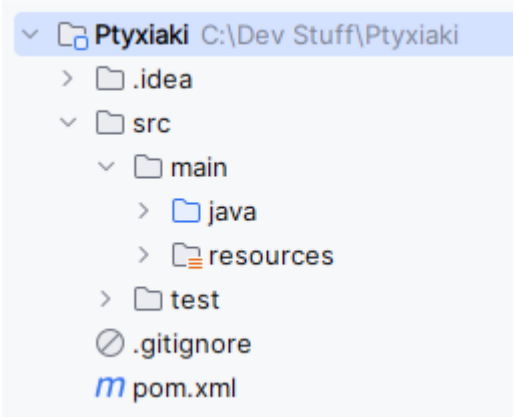
Για παράδειγμα, για το βασικό στοιχείο της εφαρμογής μας είναι η αγγελία της δουλειάς. Αυτή αντιπροσωπεύεται απο το entity ***Post.*** Άρα έχουμε και ***PostController, PostService, PostRepository και τέλος PostDto.*** Το αντίστοιχο γίνεται και για τις αιτήσεις, χρήστες κτλ. Όλα τα αρχεία αυτά είναι ανεξάρτητα και έχουν διαφορετικές ευθύνες για να μπορούν όλα τα κομμάτια της εφαρμογής μας να είναι εύκολα επεκτάσιμα και να μπορούν να τροποποιούνται με ευκολία. Για αυτό το λόγο έχουμε 5 controllers και όχι 1.

3. Ανάλυση Κώδικα

Το repository του κώδικα μπορεί να βρεθεί εδώ.
Live demo της εφαρμογής προσωρινά μπορεί να βρεθεί εδώ.

3.1 Εισαγωγή

Σε αυτή την ενότητα, θα κοιτάξουμε αναλυτικά τον κώδικα. Πρώτα θα εστιάσουμε στα θεμελιώδη κομμάτια και μετά θα εστιάσουμε σε κάθε ειδικό στοιχείο της εφαρμογής μας ξεχωριστά.



Αυτή είναι η πολύ γενική δομή του project μας. Οι φάκελοι που μας ενδιαφέρουν εμάς είναι ο ***Java*** και ο ***resources*** και το αρχείο ***pom.xml***.

Στον φάκελο Java βρίσκεται όλως ο «καθαρός» μας κώδικας, δηλαδή τα στοιχεία ***Controller*** και ***Model*** της αρχιτεκτονικής μας.

Στον φάκελο resources, βρίσκεται όλα τα στοιχεία της διεπαφής μας, δηλαδή το στοιχείο ***View*** της αρχιτεκτονικής μας.

Το αρχείο pom.xml είναι αξιοποιείται απο το εργαλείο Maven και περιέχει τις πληροφορίες του project μας που είναι χρήσιμες για το development αλλα και το build όπως ποια version της Java χρησιμοποιούμε, τι dependencies έχει η εφαρμογή μας και τις οδηγίες για το build όταν τελειώσουμε και είμαστε έτοιμοι να το ανεβάσουμε σε κάποιο server online.

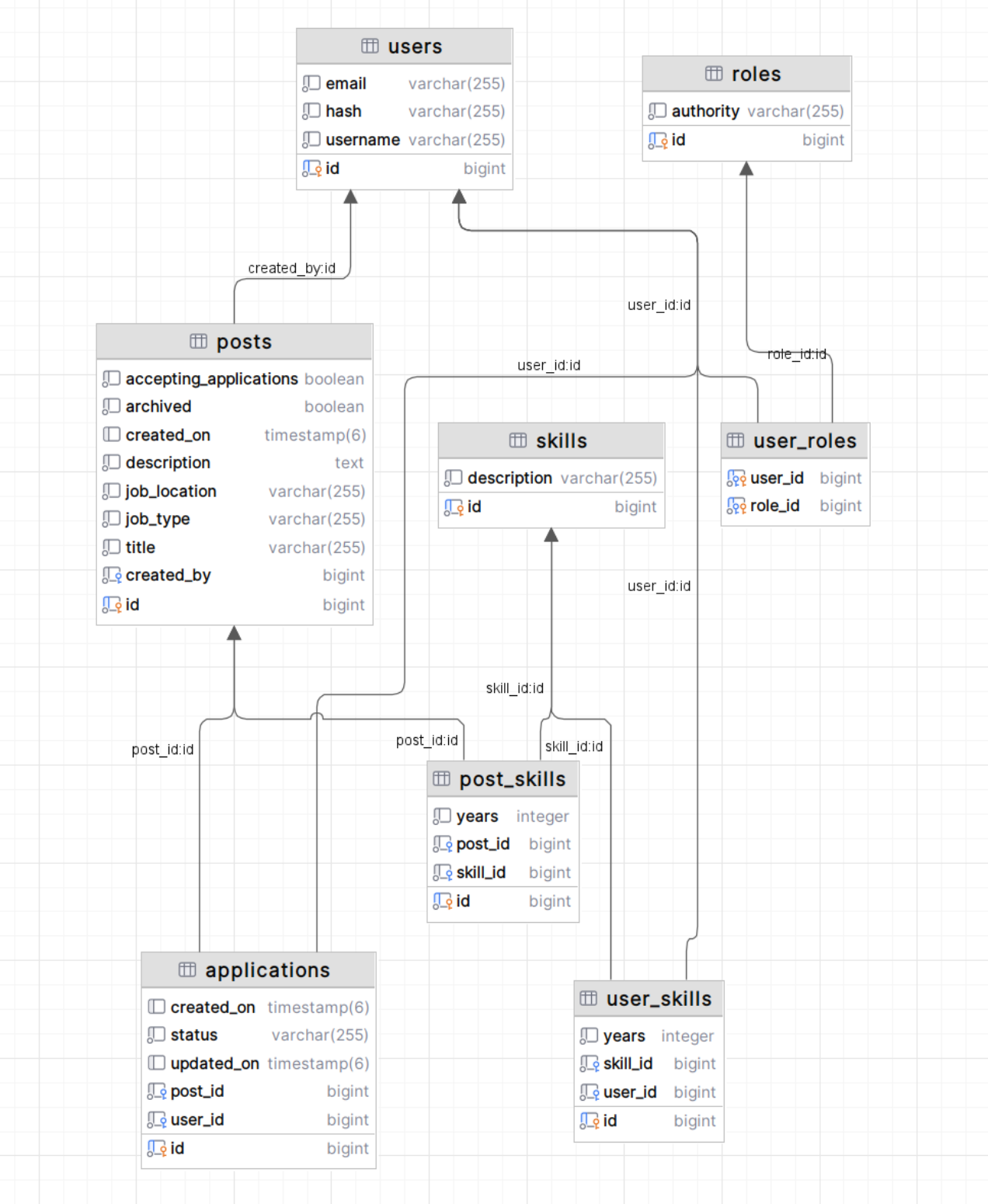
3.2 Model

Το Model όπως αναφέραμε στην αρχή, εκπροσωπεί τα δεδομένα της εφαρμογής μας. Αυτό περιλαμβάνει τη δομή τους, δηλαδή τις πληροφορίες που χρειαζόμαστε για κάθε στοιχείο άλλα και τις πιθανές του σχέσεις με άλλα στοιχεία, και με ποιόν τρόπο αναρτώνται, προσθέτονται και τροποποιούνται.

Για αυτό λοιπόν τον λόγο, θα το χωρίσουμε σε δύο κύρια μέρη. Το πρώτο είναι το ***Entities*** όπου περιλαμβάνει τη δομή των δεδομένων μας, και το δεύτερο είναι το ***Repositories*** όπου περιλαμβάνει τα query μας.

3.2.1 Entities

Η εφαρμογή μας αποτελείται συνολικά από 8 οντότητες. Παρακάτω ακολουθεί ένα διάγραμμα με όλες τις οντότητες και τις σχέσεις μεταξύ αυτών.


users
email varchar(255)
hash varchar(255)
username varchar(255)
id bigint
roles
authority varchar(255)
id bigint
created_by:id
user_id:id
role_id:id
posts
accepting_applications boolean
archived boolean
created_on timestamp(6)
description text
job_location varchar(255)
job_type varchar(255)
title varchar(255)
created_by bigint
id bigint
skills
description varchar(255)
id bigint
user_roles
user_id bigint
role_id bigint
skill_id:id
post_id:id
post_skills
years integer
post_id bigint
skill_id bigint
id bigint
applications
created_on timestamp(6)
status varchar(255)
updated_on timestamp(6)
post_id bigint
user_id bigint
id bigint
user_skills
years integer
skill_id bigint
user_id bigint
id bigint

Στις σχεσιακές βάσεις δεδομένων υπάρχουν 3 είδη σχέσεων που μπορεί να έχουν οι οντότητες μεταξυ τους.

1. One to One (1-1)

Στην εφαρμογή μας δεν υπάρχει αυτό το είδος σχέσης, τουλάχιστον όχι άμεσα. Θα μπορούσαμε όμως για παράδειγμα να προσθέσουμε την οντότητα user_profile όπου εκεί θα αποθηκεύαμε όλες τις πληροφορίες του χρήστη και να συνδέαμε αυτό με την οντότητα user. Άρα ένας χρήστης έχει ένα προφίλ

2. One to Many (1-N)

Απο την άλλη, η εφαρμογή μας έχει δύο σχέσεις αυτού του είδους. Συγκεκριμένα ένας χρήστης μπορεί να έχει πολλές αναρτημένες αγγελίες και ένας χρήστης μπορεί να έχει πολλές αιτήσεις σε αγγελίες.

3. Many to Many (N-M)

Στην εφαρμογή μας έχουμε τρεις τέτοιες σχέσης, ένας χρήστης μπορεί να έχει πολλούς ρόλους και ένας ρόλος μπορεί να χρησιμοποιηθεί από πολλούς χρήστες, ένας χρήστης μπορεί να έχει πολλές δεξιότητες και μία αγγελία μπορεί να έχει πολλές δεξιότητες.

Για όλες αυτες τις οντότητες, υπάρχει και το ανάλογο entity στον κώδικα μας, πέρα απο τον πίνακα user_roles.

Παρακάτω ακολουθεί η απεικόνιση μιας οντότητας σε μορφή Java Class.

```java
@Data
@NoArgsConstructor
@AllArgsConstructor
@Builder
@Entity
@Table(name = "users")
public class AppUser {

   @Id
   @GeneratedValue(strategy = GenerationType.IDENTITY)
   private Long id;

   @Column(nullable = false)
   private String username;

   @Column(nullable = false)
   private String name;

   @Column(nullable = false)
   private String surname;

   @Column(nullable = false)
   private String email;

   @Column(nullable = false)
   private String hash;

   @ManyToMany(fetch = FetchType.EAGER)
   @JoinTable(
           name = "user_roles",
           joinColumns = @JoinColumn(name = "user_id"),
           inverseJoinColumns = @JoinColumn(name = "role_id")
   )
   private Set<AppRole> roles = new HashSet<>();

   @OneToMany(mappedBy = "user", cascade = CascadeType.ALL, orphanRemoval =
true, fetch = FetchType.EAGER)
   private List<UserSkill> skills = new ArrayList<>();
}
```

Οι decorators `@Data @NoArgsConstructor @AllArgsConstructor @Builder` προέρχονται από την βιβλιοθήκη Lombok και δημιουργούν constructors, getters και setters.

Οι decorators `@Entity @Table(name = "users")` Προέρχονται απο το Spring Framework και ουσιαστικά καθιστουν την κλάση AppUser ως οντότητα. Όταν τρέξει πρώτη φορά ο κώδικας μας, θα μας δημιουργήσει αυτόματα στη βάση το table με όνομα users. Αυτό θα το κάνει για όλες τις κλάσεις που υπάρχουν αυτοί οι decorators. Κάθε φορά που τρέχει ο κώδικάς μας και βλέπει αλλαγές σε κάποια από τις κλάσεις με αυτούς τους decorators, τοτε κάνει και τις ανάλογες αλλαγές στη βάση.

Οι decorators `@Id @GeneratedValue(strategy = GenerationType.IDENTITY)asd` αναφέρονται συγκεκριμένα στο πεδιο id, για όταν φτιαχτεί το table στη βάση να το χειριστεί ανάλογα.

Ο decorator `@Column(nullable = false)` Δίνει την ιδιότητα στο πεδίο να μην είναι null, δηλαδή να μην είναι αδειο όταν δημιουργείται ένα νέο record AppUser.

Μετά ακολουθούν τα πεδία που απαιτούν κάποιου είδους σχέση με κάποια άλλη οντότητα.

Για τους ρόλους βλέπουμε ότι θέλει σχέση Many to Many και ότι πρέπει να γίνει με το ενδιάμεσο table user_roles με τις στήλες user_id και role_id.

Το `FetchType.EAGER` σημαίνει ότι όταν γίναι κάποιο query και κάποιον AppUser, τότε θα πρέπει να φέρει και τα που σχετίζονται με αυτήν τη σχέση. Στη συγκεκριμένη περίπτωση, οταν γίνει κάποιο query

για να βρούμε κάποιον χρήστη, τότε να φέρει και τη λίστα με τους ρόλους του αλλά και τη λίστα με τις δεξιότητές του.

Για τις δεξιότητες βλέπουν ότι θέλει σχέση One to Many. Το `mappedBy = "user"` σημαίνει ότι ο user είναι ο «ιδιοκτήτης» της σχέσης και διαχειρίζεται το UserSkill.

Το `cascade = CascadeType.ALL` σημαίνει ότι αν κάπως αλλάξει ο AppUser τότε πρέπει να επηρεαστεί και αυτό, για παράδειγμα αν διαγραφεί ο χρήστης να διαγραφούν και οι αυτά.

Εδώ μπορούμε να δούμε τα θετικά του να χρησιμοποιείτε κάποιο framework. Προσθέτοντας μερικούς decorators καταφέραμε να εξασφαλίσουμε κάποιο δομή και ασφάλεια μεταξύ των οντοτήτων μας και μέσα στη βάση άλλα και μέσα στον κώδικα μας, χωρίς να χρειαστεί καθόλου να γράψουμε κώδικα SQL. Αυτή η προσέγγιση δημιουργίας μιας βάσης δεδομένων ονομάζεται Code First Approach.

Πλέων έχουμε μία ακριβής απεικόνιση της βάσης μας μέσα στον κώδικα.

Συγκεκριμένα στης εφαρμογή μας, δεν γράφτηκε ούτε μία γραμμή σε SQL για τη δημιουργία της βάσης.

Σε καμία περίπτωση αυτό δεν σημαίνει ότι δεν χρειάζεται κάποιος να ξέρει SQL.

Την ίδια φιλοσοφία ακολουθούν όλες οι κλάσεις που αντιπροσωπεύουν κάποια οντότητα.

3.2.2 Repositories

Το Repository αποτελεί τον συνδετικό κρίκο μεταξύ μιας βάσης δεδομένων και μίας εφαρμογής. Περιέχει ένα σύνολο βασικών μεθοδος για όλες της ενέργειες (CRUD) προς τη βάση για κάποια οντότητα.

Παρακάτω ακολουθεί η απεικόνιση ενός Repository σε Java Interface.

```
public interface IPostRepository extends JpaRepository<Post, Long> {

   @Override
   @Query("SELECT p FROM Post p WHERE p.archived = false")
   List<Post> findAll();

   @Override
   @Query("SELECT p FROM Post p WHERE p.archived = false")
   Page<Post> findAll(Pageable pageable);

   Optional<Post> findByIdAndCreatedByUsername(Long postId, String
username);

   Page<Post> findAllByCreatedBy(AppUser user, Pageable pageable);

   Page<Post> findByJobTypeAndArchivedFalse(JobType jobType, Pageable
pageable);

   Page<Post> findByJobLocationAndArchivedFalse(JobLocation jobLocation,
Pageable pageable);

   Page<Post> findByJobTypeAndJobLocationAndArchivedFalse(JobType jobType,
JobLocation jobLocation, Pageable pageable);

   Page<Post> findBySkillsSkillIdInAndArchivedFalse(Set<Long> skillIds,
Pageable pageable);

   Page<Post> findByJobTypeAndSkillsSkillIdInAndArchivedFalse(JobType
jobType, Set<Long> skillIds, Pageable pageable);

   Page<Post>
findByJobLocationAndSkillsSkillIdInAndArchivedFalse(JobLocation
jobLocation, Set<Long> skillIds, Pageable pageable);
```

```
   Page<Post>
findByJobTypeAndJobLocationAndSkillsSkillIdInAndArchivedFalse(JobType
jobType, JobLocation jobLocation, Set<Long> skillIds, Pageable pageable);
}
```

Συγκεκριμένα βλέπουμε το Repository για ένα entity Post. Αν το interface αυτό ήταν άδειο, τότε θα μπορούσε να εξυπηρετησει αυτόματα όλες τις βασικές ενέργειες προς τη βάση για αυτό το entity. Στο συγκεκριμένο Repository βλέπουμε ότι χρησιμοποιεί πολλές μεθόδους για αναζήτηση διαφόρων post.

Βλέπου ότι έχει γίνει override η βασική μέθοδος findAll, για να φέρει μόνο αποτελέσματα που δεν έχουν διαγραφεί.

Κάποιος μπορεί πολύ εύκολα να παρατηρήσει ένα μοτίβο στην ονομασία όλων των μεθόδων και αυτό δεν είναι τυχαίο.

Η ονομασία των μεθόδων στο repository ακολουθεί τις συμβάσεις ονομασίας μεθόδων JPA του Spring Data που αποτελεί μέρος του Spring Framework. Αυτές οι συμβάσεις επιτρέπουν την αυτόματη δημιουργία SQL query με βάση τα ονόματα των μεθόδων [12].

`findBy[Property][Condition][And/Or][Property][Condition]`

1. `findByIdAndCreatedByUsername(Long postId, String username)`
    - Βρίσκει με βάση το id και το username του δημιουργού
2. `findAllByCreatedBy(AppUser user, Pageable pageable)`
    - Βρίσκει όλα τα post κάποιου χρήστη

…..

Με αυτόν τον τρόπο μπορούμε πολύ εύκολα να γράψουμε queries με πολλές παραμέτρους χώρις πάλι να γράψουμε SQL. Ένα πλεονέκτημα αυτού είναι ότι αυτοί οι μηχανισμοί έχουν ενσωματωμένες δικλείδες ασφαλείας για να αποφύγουμε πιθανές κακόβουλες ενέργειες. Για παράδειγμα SQL injection.

3.3 View

Το View όπως αναφέραμε στην αρχή, εκπροσωπεί τη διεπαφή του χρήστη στην εφαρμογής μας. Αυτή περιλαμβάνει και δυναμικά άλλα και στατικά κομμάτια.

3.3.1 Layout

Ένα βασικό μέρος της δομής της διεπαφής μας, αποτελεί το αρχείο layout.html. Με τη βοήθεια του Thymeleaf, μας παρέχει ένα τρόπο να δομήσουμε σωστά τη διεπαφή μας παρέχοντας και δυναμικό και στατικό περιεχόμενο και τη μείωση του duplicate κώδικα.

Πολύ σημαντικό component μίας σελίδας αποτελεί για παράδειγμα το μενού πλοήγησης μέσα σε αυτή, είναι κάτι που εμφανίζεται σε κάθε σελίδα.

Όταν κάποιος χρήστης ζητήσει για παράδειγμα την αρχική σελίδα, ο server μας δεν θα του δώσει απλα αυτή τη σελίδα.

```
<!DOCTYPE HTML>
<html xmlns:layout="http://www.ultraq.net.nz/thymeleaf/layout"
   layout:decorate="~{layout}">
   <head>
      <title>Home</title>
   </head>
   <body>
      <div layout:fragment="content" class="mt-4">
         <h1 class="mb-4">Welcome to Dev Jobs</h1>
      </div>
   </body>
</html>
```

Με μία πρώτη ματία, ο κώδικας φαίνεται να είναι απλή HTML, άλλα άμα κάποιος κοιτάξει καλύτερα θα δει ότι δεν είναι.

Στο <html> tag βλέπουμε ότι ορίζεται ότι η διάλεκτος της σελίδας πια περιέχει και το Thymeleaf, και ότι αυτή η σελίδα κάνει extend και στο layout.

3.3.2 Reusable Components

Το δεύτερο πράγμα που βλέπουμε είναι ότι υπάρχει ένα που και αυτό δείχνει να κάνει κάτι με το layout. Συγκεκριμένα δημιουργεί ένα ***fragment*** με το ***όνομα content***, και θα ρίξει μέσα στο layout να βρει το αντίστοιχό fragment να το ***αντικαταστήσει***. Αυτό είναι μία πολύ καλή τακτική για να φτιάξουμε επαναχρησιμοποιούμενα κομμάτια κώδικα που μπορεί να χρειαστούν σε πολλά σημεία του κώδικα.

Ένα πολύ συχνό παράδειγμα είναι components για φόρμες. Στον φάκελο fragments υπάρχει ένα αρχείο με το όνομα forms.html. Αυτό είναι ένα αρχείο με πολλών ειδών reusable form components.

Παρακάτω ακολουθεί η φόρμα εγγραφής νέου χρήστη που χρησιμοποιεί αυτα τα resumable components. Έτσι μειώνεται αρκετά ο κώδικας που γράφουμε και έχει ως αποτέλεσμα την παραγωγή πιο ευκολοδιάβαστων αρχείων.

```
<form th:action="${requestUri}" method="post">
   <div class="row form-row">
       <div class="col-md-6">
           <div
th:replace="~{fragments/forms::inputRow(object='registerDto', field='name',
required=true, label='First Name')}"/>
       </div>
       <div class="col-md-6">
           <div
th:replace="~{fragments/forms::inputRow(object='registerDto',
field='surname', required=true, label='Last Name')}"/>
       </div>
   </div>

   <div class="row form-row">
       <div class="col-md-6">
           <div
th:replace="~{fragments/forms::inputRow(object='registerDto',
field='username', required=true, label='Username')}"/>
       </div>
       <div class="col-md-6">
           <div
th:replace="~{fragments/forms::inputRow(object='registerDto',
field='email', required=true, label='Email')}"/>
       </div>
   </div>

   <div class="row form-row">
       <div class="col-md-6">
           <div
th:replace="~{fragments/forms::inputRow(object='registerDto',
field='password', required=true, type='password', label='Create
Password')}"/>
       </div>
       <div class="col-md-6">
           <div
th:replace="~{fragments/forms::inputRow(object='registerDto', field='role',
required=true, type='radio', label='I am registering as a')}"/>
       </div>
   </div>

   <input type="submit" th:value="Register" class="btn btn-primary mt-4
register-button"/>
</form>
```

3.3.3 Security

Η ασφάλειο είναι κάτι που βρίσκεται σε όλα τα σημεία της εφαρμογής. Στη διεπαφή είναι αρκετά σημαντική διότι είναι το το σημείο που αλληλεπιδρά ο χρήστης, που δεν είναι πάντα γνωστές οι προθέσεις του.

Ευτυχώς, το Thymeleaf μας παρέχει εργαλεία για να μας βοηθήσει να αποτρέψουμε πράγματα που ίσως δεν θέλουμε να είναι πάντα εφανή ή τουλάχιστον όχι σε όλους.

Ένα καλό παράδειγμα είναι η πλοήγηση μέσα στη σελίδα. Επειδή έχουμε χρήστες με διαφορετικούς ρόλους και διαφορετικές ενέργειες, θέλουμε να να τους παρέχουμε πρόσβαση μόνο σε ότι χρειάζεται.

Για παράδειγμα ένας χρήστης που έχει τον ρόλο του Developer, δεν μπορεί να έχει πρόσβαση στη σελίδα διαχείρισης των αγγελιών.

```
<li class="nav-item" sec:authorize="isAuthenticated() &&
hasAuthority('RECRUITER')">
   <a class="nav-link" th:href="@{/posts/manage}">My
Posts</a>
</li>
```

Εδώ βλέπουμε να γίνεται έλεγχος με δύο μεθόδους. Η πρώτη μέθοδος `isAuthenticated()` κοιτάζει να δει αν ο χρήστης είναι συνδεδεμένος, και η δεύτερη `hasAuthority('RECRUITER'` κοιτάζει να δει αν έχει τον ρόλο RECRUITER. Αν λοιπόν αυτές οι συνθήκες δεν τηρούνται, τότε αυτή η επιλογή όχι μόνο δεν θα είναι εμφανής στον χρήστη, αλλα ***δεν θα γίνει καν render*** στη σελίδα. Αυτό δεν σημαίνει ότι αν δεν υπάρχει λέγχος και σε άλλα σημεία μέσα στην εφαρμογή, ότι αυτά εδώ είναι επαρκεί για απατρέψουν το χρήστη να κάνει ενέργειες που δεν πρέπει.

Σε αυτό εδώ το σημείο θα ήταν καλό να πούμε και τη διαφορά μεταξύ ***Authentication*** και ***Authorization***. Ένα απλό παράδειγμα είναι το δίπλωμα οδήγησης. Στο μπροστά μέρος, υπάρχουν όλά τα στοιχεία του χρήστη που τον ταυτοποιούν, ότι είναι δηλαδή αυτός που ισχυρίζεται. Στο πίσω μέρος υπάρχει μια λίστα με όλες τις ικανότητες του, δηλαδη αν μπορεί να οδηγήσει αμάξι, μηχανή, φορτηγό κτλ.

Το μπροστά μέρος, η ταυτοποίηση δηλαδή, είναι το ***Authentication***, ενώ το πίσω μέρος, το τι μπορεί να κάνει, είναι το ***Authorization***. Είμαι ο χρήστης Χ και μπορώ να κάνω Υ πράγματα.

Πρώτα γίνεται το authentication και μετά γίνεται το authorization.

3.4 Controller

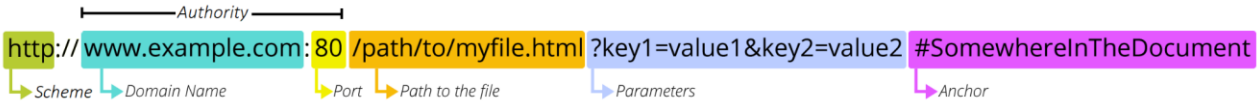
Το Controller όπως αναφέραμε στην αρχή, εκπροσωπεί τη λογική της εφαρμογής μας και αποτελεί τον συνδετικό κρίκο μεταξύ του Model και του View, και αποτελείται από δύο επίπεδα: ***Controllers*** και ***Services***.

3.4.1 Controllers

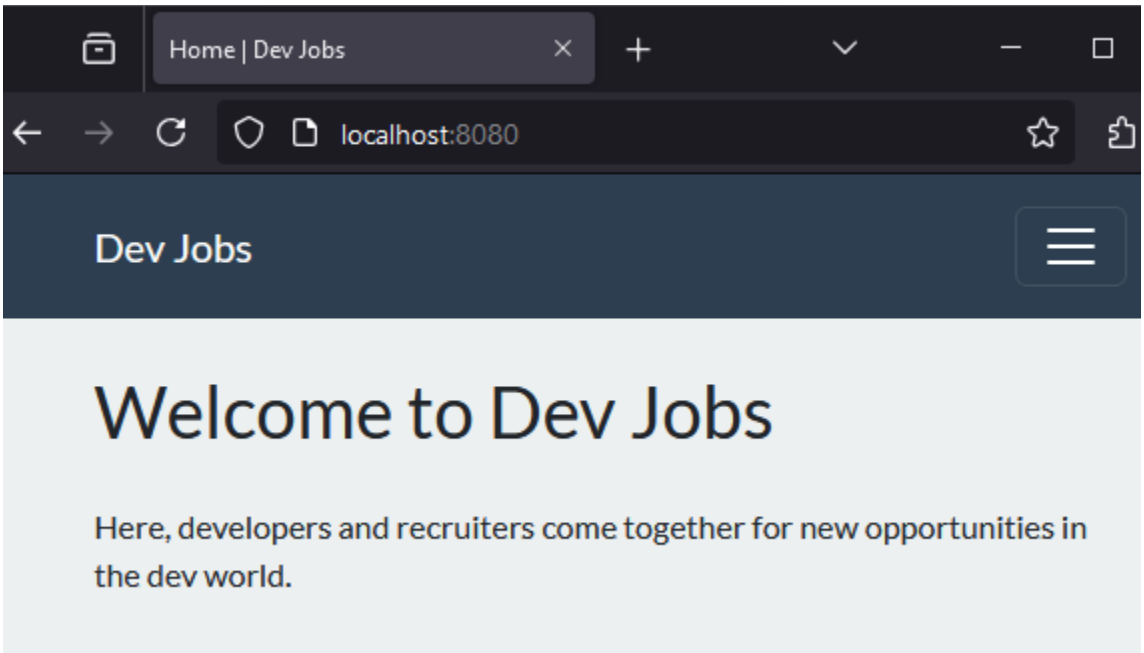
Η κύρια λειτουργία ενος Controller είναι η να δίνει στον χρήστη τις πληροφορίες που του ζητάει, αν πρέπει πάντα. Αυτό πάντα γίνεται με τη χρήση τω ***URL*** (Uniform Resource Locator). Το URL είναι μια τυποποιημένη διεύθυνση που χρησιμοποιείται για τον εντοπισμό και την πρόσβαση σε πόρους στο διαδίκτυο. Είναι ουσιαστικά η διεύθυνση Ιστού που πληκτρολογείτε σε ένα πρόγραμμα περιήγησης (browser) για να επισκεφθείτε μια συγκεκριμένη ιστοσελίδα ή να αποκτήσετε

πρόσβαση σε έναν συγκεκριμένο πόρο στο διαδίκτυο. Στην δικιά μας περίπτωση, επειδή η εφαρμογή μας είναι MVC, κάθε φορά η απάντηση που θα λαμβάνουμε απο τον server θα είναι ένα αρχείο HTML. Η δουλειά λοιπόν του Controller είναι να αντιστοιχεί αυτά τα URL με μεθόδους μέσα στην εφαρμογή μας, για να μπορεί να δώσει στον χρήστη τις πληροφορίες που ζήτησε **[11]**.

Παρακάτω ακολουθεί μια απεικόνιση ενος παραδείγματος URL:



Για παράδειγμα όταν ένας χρήστης πληκτρολογήσει τη διεύθυνση της σελίδας μας στον browser του, εμείς θέλουμε να του δείξουμε την αρχική σελίδα της εφαρμογής μας. Το αποτέλεσμα θα είναι το παρακάτω:



Γιατί όμως; Πώς γίνεται η αντιστοίχησης; Ας δούμε πιο προσεκτικά τη δομή ενος Controller.

Παρακάτω ακολουθεί ένα παράδειγμα Controller:

```
@Controller
@RequestMapping("/")
public class HomeController {

    @GetMapping("/")
    public String index() {

        return "home/index";
    }

    @GetMapping("/about")
    public String about() {

        return "home/about";
    }
}
```

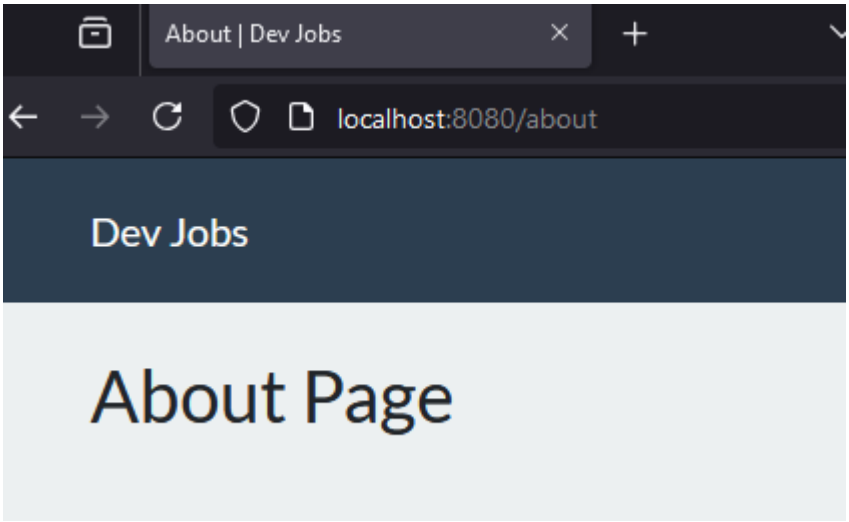
Παραπάνω είναι ο Controller που διαχειρίζεται την αρχική σελίδα της εφαρμογής μας. Βέπου οτι η κλάση HomeController έχεις τους Spring decorators @Controller, όπου της δίνει όλες τις ιδιοτητες ενος MVC Controller και το @RequestMapping(“/”) που της ορίζει ποιο URL να χειρίζεται, όπου στη συγκεκριμένη περίπτωση είναι κενό.

Μετά βλέπουμε δύο μεθόδους, την index και την about, οι οποίες όμως δεν είναι απλές μέθοδοι, γιατί όπως βλέπουμε έχουν και αυτές ένα decocator, τον @GetMapping, όπου η κάθε μια έχει το δικό της ξεχωριστό path. Οπότε για την αρχική σελίδα, αν συνδυάσουμε τα paths του controller και της μεθόδου της σελίδας, όπου για την αρχική σελίδα είναι και τα δύο κενά, τότε θα λάβουμε το μοναδικο URL της αρχικής σελίδας της εφαρμογης μας, που είναι όντως αυτό που είδαμε στην προηγούμενη φωτογραφία.

Και οι δύο μέθοδοι βλεπουμε επιστεύουν ένα String για απάντηση, αλλα εμείς βλέπουμε μία σελίδα. Γιατι;

Αυτό το String που βλεπουμε, αντιπροσωπεύει το path του template μας που αναφέρεται στην προηγούμενη ενότητα. Άρα αφού ξέρει η μέθοδος οτι στο path που επιστεύει, θα βρεί τη σελίδα που πρέπει να δείξει στον χρήστη. Στον φάκελο home λοιπον θα βρεί τις σελίδες index και about αντίστοιχα.

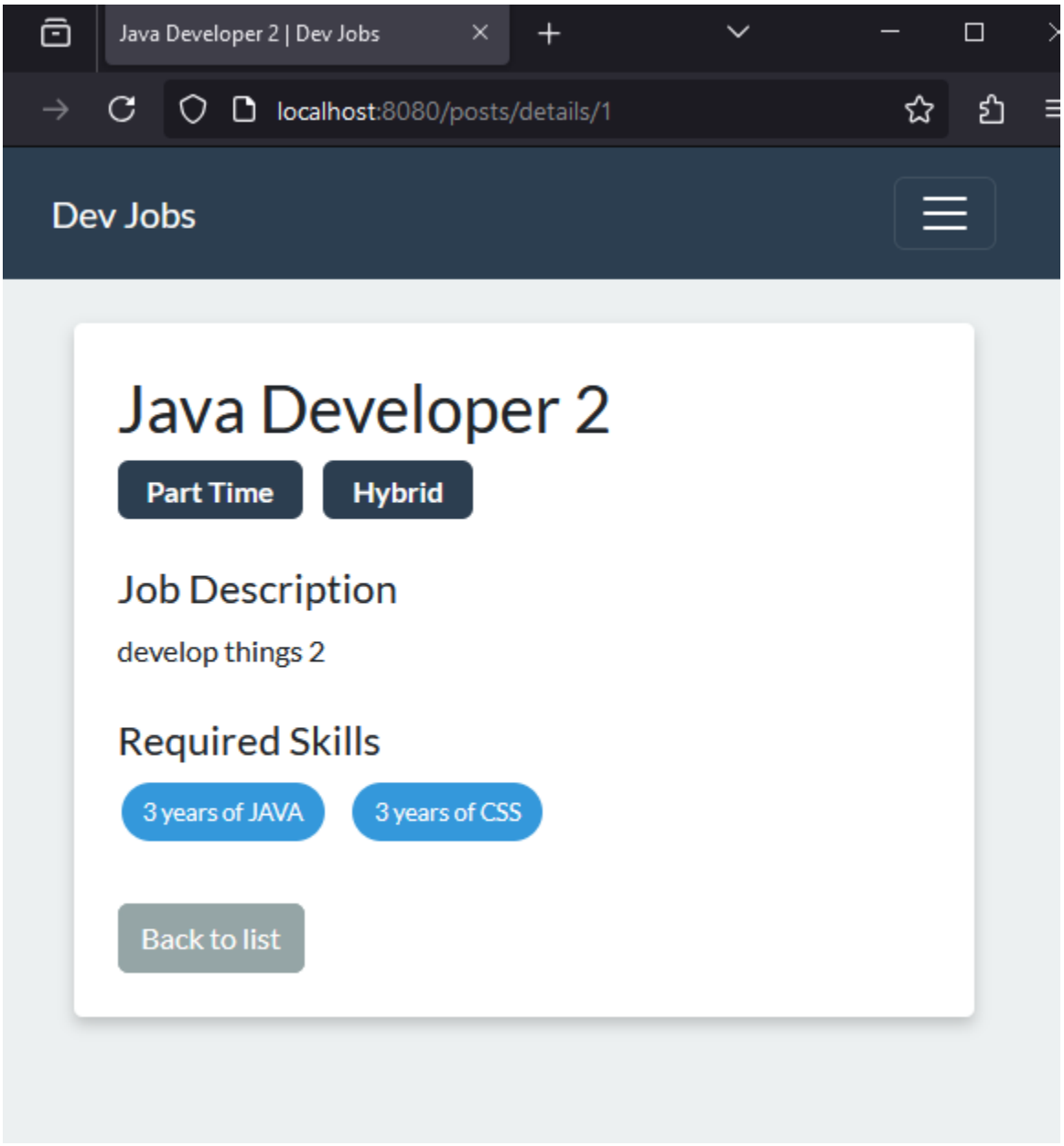
Για να δούμε λοιπόν τη σελίδα about, θα πρέπει να συνδυάσουμε το path του controller και της μεθόδου, δηλαδή κενο και “/about”. Για να δούμε:



Όντως έγινε αυτό που περιμέναμε.

Ας δούμε ένα πιο σύνθετο παράδειγμα, όπου θα μας βοηθήσει να μεταβούμε και στην επόμενη ενότητα του κεφαλαίου.

Παρακάτω ακολουθεί μία φωτογραφία με το URL για τις λεπτομέρειες μιας αγγελίας



Ας δούμε με πιο πολύ λεπτομέρεια τον PostController.

```
@Controller  perpiris
@RequestMapping("/posts")
public class PostController {

    private final IPostService postSe
```

```
@GetMapping("/details/{id}")  perpiris
public String details(@PathVariable("id") long postId, Model model, Principal principal) {

    PostDto post = postService.get(postId);
    model.addAttribute( attributeName: "post", post);
    model.addAttribute( attributeName: "userHasApplied", applicationService.hasUserAppliedToPost(postId));

    if (principal != null) {
        var userSkills = userService.getUserSkills(principal.getName());
        model.addAttribute( attributeName: "userSkills", userSkills);
    } else {
        model.addAttribute( attributeName: "userSkills", java.util.Collections.emptySet());
    }

    return "post/details";
}
```

Απο το URL που έχουμε λοιπον βρήκαμε και τον Controller άλλα και το Action που αντιπροσωπεύει. Σε αυτό το παράδειγμα βλέπουμε και τη χρήση παραμέτρων στο URL. όπου στη συγκεκριμένη περίπτωση, ζητήσαμε να δούμε τις πληροφορίες της αγγελίας με το id=1.

Οι Controllers μπορούν να παρέχουν και αυτοί ένα στρώμα ασφαλείας, πολύ όμοιο με αυτό που είδαμε και στο Thymeleaf προηγουμένως. Μπορούμε με decorators να χρησιμοποιήσουμε τις ίδιες μεθόδους και σε κάθε Controller άλλα και σε κάθε action ξεχωριστά.

Για παράδειγμα από τη λογική της εφαρμογής μας, ξέρουμε ότι μόνο ένας χρήστης με ρόλο RECRUITER μπορεί να δημιουργήσει μια αγγελία. Αυτό φαίνεται κάπως έτσι:

```
    @PreAuthorize("hasAuthority('RECRUITER')")
    @GetMapping("/create")
    public String create(@ModelAttribute("post")
PostDto postDto) {

        return "post/create";
    }

    @PreAuthorize("hasAuthority('RECRUITER')")
    @PostMapping("/create")
    public String create(@ModelAttribute("post")
@Valid PostDto postDto, BindingResult bindingResult,
RedirectAttributes redirectAttributes, Model model)
{

        if (bindingResult.hasErrors()) {
            model.addAttribute("skills",
postDto.getSkills());
            return "post/create";
        }

        var newPostId = postService.create(postDto);

redirectAttributes.addFlashAttribute("MSG_SUCCESS",
"Post created successfully.");

        return "redirect:/posts/details/" +
newPostId;
    }
```

Με τον decorator `@PreAuthorize("hasAuthority('RECRUITER')")` μπορούμε να εξασφαλίσουμε ότι η πρόσβαση στα action της δημιουργίας μιας αγγελίας μπορούν να έχουν μόνο οι RECRUITERS.

Για παράδειγμα, απο τη λογική της εφαρμογής μας ξέρουμε ότι μόνο χρήστης με ρόλο DEVELOPER μπορούν να κάνουν αίτηση σε μία αγγελία. Άρα ένας χρήστη πρέπει πρώτα να έχει κάνει authenticate και μετά να έχει περάσει και το authorization με τον ρόλο του. Ένα απλό τρικ για να δούμε ότι ένας χρήστης έχει γίνει authenticate,

3.4.2 Services

Το Service είναι το επίπεδο που που αναλαμβάνει τη «καθαρίσει» τα δεδομένα αμφίδρομα, δηλαδή και απο τον χρήστη προς τη βάση αλλά και από τη βάση προς τον χρήστη. Πρακτικά θα μπορούσε να μήν υπάρχει, και όλα αυτά να γίνονται μέσα στον Controller, αλλα έτσι χάνουμε το πλεονέκτημα του Separation of Concerns άλλα και του teasing, όπου δεν θα καλύψουμε σε αυτή την εργασία. Μία άλλη ονομασία αυτού του επιπέδου είναι και ***Business Logic***.

Αυτή η λογική μπορεί να είναι απλή, όπως για παράδειγμα όταν δημιουργείται μία νέα αγγελία, να πρέπει να έχει απαραίτητα κάποια πεδία συμπληρωμένα. Μπορεί να είναι και πιο περίπλοκο, όπως όταν γίνει επεξεργασία σε κάποια αγγελία, να μπορεί να την κάνει μόνο αυτός που την δημιούργησε αρχικά. Αν θυμάστε και στο Model, είχαμε θέσει τέτοιος κανόνες

Ένα σημαντικό καθήκον του Service, είναι όταν δεν τηρούνται αυτοί οι κανόνες να μπορεί να ειδοποιήσει τον χρήστη με βοηθητικά μηνύματα.

Ας δούμε αναλυτικά τη μέθοδο τροποποίησης μιας αγγελίας:

```
public void edit(final Long postId, final PostDto
postDto) {
    final Post post = postRepository.findById(postId)
            .orElseThrow(() -> new
NotFoundException("Post not found with ID: " + postId));

    if (!isOwner(postId)) {
        throw new UnauthorizedException("Only the creator
can edit this post");
    }

    post.setTitle(postDto.getTitle());
    post.setDescription(postDto.getDescription());
    post.setJobType(postDto.getJobType());
    post.setJobLocation(postDto.getJobLocation());

    post.getSkills().clear();

    postDto.getSkills().forEach(skillDto -> {
        Skill skill =
getOrCreateSkill(skillDto.getDescription().toUpperCase()
);
        PostSkill postSkill = PostSkill.builder()
                .post(post)
                .skill(skill)
                .years(skillDto.getYears())
                .build();
        post.getSkills().add(postSkill);
    });

    postRepository.save(post);
}
```

Ο πρώτος έλεγχος που κάνει είναι να δει αν όντως υπάρχει η αγγελία που πάμε να τροποποιήσουμε. Είναι πολύ σημαντικό να ελέγχουμε όλες τις παραμετρους στις μεθόδους. Μετά ο δεύτερος έλεγχος που κάνει είναι να ελέγξει αν όντως ο χρήστης που πάει να την τροποποιήσει είναι όντως ο δημιουργός της. Και στις δύο περιπτώσεις, αν δεν τηρείται ο κανόνας, η ενέργεια δεν θα γίνει και ο χρηστης θα ειδοποιηθεί με το ανάλογο μήνυμα. Μετά από αυτούς του ελέγχους θα μπορούσαμε να ελέγξουμε 1-1 τα πεδία που απαιτούνται να δούμε αν είναι συμπληρωμένα, άλλα υπάρχει ένας πιο γρήγορος τρόπος και αυτός είναι μέσα στα ***Dto*** (αμέσως μετά θα δούμε τι ακριβώς είναι το Dto).

Παρακάτω ακολουθεί η δομή του PostDto

```
@Getter
@Setter
public class PostDto {

    private Long id;

    @NotEmpty(message = "This field is required.")
    @Size(max = 255)
    private String title;

    @NotEmpty(message = "This field is required.")
    private String description;

    @NotNull(message = "This field is required.")
    private JobType jobType;

    @NotNull(message = "This field is required.")
    private JobLocation jobLocation;
```

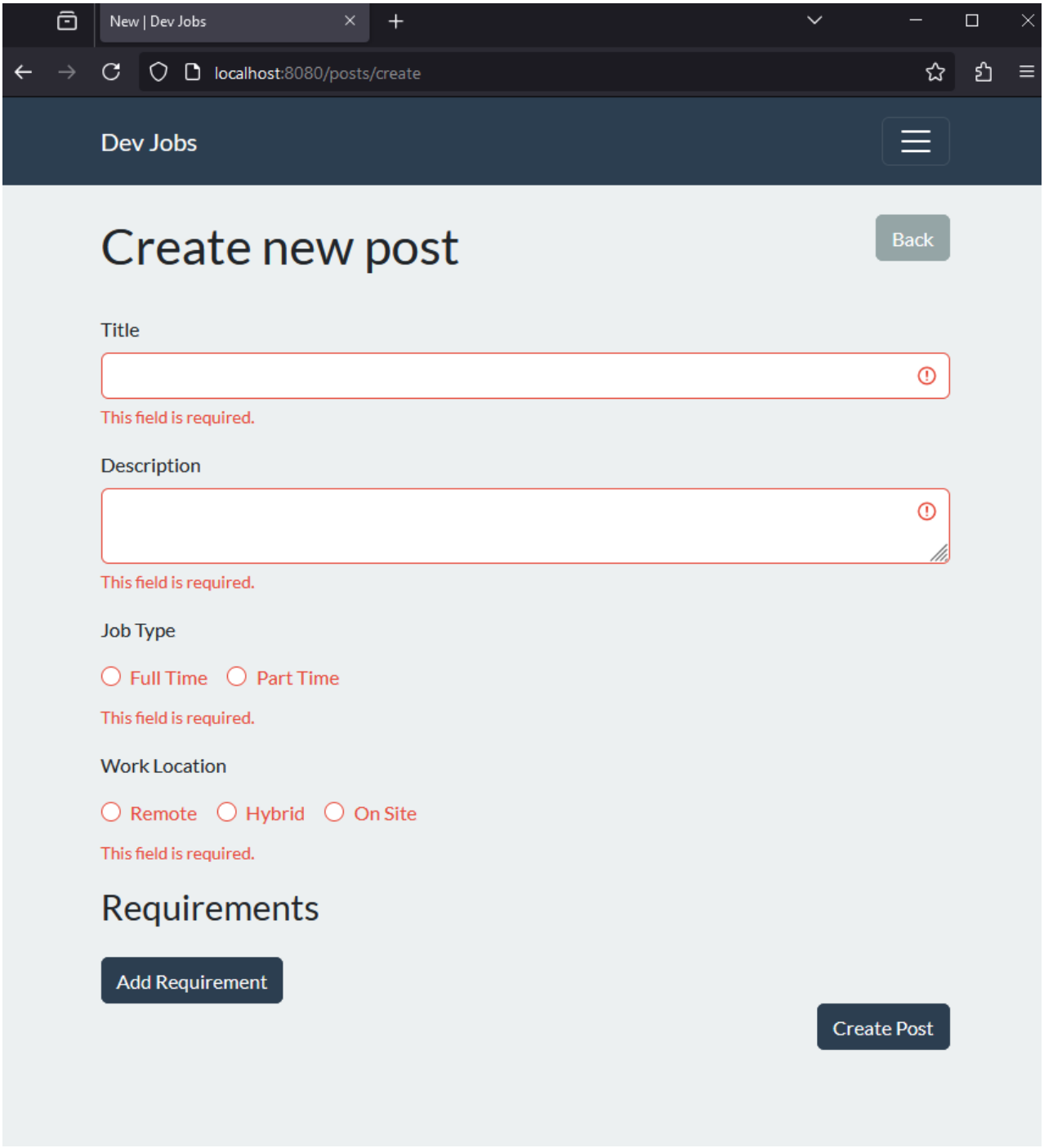
```
    private String createdOn;

    private long applicationCount;

    private boolean acceptingApplications;

    private boolean archived;

    private List<PostSkillDto> skills = new
ArrayList<>();
}
```

Βλέπουμε ότι κάποια πεδία όπως τον τίτλο και την περιγραφή, δεν μπορούν να είναι κενά γι αυτό έχουν τον decorator

```
@NotNull(message = "This field is required.")
```

Αυτό σημαίνει ότι δεν μπορούν να είναι κενά και αν είναι θα εμφανίσει το μήνυμα «This field is required».

Παρακάτω ακολουθεί μία φωτογραφία με την αποστολή κενής φόρμας προσθήκης αγγελίας.



3.4.3 Dtos

Τα DTO (Data Transfer Object) χρησιμεύουν ως ελαφροί φορείς δεδομένων μεταξύ των επιπέδων εφαρμογής, ενσωματώνοντας μόνο βασικές πληροφορίες χωρίς επιχειρηματική λογική. Προωθούν την αποτελεσματική σειριοποίηση για μεταφορά δικτύου, αποσυνδέουν την αναπαράσταση δεδομένων από τα μοντέλα και μπορούν να προσαρμοστούν στις συγκεκριμένες ανάγκες του πελάτη, ενισχύοντας τελικά την απόδοση και την ευελιξία της εφαρμογής.

Τα DTO είναι μεγάλο κομμάτι σε REST εφαρμογές γιατί έχουν περισσότερη χρησιμότητα από ότι σε μια MVC εφαρμογή. Αποφεύγουμε να δείχνουμε απευθείας την αναπαράσταση των δεδομένων μας στον χρήστη όπως είναι στη βάση, κυρίως για λόγους ασφαλείας.

Για μία αγγελία θα μπορούσαμε να έχουμε ένα DTO για κάθε διαφορετική χρήση. Πχ ένα που έχει τις πληροφορίες που δείχνει η σελίδα με τα details, ένα άλλο με τις πληροφορίες που χρειάζεται για την δημιουργία, άλλο για την επεξεργασία κτλ.

Άρα τα DTOS είναι για τον χρήστη και τα Entities για τη βάση. Η μέθοδος μετατροπής από το ένα είδος στο άλλο λέγεται mapping.

Παρακάτω ακολουθεί παράδειγμα μεθόδου mapping, απο DTO σε Entity:

```
private void mapToEntity(final PostDto postDto,
final Post post) {
   post.setTitle(postDto.getTitle());
   post.setDescription(postDto.getDescription());
   post.setJobType(postDto.getJobType());
   post.setJobLocation(postDto.getJobLocation());

post.setAcceptingApplications(postDto.isAcceptingApp
lications());
   post.setArchived(postDto.isArchived());

   if (post.getSkills() == null) {
       post.setSkills(new ArrayList<>());
   } else {
       post.getSkills().clear();
   }

   postDto.getSkills().forEach(skillDto -> {
       PostSkill postSkill = new PostSkill();
       postSkill.setYears(skillDto.getYears());
       post.getSkills().add(postSkill);
   });
}
```

3.4.5 Security

Το Spring Security μας παρέχει πολλά εργαλεία για την ασφάλεια. Το κυριότερο είναι το ***Session***. Το Session αναφέρεται σε έναν τρόπο διατήρησης πληροφοριών κατάστασης (state management) και συγκεκριμένου χρήστη σε πολλαπλά αιτήματα HTTP. Ακολουθεί μια συνοπτική επισκόπηση των περιόδων σύνδεσης στο Spring Web Security:

Session

Session, είναι ένας μηχανισμός αποθήκευσης από την πλευρά ***του διακομιστή*** (server) που διατηρεί δεδομένα ειδικά για το χρήστη σε πολλαπλά αιτήματα HTTP. Επιτρέπει στην εφαρμογή να θυμάται τους πιστοποιημένους χρήστες (authenticated users) και το περιβάλλον τους σε όλη την αλληλεπίδρασή τους με την εφαρμογή.

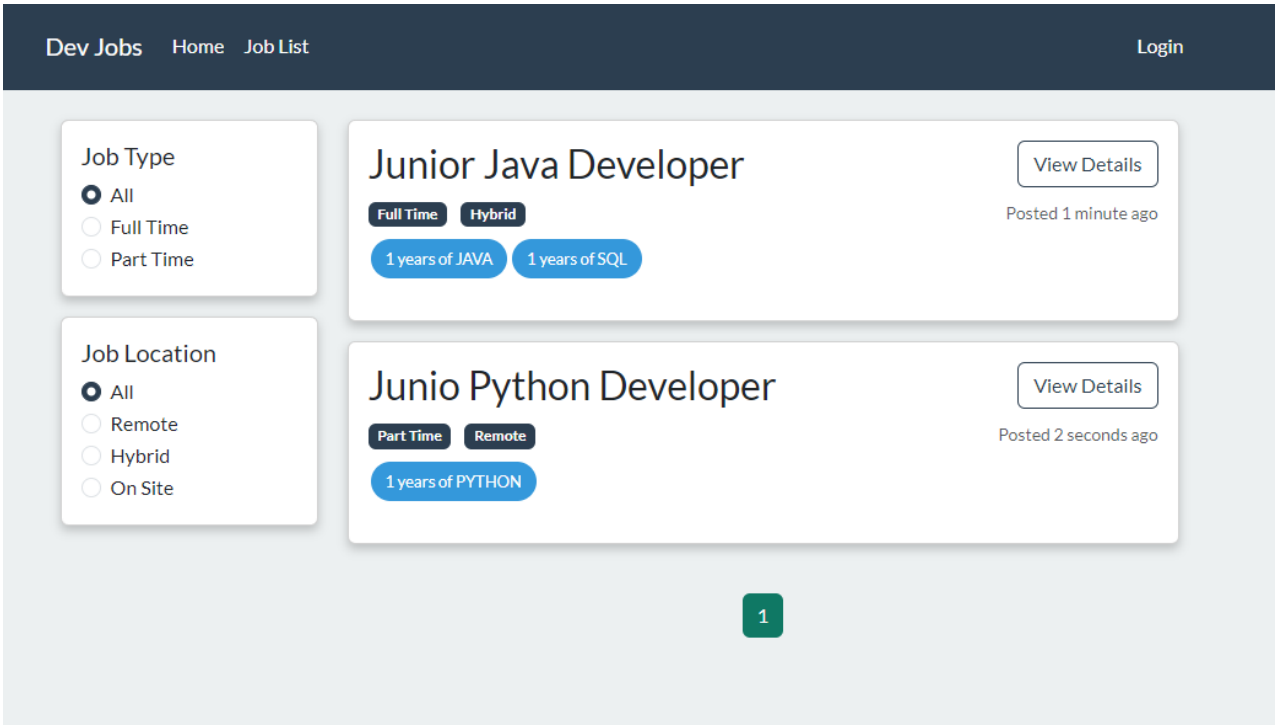
Cookie:

Cookie, είναι ένα μικρό κομμάτι δεδομένων που αποθηκεύεται στο πρόγραμμα περιήγησης (browser) ***του χρήστη*** (client), που συνήθως περιέχει ένα αναγνωριστικό περιόδου σύνδεσης (session identifier). Επιτρέπει στον διακομιστή να αναγνωρίζει και να συσχετίζει τα εισερχόμενα αιτήματα με την κατάλληλη περίοδο λειτουργίας χρήστη, διευκολύνοντας την επικοινωνία κατάστασης στο πρωτόκολλο HTTP χωρίς κατάσταση (stateless) [13].

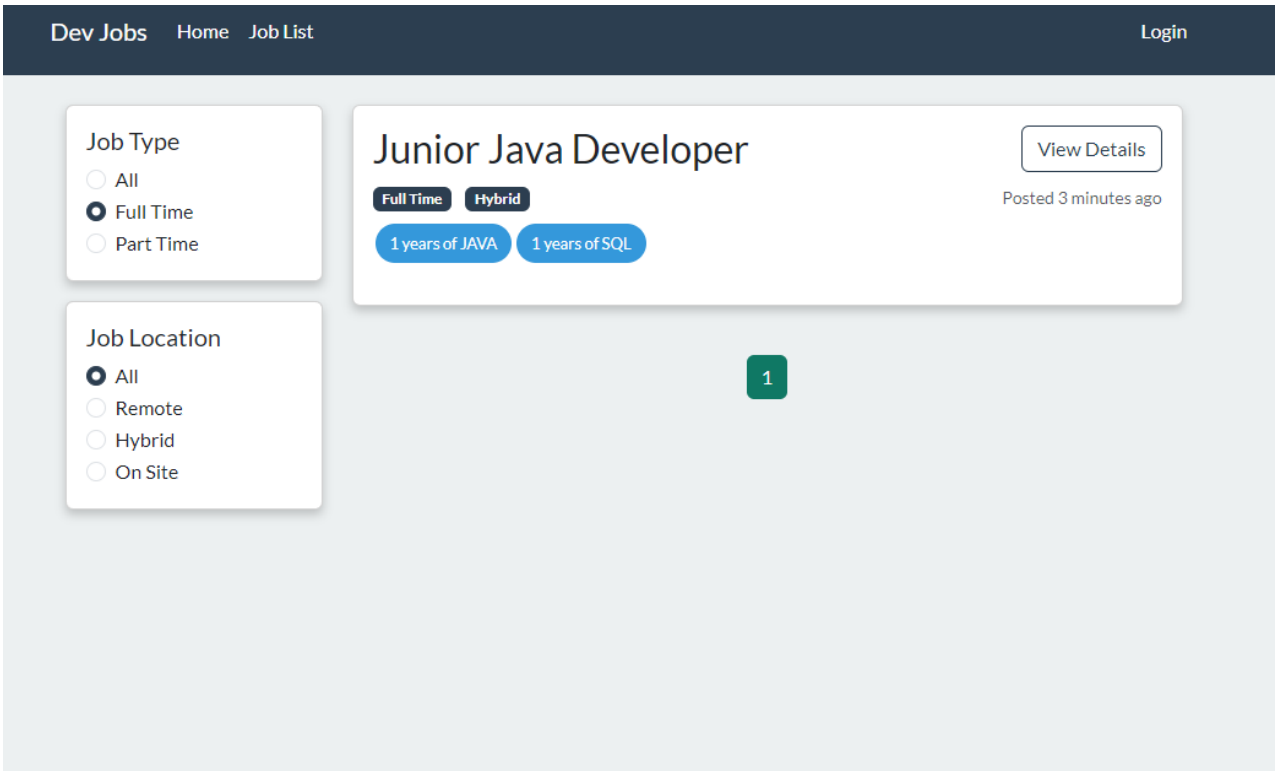
4. Λειτουργία Εφαρμογής

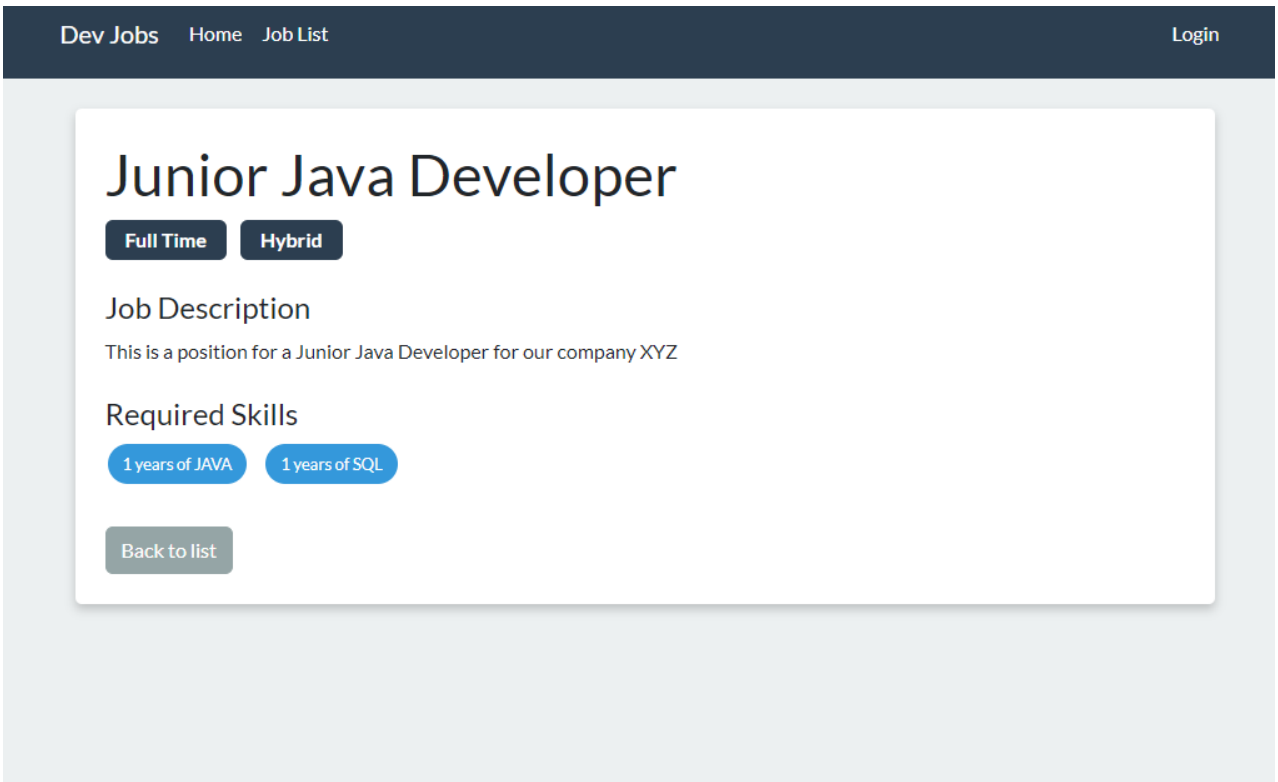
4.1 Αναζήτηση και προβολή

Όλοι οι χρήστες, ακόμα και αν δεν έχουν κάνει εγγραφή, μπορούν να κάνουν αναζήτηση και να δουν πληροφορίες για μία θέση.



Δεξιά βλέπουμε πως παρέχονται και φίλτρα

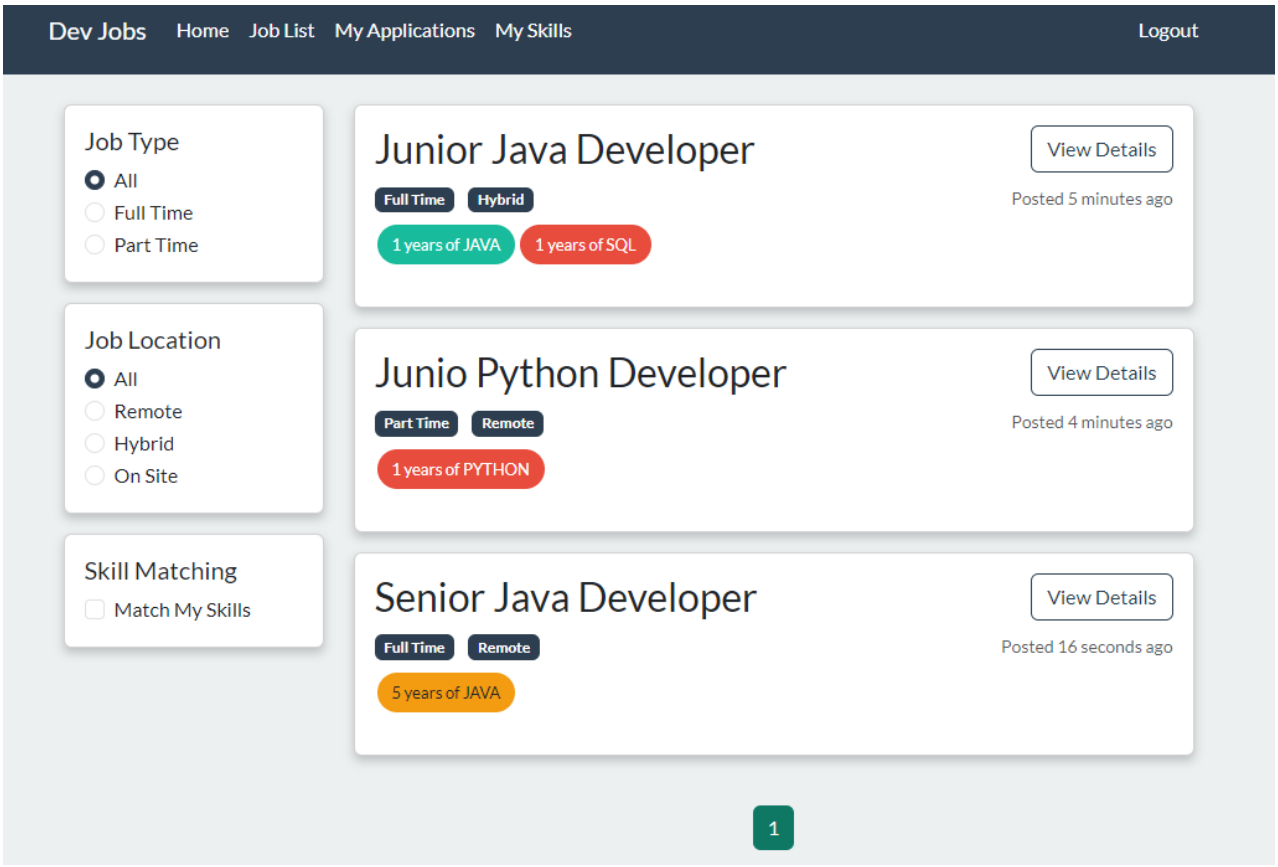




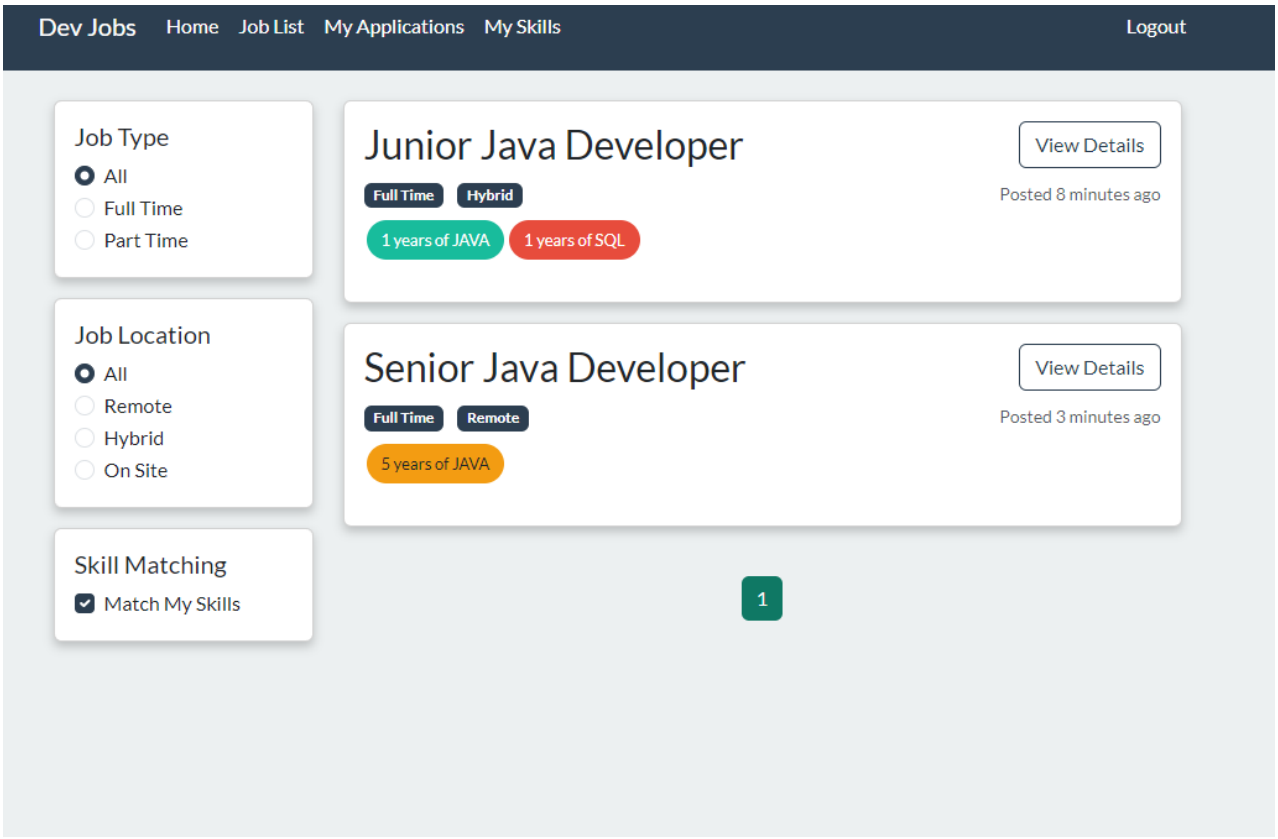
Αν ο χρήστης είναι συνδεδεμένος και είναι Developer, τότε μπορεί να δει γρήγορα αν έχει τα skills για κάποια δουλειά με βάση το χρώμα

τους. Αν έχει και το skill και τα χρόνια τοτε τα βλέπει πράσινα, αν έχει το skill αλλα όχι τα χρόνια τότε πορτοκαλί αλλιως κόκκινα.

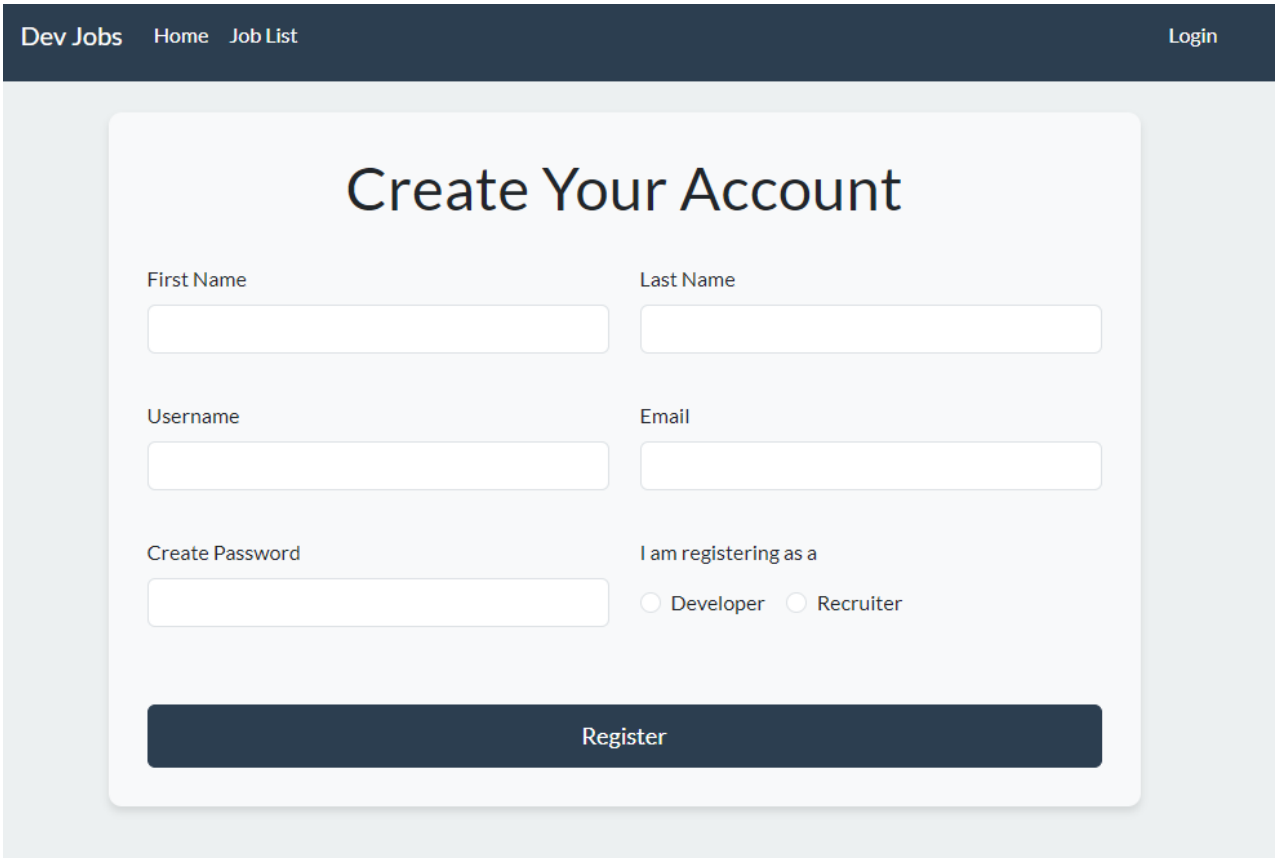
Επίσης έχει τη δυνατότητα να αναζητήσει δουλείες μόνο με τα skills του.

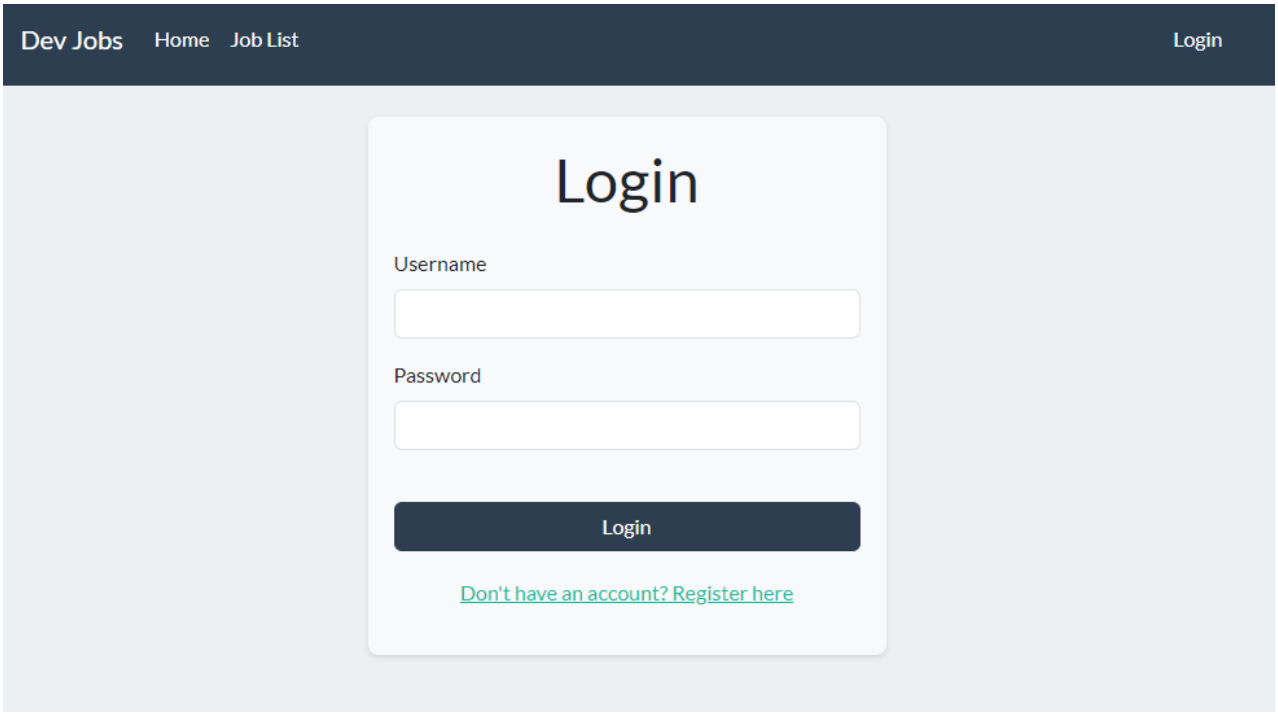


Ο χρήστης μας σαν skills έχει μόνο Java

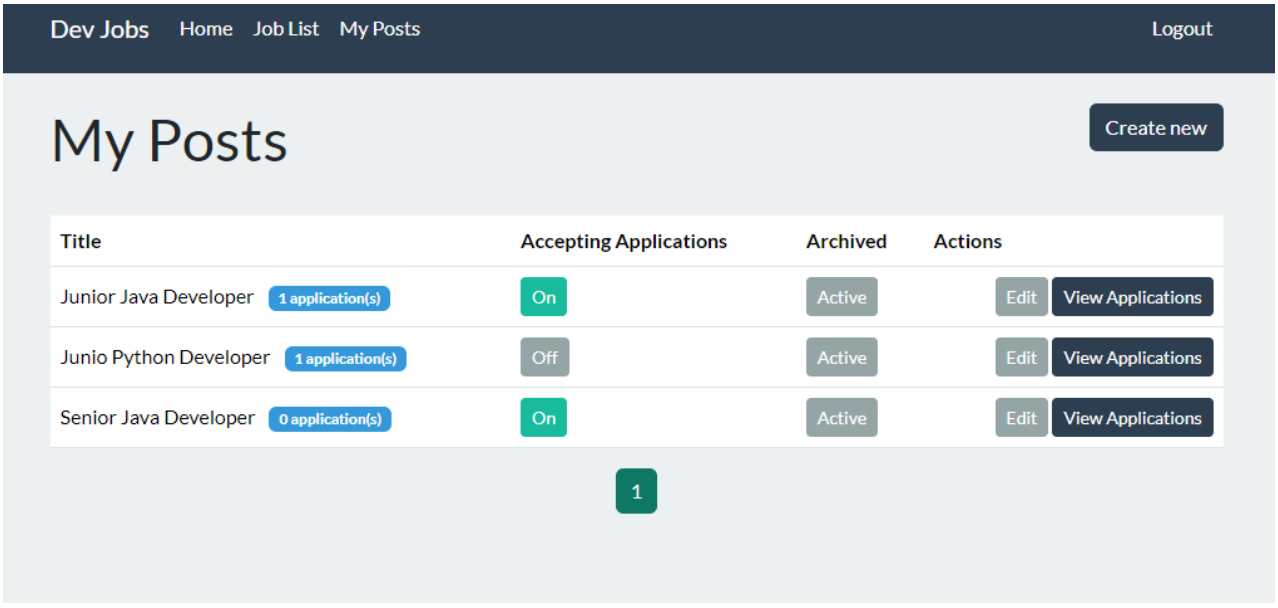

Dev Jobs
Home
Job List
My Applications
My Skills
Logout
Job Type
All
Full Time
Part Time
Job Location
All
Remote
Hybrid
On Site
Skill Matching
Match My Skills
Junior Java Developer
View Details
Full Time
Hybrid
Posted 8 minutes ago
1 years of JAVA
1 years of SQL
Senior Java Developer
View Details
Full Time
Remote
Posted 3 minutes ago
5 years of JAVA
1

4.2 Εγγραφή και σύνδεση

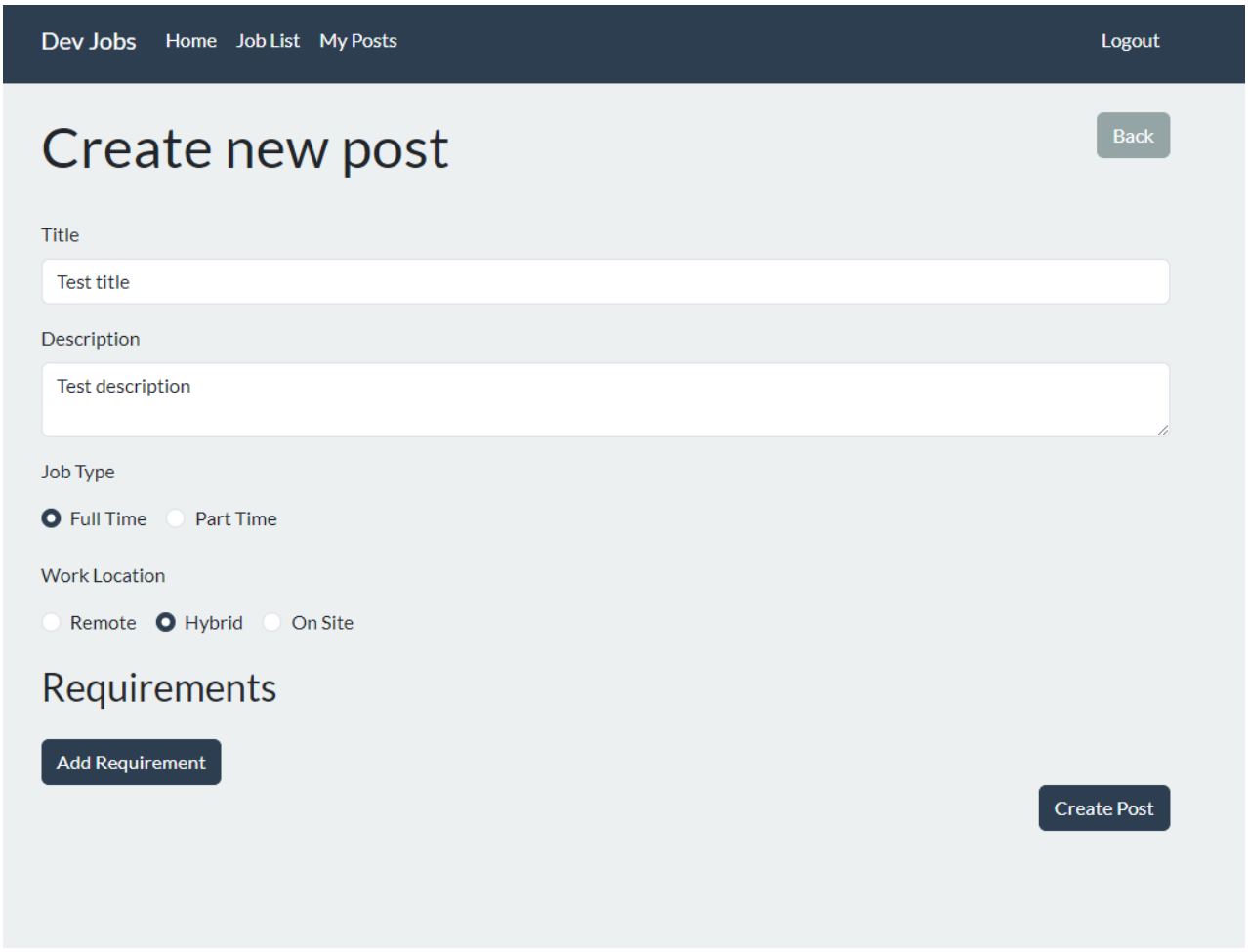


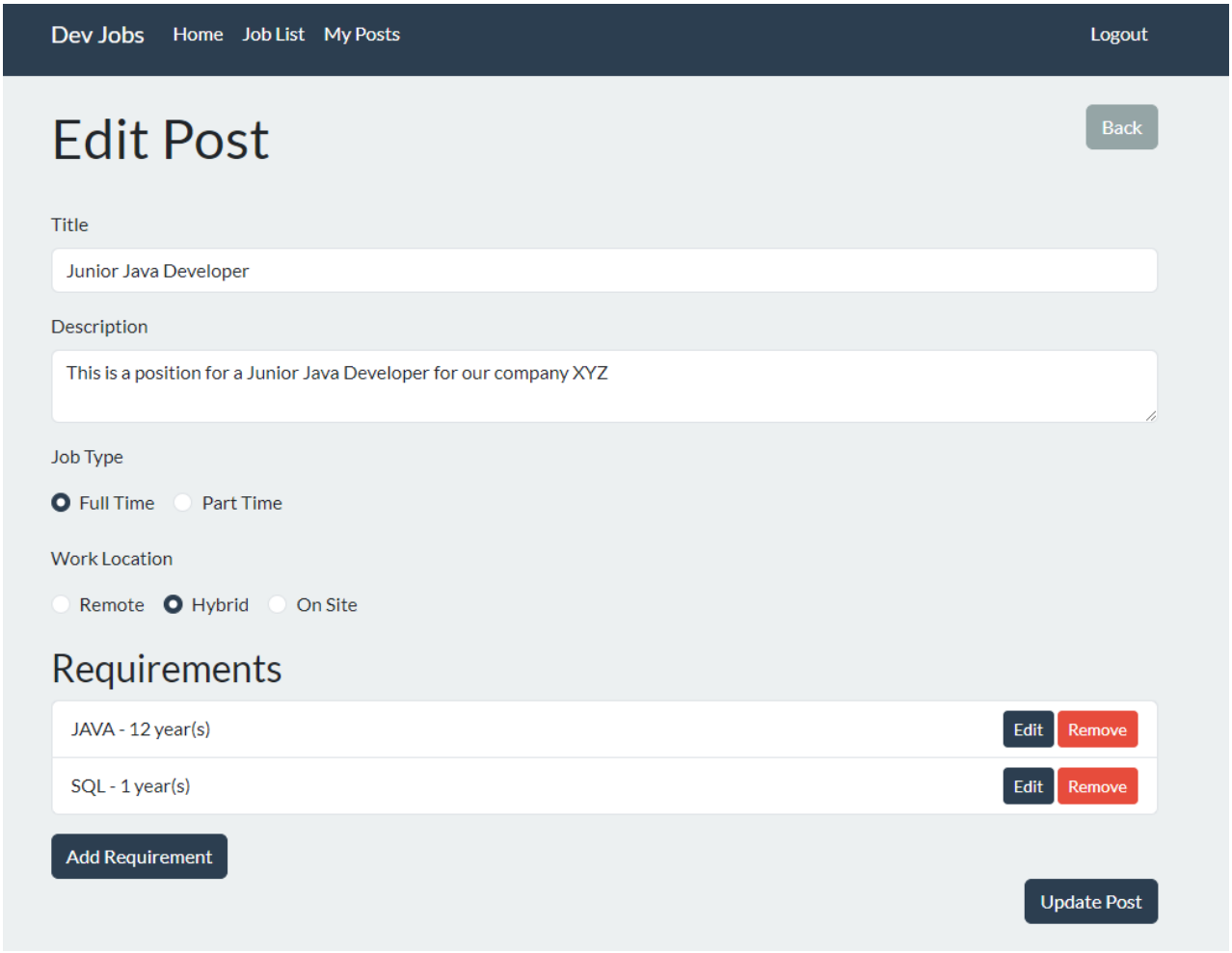


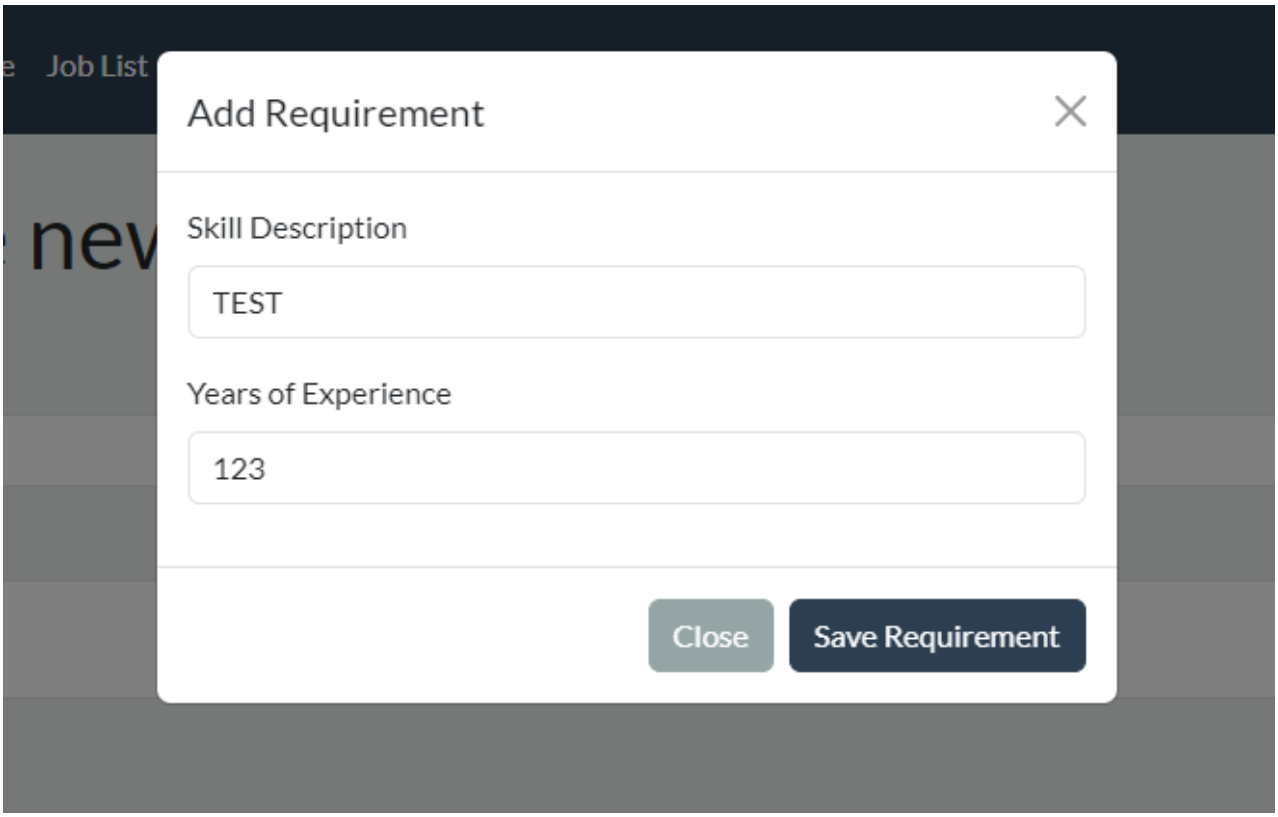
4.3 Διαχείριση Αγγελιών



Ο Recruiter μπορεί να διαχειρίζεται από δικό του μενού όλες τους τις αγγελίες. Δημιουργία, Επεξεργασία, “Διαγραφή”, να μπορεί να επιτρέπει και να σταματά την ιδιότητα να μπορεί κάποιος να κάνει αίτηση, και να διαχειρίζεται τις αιτήσεις




Dev Jobs
Home
Job List
My Posts
Logout
Edit Post
Back
Title
Junior Java Developer
Description
This is a position for a Junior Java Developer for our company XYZ
Job Type
Full Time
Part Time
Work Location
Remote
Hybrid
On Site
Requirements
JAVA - 12 year(s)
Edit
Remove
SQL - 1 year(s)
Edit
Remove
Add Requirement
Update Post


Job List
Add Requirement
Skill Description
TEST
Years of Experience
123
Close
Save Requirement


Dev Jobs
Home
Job List
My Posts
Logout
Applications for Junio Python Developer
Back
dev dev - dev
PENDING
Applied on: 13-08-2024 00:19
Skills:
JAVA (12 years)
Approve
Reject
Back to Posts

5. Σύνοψη

5.1 Συμπεράσματα

Μιλώντας εκ του αποτελέσμας, συμπεραίνουμε πως ό,τι και αν είναι αυτό που φτιάχνουμε, απλό, περίπλοκο, με λίγες ή πολλές δυνατότητες, αν δεν έχουμε θέσει από την αρχή ένα ουσιώδες καλούπι και κάποια σαφή santards, θα καταλήξουμε να κάνουμε τη ζωή μας δύσκολη. Σίγουρα δεν μπορούμε να τα σχεδιάσουμε όλα από την αρχή, σίγουρα κάτι θα πάει στραβά. Σε αυτή την εργασία μπορεί να χρησιμοποιούμε εργαλεία όπως τη Java και το Spring Framework για να φτιάξουμε μία MVC εφαρμογή. Αυτό δεν σημαίνει ότι δεν μπορούμε να κάνουμε το ίδιο και με τη C# μαζί με το .Net Framework. Οι φιλοσοφία είναι η ίδια.

Φτιάξαμε μια δυναμική web εφαρμογή.

5.2 Μελλοντικές επεκτάσεις

Μία πολύ πιθανή μελλοντική επέκταση της εφαρμογής θα ήταν η ενσωμάτωση LLMs (Large Language Model) στην εφαρμογή μας που θα μπορούσαν να βοηθήσουν και τα δύο είδη χρηστών, τους developers στην αναζήτηση και τους recruiters στο evaluation των αιτήσεων.

6. Βιβλιογραφία

1. Τί είναι framework και τα πλεονεκτήματά του - Spring Start Here, Laurentiu Spilca
2. Java
3. Maven
4. Spring Framework, Spring in Action, sixth edition - Craig Walls
5. Lombok & Lombok
6. Bootstrap
7. Thymeleaf
8. Τί είναι Postgres και τα πλεονεκτήματα της - Mastering Postgres, Frahaan Hussain, Kameron Hussain
9. HTML - CSS - Javascript
10. Τί είναι MVC και τα πλεονεκτήματα του MVC
11. URL
12. Spring JPA
13. Spring Session and Cookie