**ТЕХНОЛОГИЧНО УЧИЛИЩЕ ЕЛЕКТРОННИ СИСТЕМИ**

**към ТЕХНИЧЕСКИ УНИВЕРСИТЕТ - СОФИЯ**

**ДИПЛОМНА РАБОТА**

Тема: Онлайн портал за намиране и предлагане на почасова работа,

базирано на Spring MVC.

Дипломант: Научен ръководител:

*/Петър Димитров Михайлов/ /инж. Динко Иванов/*

СОФИЯ

2019

## **Увод**

В днешно време на различни бизнес компании и корпоративни организации е трудно да се намери временна работа за изкарване на достатъчно пари почасово. Затова и идеята на този уеб базиран портал е именно да се улеснят взаимоотношенията на работодател със служител като се използва уеб среда и онлайн комуникация. По този начин се цели да се позволи на хората с недостатъчна квалификация да бъдат лесно ангажирани на непълно работно време и съответно да получат възнаграждение за свършената работа. Още повече те сами ще могат да пуснат обява като рекламират своите умения , а по този начин ще улеснят своя работодател. Приложението пести време и свързва работодателите с кандидатите на базат на взаимните интереси, опит и умения като се стреми да повиши равнището на заетост. Основният проблем на пазара в момента е изключително ниската ефективност на услугите и за двете странни – за търсещите работа и за работодателите. Процесът и за компаниите, и за търсещите е тромав и изисква много време. Проблем има и пред работодателите, които провеждат интервюта с кандидати, които не са добре информирани за позицията. Целева група, която проекта се стреми да достигне, са лицата между 16 - 35 години, които търсят средства за да развият своето хоби, таланат или допълнителни умения и знания в дадена сфера. С повишаващата се сложност на корпоративните приложенията, подръжката и гъвкавостта се превръща в доста трудуемка и скъпа задача. Затова Model-View-Controller (MVC) архитектурата става ефикастно решение на тези проблеми за проекта като ясно се разграничават слоевете за презентация, бизнес логика и съхранение на данни. Така и Spring MVC става популярна за разработката на качествени уеб приложения. Използва Hibernate като мощна и високопроизводителна обектно-релационна услуга, кoято ни служи да извлечем информация от базата данни. Приложенията, разработени с помощта на Spring MVC и Hibernate, са пълни, модулни и лесни за използване. В този проект се реализира система за намиране и предлагане на почасова работа базирана именно на тази архитектура.

## **ПЪРВА ГЛАВА**

## **МЕТОДИ И ТЕХНОЛОГИИ ЗА РЕАЛИЗИРАНЕ НА WEB ПРИЛОЖЕНИЯ**

## **Основни принципи, технологии и развойни среди за реализиране на web приложения**

## **Развойни средства и среди**

## **IntelliJ IDEA**

IntelliJ IDEA (2018.2.5) е мощна интегрирана среда за разработка (IDE) на софтуерни проложения за Windows, Mac OS X, GNOME или KDE в Java. Създадена е от JetBrains (известна преди като IntelliJ), и е достъпна като Apache 2 Licensed community edition, и в патентованото commercial edition, могат да се изпозлват за търговски разработки. Писането с текстов редактор, компилирането и изпълнението на една програма от командния ред е сравнително проста работа, но за да направим това за по-голям проект, може да се окаже сложно и трудоемко занимание. За намаляване на сложността, улесняването на писането, компилирането и изпълнението на софтуерни проложения чрез един единствен инструмент съществуват визуални (Integrated Development Environment, IDE). Средите за разработка най-често предлагат множество допълнения към основните функции за разработка, като например дебъгване, изпълнение на unit тестове, проверка за често срещани грешки, достъп до хранилище за контрол на версиите и други. IntelliJ IDEA е достъпна в Community версията с отворен код, безплатно като използването се регулира от Apache 2.0 License. IntelliJ IDEA Community Edition е идеален избор за тези, които работят върху чисти Java/Groovy приложения. Разполага с различни редакции и кодови инспекции, помощни кратки пътища до различни функции, автоматично отсраняване на грешки и тестване, CVS, Subversion, подръжка на Git, а както и на Ant и Maven. Редакторът предоставя възможността да се работи с външни библиотеки, които да са дефинират глобални (global library) или модулни (module library) като и улеснява разработката на софтуер с някои от основните си функции.

* Автоматично зареждане на последната сесия на работа с проекта. Запазва последнитед файлове по начина по-които са оставени.
* IDEA индексира целия проект като дори и построява синтактично дървоидна структура. Благодарение на това където и да поставим курсора средата знае къде е и какво може да бъде направено.
* Дебъгването не се извършва със селектиране на целия ред както при Eclipse примерно, което IDE ще разгледаме по-къснo.

## **Eclipse**

**Eclipse** е многоезична среда за разработване на софтуер, която включва интегрирана среда за разработка (IDE) и плъгин система. Написана е главно на Java и може да бъде използвана за разработване на приложения на Java. Eclipse се справя с интеграцията много лесно. От взаимодействие със Selenium, до интеграцията на Sikuli. Използва различни плъгини или допълнителните комплекти, чиято инсталация е толкова проста, колкото да изберете „Помощ“ и „Инсталиране на нов софтуер“. Понякога Eclipse изисква рестартиране. Понякога това може да бъде тромаво при актуализиране на софтуера. Бих искал да инсталирам или актуализирам своя софтуер, след това да продължа да пиша моя код и да стартирам тестовете си. Необходимостта да рестартирате е малък проблем, но все пак е проблем.

## **Съществуващи подобни WEB проложения**

Подобни платформи за предлагане и търсене на работа не са рядкост в днешни дни, някои уеб приложения са доста сходни. Например работодателя създава обява, чрез която цели да наеме достатъчно квалификацирани служители като поставя и своите изисквания, а от друга страна служителите пишат мотивационно писмо и/или CV, което качват и изпращат. Това, с което обаче се различава уеб приложението е, че ще има и обратната връзка. Служителят, който може и да не притежава прекалено голяма квалификация ще може да създаде своя обява/пост, с която да представи своите умения или дейността, с която иска да се занимава, ще може да уточни времето, в което ще е свободен и начина, по който иска да бъде възнаграден. По този начин се създава една гъвкавост като може да бъде наеман само за няколко часа. Ще се спести чакането на работодателя някой да кандидатсва за неговата обява като той самия ще може да потърси хората, които иска да наеме. Другото преимущество е, че ще бъде сключвано устно споразумение като ако някой не е доволен ще има възможността да даде рейтинг на човека. По този начин хората, които са най-надеждни ще могат да изпъкнат и съответно по-лесно да се развият.

## **Jobs Tiger**

JobTiger.bg е първият кариерен сайт в България, създаден през 2000 г. Мисията им е да се създава стойност за хората, търсещи работа, бизнеса в обществото. Написан е на PHP и това, което предлага като услуги е свързано в няколко области като подбор на персонал, организиране на събития, обучения, стажове и наемане на персонал под наем. Приложението предлага лесен начин след кратка регистрация да създадеш обява. Като можеш да избереш да се регистрираш като търсещ работа или работодател. Имащ профил ще можеш да получиш достъп до всички услуги, които сайта предлага като да се консултираш с техния кариерен съветник или да споделиш своята автобиография. Ако си в ролята на работодател ще можеш да рекламираш своя бранд на сайта им.

## **Zaplata.BG**

Zaplata.bg e също доста сходно уеб приложение като отново в ролята на потребител може да си изберем какъв вид обява да бъде публикувана. Тоест съществува така наречената платена обява за работа, публикувана заедно с логото на компанията, която притежавате като работодател и се асонира в големите български сайтове като (**mobile.bg, imot.bg, bazar.bg и fakti.bg**). По този начин актуалните предложения за работа, които потребителя попълва ще може да се видят от над 2 000 000 уникални посетители всеки месец. Получавате допълнително промотиране от Google и Facebook като по този начин идеята им е промотирането да достигне до всеки пасивно търсещ работа. Като работодател ще имате възможност да се сдобиете с база данни от анонимни СV-та както в IT така и в BRO сектора.

Главен недостатък е, че услугите, които се предлагат от този кариерен сайт са само достъпни за България.

## **JOBS BG**

JOBS.BG е първият работен сайт - и един от най-популярните сайтове в България, посещаван от повече от 1.2 милиона уникални потребители всеки месец (Google Analytics). Като основен източник за кариерно консултиране по ИТ и инженерни сектори, JOBS.BG има добра репутация сред най-добрите кандидати в тези отрасли. JOBS.BG е медия за набиране на персонал с най-голяма аудитория в България, с аудитория, превишаваща многократно общата аудитория на всички свои конкуренти. Ето защо над 70 000 фирми публикуват над 20 000 предложения за работа на месец.

Jobs.bg е създадено през 2002 г. и управлявано от изпълнителния директор Пламен Вушев. Сайтът Jobs.bg е на разположение на английски и български език. Без да бъде най-впечатляващото работно място, това е 19-ият най-посещаван уебсайт в страната. Няма секция „За нас“ и на практика нищо на Linkedin (с изключение на 211 последователи), но в секцията „Въпроси и отговори“ има информация за таблото за работа. Дори социалните медии трудно могат да бъдат проследени: няма Twitter страница и редица фалшиви Facebook. Jobs.bg притежава Economy.bg, който публикува статии за пазара на труда, поддържа актуални информация за местните и международните събития. Главен минус, от който някой от потребителите се оплакват е, че Jobs.bg не изпращат автобиографиите на кандидаствалите на компаниите. Те очакват компаниите да влязат в профила си, където да отидат на списъка с кандидаствали и оттам да изтеглят въпросните автобиографии. Една дребна промяна, която може да коства загубена кандидатура за позиция. По-големият проблем е, че работодателите и търсещите работа дори не осъзнават това, те не разбират, че има кандидати или пък че тяхното CV e стигнало просто до jobs.bg.

## **Buljobs.bg**

Отново доста подобен на останалите сайт, който притежава вертикална търсачка като по този начин осигурява бърз и удобен достъп до хиляди обяви за работа в България чрез събиране на предложения за работа от три основни вида източници: работни портали, фирми, които търсят пряко служители и агенции за подбор на персонал. Чрез специализирана платформа за търсене Buljobs индексира свободните работни места, автоматично следи и предлага ефективни методи за систематизиране / категоризиране и филтриране на съдържанието по различни критерии: профил на позицията, бизнес сфера, местоположение, ключови думи, име на фирма или агенция за подбор на персонал.

## **Втора глава**

## **Функционални изисквания към системата**

Приложението се състои от две главни разделения тази на административната част и тази на обикновения потребител, който няма да има достъп до цялата функционалност на приложението. Обикновеният потребител може да бъде регистриран в качеството си на работодател или служител също с определени права.

* Системата ще позволява на анонимния потребител да:
  + Разглежда обяви за работа
  + Разглежда профили на търсещи работа
  + Се регистрира като работник (търсещ работа)
  + Се регистрира като работодател (предлагащ работа)
  + Се аутентикира с име и парола
  + Създава анонимно CV
* Системата ще позволява на аутентикирания работник да:
  + Разглежда обяви за работа
  + Редактира профила си
  + Кандидатства по обява
* Да управлява кандидатурата си по дадена обява
* Като води диалог с работодателя по обявата във вид на съобщения
* Като потвърди в случай, че бъде избран за работата
* Да маркира работата като приключена
* Да даде обратна връзка за работодателя
* Да създаде обява, с която да покаже уменията си като зададе :
  + - Период, за който може да извърши конкретната работа
    - Опит, който притежава
    - Избере сума, която би получил при свършените задължения
    - Начин на плащане
* Системата ще позволява на аутентикирания работодател да:
  + Разглежда профили на търсещи работа
* Редактира профила си
* Получи CV-то или прикачените файлове на кандидатсвалите за конкретната длъжност на посочения имейл
* Създаде нова обява
* Да търси обяви на служители, които отговарят на неговите изисквания
* Да управлява кандидатурите по дадена обява
* Да покани работник да кандидатства за обявата
  + - Като води диалог с кандидатите по обявата във вид на съобщения
    - Да избере кандидат
    - Да маркира работата като приключена
    - Да даде обратна връзка за работника
* Системата ще позволява на администратора да:
* Изпълнява CRUD ( Create, Read, Upadate, Delete) операциите – създава, преглежда, редактира и изтрива профили на потребителите и обявите свързани с тях
* Получава статистика с най-търсените длъжности, броя на кандидатсвалите и различна информация свързана с тях
* Блокира, създва или актуализира дадени профили
* Влиза в уеб приложението като аунтетикиран потребител с най-много права
* Филтриране на обяви за почова работа чрез:
  + - Заглавие
    - Местоположение
    - Минимална и максимална сума за заплащане
    - Период от две дати, межу които да започне работата
* Регистриране на потребител чрез въвеждане на:
  + - протребителско име
    - имейл
    - парола
    - две имена
    - местоживеене
    - рожденна дата
    - телефон за връзка

## **Управление на потребители:**

Тази операция се изпълнява от ролята на администратора. Администраторът може да направи търсене върху съществуващия списък с потребители, използвайки имейл или име или фамилия или потребител роля. Също така, администраторът може да създава нови потребители и да актуализира потребителската информация.

## **Управление на публикуваните обяви:**

Тази операция се извършва от администраторската роля.

Администраторът може да създава / редактира подробности за обявата, включително и профила на потребителя и неговите кандидатури, които е създал.

## **Регистрация на потребител:**

Тази операция може да се извърши от потребителя или администратора. След като е въвел личните си данни (потребителско име, собсвено име, имейл, парола, населено място) трябва да избере една от двете роли – работодател/служител.

Админът може да регистрира потребителя с принадлежност по избор - ролята на работодател или служител. Също така, администраторът може да модифицира принадлежността, заглавие на обявата и описание и нейното описани, да премахва качени файлове на потребителите в базата данни или да променя техни стойности.

## **Специално гласуване**

Тази операция се извършва от потребителя в двете си роли като работодател или служител.

След като дадена работа е свършена или нулирана потребителя ще има възможността да даде своята оценка – рейтинг. Като информация получава администратора под формата на статистики.

## **Търсене по обява**

Тази операция може да се извършва от всеки потребител като цели да улесни търсенето на работа на всеки, служещ си с нея.

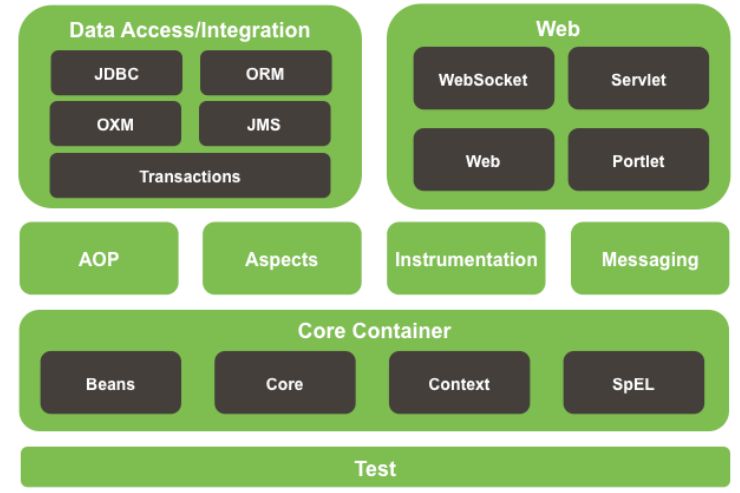
## **Съображения за избор на програмни средства и развойната среда**

## **Избор на език за програмиране**

Java е съвременен език за програмиране от високо ниво с отворен код, широкоразпространен, масово използван при разработката на сървърни системи и мобилни приложения, с много силно развита екосистема, с многобройни библиотеки и технологични рамки и съответно дава много перспективи за работа и професионално развитие. Java комбинира парадигмите на процедурното, обектно-ориентираното и функционалното програмиране по съвременен начин с лесен за употреба синтаксис.

## **Избор на технологична рамка**

Spring Framework е технологична рамка (или „framework), която е с отворен код. Тя е предназначена за Java платформата, като представя множество различни функции, които улесняват разработката на различни приложения, базирани на Java. Spring Framework е колекция от модули, позволяващи на програмистите да избират модулите които са необходими за техните приложения. Различните модули, предоставени от Spring са показани на *Фигура 2.2.2.1*.



Фигура 2.2.2.1

Като най-основните функционалности, които Spring Framework предлага за постигането на своите цели са:

**Dependency Injection** - Dependency injection (или просто „Инжектор“, но познат още и като „Контейнер на зависимостите“) – най-общо казано това е IoC контейнер (Inversion of Control), който предоставя логически средства за управление и настройка на Java обекти, като за целта използва reflection (отражение).

**Events (Събития)** - Механизъм, разработен така, че да се използва вътрешността на Spring Framework като например уведомяване за обновяване на контекста или може да се използва и за специфични за приложенията събития. Този eventing API е базиран на интерфейс, наречен org.springframework.context.ApplicationListener, който дефинира един метод, наречен onApplicationEvent.

**ORM** (Object-Relatoinal Persistence Framework) Обектно релационен мапинг или (ORM) е способността на програмирането да конвертира информацията от обект до релациония модел на таблицата в базата данни. Плюсовете на използването на ORM е, че ни позволява да пишем persistence класове чрез релационния мапинг, високата скорост (има много надеждни техники за извличане и заключване на информацията), стабилно, надеждно е и се използва като добра практика в индустрията.

**DAO support** - Поддръжката на обекти за достъп до данни (DAO) в Spring има за цел да улесни работата с технологии за достъп до данни като JDBC, Hibernate, JPA или JDO по последователен начин. Това позволява сравнително лесно превключване между гореспоменатите технологии за устойчивост. Също така позволява програмиране, без да се притеснявам за изключения, които са специфични за всяка технология.

**Spring Data JDBC**, част от по-голямото семейство на Spring Data, улеснява прилагането на JDBC базирани хранилища. Този модул се занимава с подобрена поддръжка за JDBC базирани слоеве за достъп до данни. Има за цел да бъде концептуално лесен.

**Software Design Pattern** Шаблоните за дизайн (design pattern) представляват най-добрите практики, използвани от опитни обектно-ориентирани програмисти. Шаблоните за дизайн са решения на често срещани проблеми, с които програмистите се сблъскват по време на разработка, придобити повреме на работа чрез метода за проба грешка.

## **Библиотеки**

**Logging Libraries** - са много често срещани, защото се нужни почти във всеки проект. Те са най-важното нещо за server-side приложенията. Въпреки, че JDK притежава собствена биоблиотека за регистриране, съществуват по-добри алтернативи като Log4j, SLF4j и LogBack.ries.

**Messaging Libraries**

Подобно на регистрирането и връзката с базата данни, съобщенията също са обща характеристика на много приложения в реалния свят на Java. Java предоставя JMS или Java Messaging Service, което не е част от JDK. За този компонент трябва да включите отделен jms.jar.

**Angular**

Angular е фреймуърк за разработка на приложения, поддържан от Google и специфично разработен за езикът за програмиране JavaScript.

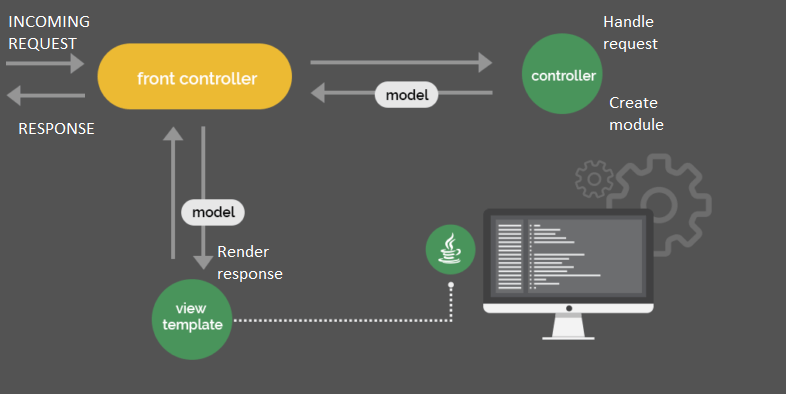
Целта на Angular е да разшири лексиката на HTML, като позволява дефинирането на персонализирани тагове. Езикът за програмиране, на който е написан Angular и който се използва с него е TypeScript. Този език е надмножество на JavaScript. Това означава, че целия JavaScript синтаксис е валиден TypeScript синтаксис, т.е. всеки код на JavaScript се компилира успешно като TypeScript код. TypeScript е създаден и се поддържа от Microsoft и популярността му все повече нараства.

## **Архитектура на приложението**

Уеб приложението се дели на два основни дяла: front-end и back-end. Front-end делът е частта от приложението, която потребителят вижда и с която взаимодейства. Този дял включва потребителски интерфейс на приложението. Back-end делът се състои от сървър, функционалната част на едно приложение и база данни. Приложението запазва въведената от front-end частта информация в база данни, която е създадена и се съхранява на сървър.

Back-end делът на уеб приложението е реализирано с приложение на Java SpringMVC и съответно Apache server и MySql база данни, чиято връзка се конфигурира в application.properties файла. Back-end делът на уеб приложението е реализирано благодарение на Spring Framework, което е inversion of control контейнер за Java платформата (управлението на обекти или части от програмната среда, които се прехвърлят в някаква рамка).

Front-end е реализиран чрез Thymeleaf, модерен сървър-сайд Java темплейт машина направена главно за уеб разработка. HTML шаблоните, написани с Thymeleaf, все още изглеждат и работят като HTML, позволявайки на действителните шаблони, които се изпълняват в приложението, да работят като полезни артефакти за проектиране.



Фигура 2.3.1

На *Фигура 2.3.1* е представено обработването на заявка на високо ниво.

## **Spring MVC**

В традиционното Java базирано уеб приложение Java Server Pages (JSP) отговаря за всички дейности като получаване на заявка, обработка на бизнес логиката, избиране на следващата страница и изпращане, отговор на потребителя. В този случай една малка промяна в функцията на приложението ще се нуждае от повече промените в съществуващия JSP код, който по-вероятно може да причини грешки в регресията (промяна в една част от код, предизвикващ грешки в друга част на кода).

С нарастващата сложност на продуктовите приложения, този подход е твърде скъп за поддръжката и гъвкавостта на приложения. Като решение на този проблем, архитектурата Model-View-Controller (MVC) придобива популярност.

MVC е архитектурен шаблон за дизайн в програмирането (sofrware design pattern), който предоставя разделяне и изолзаци между отделните нива, между бизнес логиката в моделите, потребителския интерфейс чрез изгледите (views) и връзката между тях – контролерът, който взима и обработва данни от модели и ги визуализира чрез изгледи или взима информация от изгледи и я запазава в базата данни чрез моделите (entity models).

В архитектурата на MVC различни аспекти на приложението са както следва:

* + (Models) Моделната част, която се състои от данните на приложението.
  + (Views) изгледите - частта, която е отговорна за визуализиране на данните на модела и генериране на HTML код, който браузърите могат да показват.
  + (Controller) Контролерът - отговорен за обработката на потребителските заявки, изграждането на подходящи модели или най-общо казано функционалността.

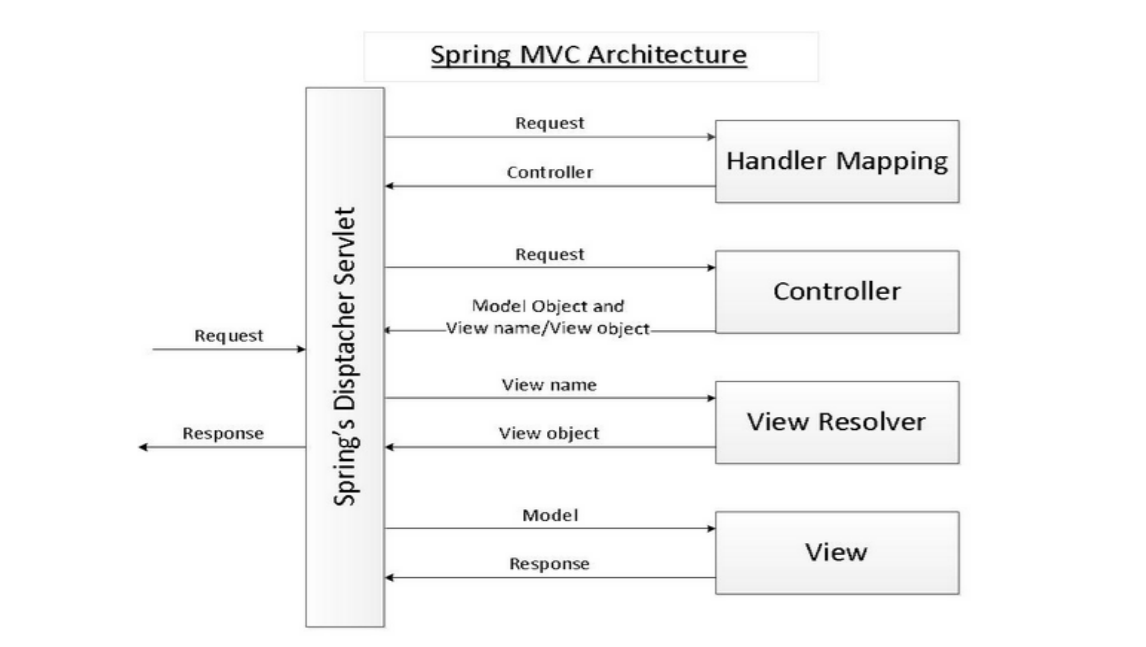
В **MVC** архитектура, не е необходима промяна с оглед към бизнес логиката, като по този начин се подобрява поддръжката и разширяемостта на приложенията. Така в този проект MVC архитектурата е предпочитана рамка за изграждане на приложението, защото:

* Spring MVC осигурява ясно разделение на ролите между контролер, JavaBean модел и изглед.
* Spring MVC предлага по-добра интеграция с други view машини, различни от JSP, като Velocity или Thymeleaf.
* Броят на необходимите конфигурации в Spring MVC е сравнително малък, което увеличава продуктивността на процеса на разработка.
* По дизайн Spring MVC работи с атрибути, които инжектират всички методи в класа, така се постига една оперативната ефективност.
* Можем почти изцяло да се абстрахираме от това, че работим с уеб.
* Spring MVC се основава на използването на интерфейси, което го прави по-гъвкав за наследяване на желания клас.

Въпреки, че архитектурните рамки на MVC решават много проблеми, свързани с развитието, подобряването на приложението, то взаимодействието му с базата данни все още е пречка за ефективността на проекта. Причината е, че обектно-ориентираните езици като Java и C# представят данните под формата на обекти, а традиционните системите за база данни съхраняват информацията под формата на таблици. Така, че съхранението и извличането на данни е сложна и неефективна задача. Като решение в проекта се използва Обектно-релационни карти (ORM), което е механизъм, чрез който се директно се „картографира“ информацията между обекта и базата данни, което в областта на компютърните науки е средство за конвертиране на данни между две системи, които са несъвместими, и ги кара да работят помежду си чрез обектно-ориентирано програмиране. Hibernate e едно такова ORM решение, което се използва в този проект. Тъй като Hibernate има вградена поддръжка за голямо разнообразие от релационни бази данни, приложението може лесно да бъде конфигурирано използвайте различни релационни бази данни в бъдеще. Така че, поради това, можете да използвате Hibernate, за да комуникирате с всяка база данни, която имате, като използвате само малка промяна с общия код на вашето уеб приложение. Това е изключително удобно и полезно, ако работите с множество бази данни, които иначе са несъвместими или трудни за използване.

## **Spring MVC архитектура**

Изградена е върху понятието централен сервлет (Dispatcher Servlet), който обработва всички HTTP заявки и отговори.



Фигура 2.3.1.1.1

## **Spring’s Dispatcher Servlet**

Служи като преден контролер. Той действа като посредник между приложението и неговите клиенти. Всички заявки към приложението се обработват от сървлета Dispatcher, който консултира Handler Mapping, за да определи кой контролер трябва да бъде извикан за конкретен поискване.

## **Handler Mapping**

Открива подходящите контролери за дадена заявка. URL адресите могат да бъдат съпоставени с контролните класове с помощта на XML конфигурация или анотации.

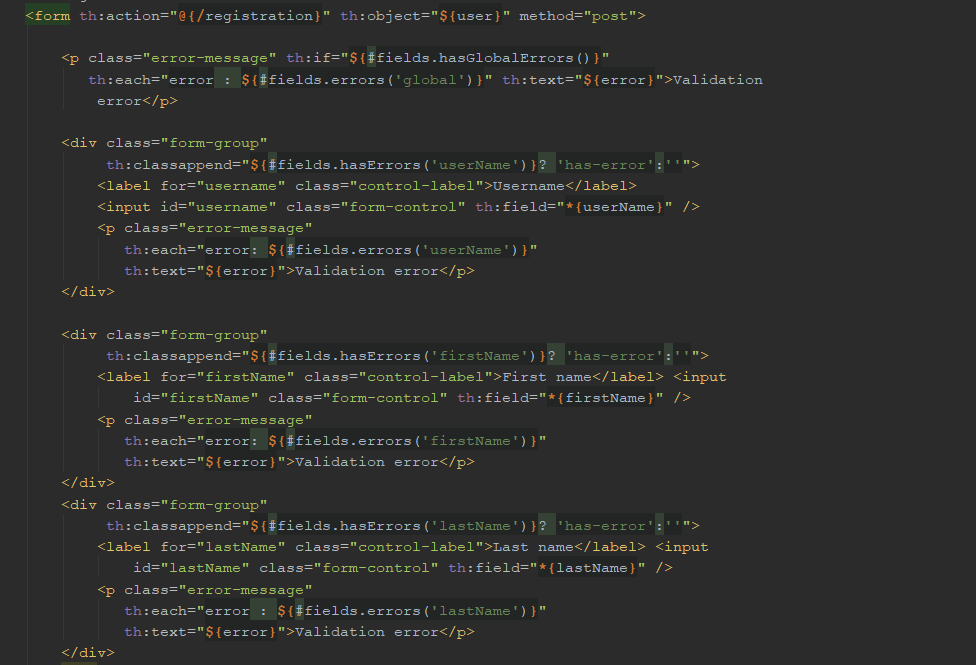
## **Controller**

## **View Resolver**

Точно както с всяка друга технология за преглед, която се интегрира с Spring, ще необходим резолвер за преглед, който ще разреши вашите важдания.

Точно както с всяка друга технология за преглед, която интегрирате с Spring, за Thymeleaf ще ви е необходим резолвер за преглед, който ще разреши вашите виждания. Най-често използваните резолвери за преглед при разработване с Thymeleaf са InternalResourceViewResolver и ResourceBundleViewResolver. И двете са декларирани в WebApplicationContext.

## **Registration View**



Фигура 2.3.1.6.1

В изгледа за регистрация е подаден обекта user и специфициран в Thymeleaf чрез:

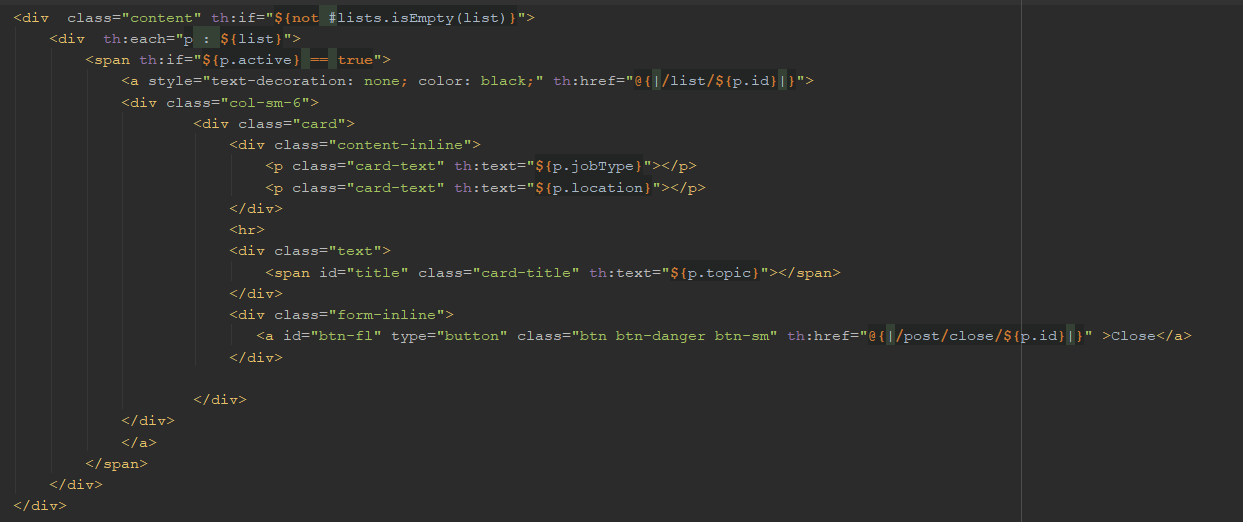


и достъпваме неговите атрибути чрез:



Дава се възможност на потребителя да попълни своите данни като се валидират полетата за парала и имейл.

## **Index page View**

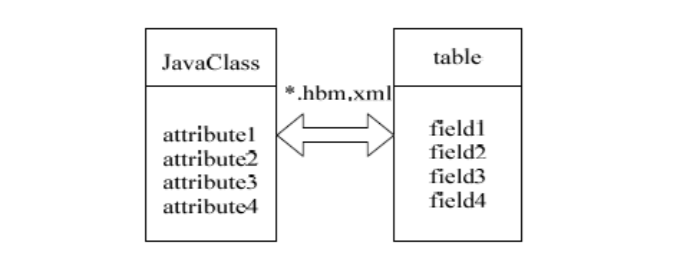


Фигура 2.3.1.7.1

На този изглед се подава лист от постове като се показват атрибутите на поста. По този начин потребителя с роля на работодател може да прави дадена обява активна или да я затваря.

## **Hibernate**

Hibernate е open-sourcce, сравнително малко, ORM (Object Relational Mapping) решение за Java приложения. Това е мощна, висококачествена услуга за обектно-релационна устойчивост. Използва се за мапване на обекти от обектно-ориентиран език в структурата на реалционни модели, базирани на SQL. Този ORM framework работи чрез трансформирането на едно представяне на дадени данни в друго както е посочено на *Фигура 2.3.2.1.*



Фигура 2.3.2.1

Файлът "\* .hbm.xml" дефинира връзката между класа Java и таблицата на базата данни, атрибутите на класа Java и полета в таблицата.

## **Слоеве на приложението**

Следните слоеве са включени в разработването на системата за предлагане и търсене на работа:

* + 1. **Presentation Layer:**

Този слой съответства на частта View в шаблона MVC. То включва взаимодействието с потребителя, реализира чрез три технологии - Thymeleaf, HTML и CSS. Thymeleaf е технология, използвана за създаване на платформа независими и динамични уеб страници. HTML е език за маркиране на съдържанието представяне в уеб страници. Cascading Style Sheets (CSS) се използва за управление на оформлението на няколко уеб страници

* + 1. **Controller Layer:**

Този слой съответства на контролера в модела MVC. Pотребителските заявки се обработват в този слой. Този слой се обработва от Spring. Той превежда входящите заявки, приема данните за заявката и ги предава на съответното в Service слоя. Контролерите осигуряват достъп до поведението на приложението без да знаят точното изпълнение на логиката на услугата. При получаване на отговор от слоя Service, данните на модела се визуализират в подходящо view за показване на потребителя.

## **Service Layer:**

Този слой се използва от слоя Controller за изпълнение на приложението функционалност. Той има цялата бизнес логика на приложението. Всяка услуга трябва инкапсулира логическо събиране на транзакции. Всеки метод на обслужване може да бъде дефиниран като най-малката атомна операция, която приложението прави в базата данни - така че методът на услугата или завършва цялата транзакция и поема или премахва цялата транзакция да поддържа базата данни в постоянно състояние във всеки даден момент.

## **Persistence layer**

Връзката с базата данни се репрезентира чрез:

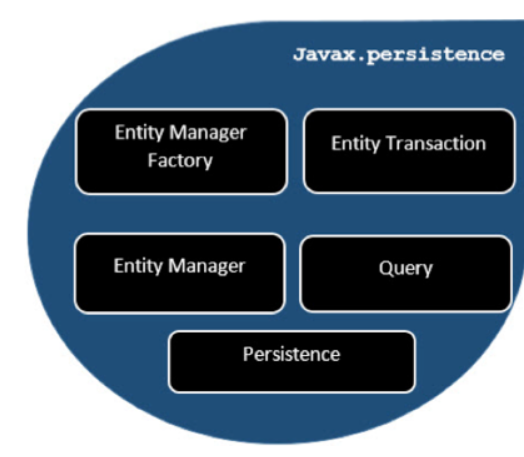
**EntityManager** инстанция, което осигурява функционалност за извършването на действия с базата данни. Много приложения изискват множество връзки с базата данни повреме на техния жизнен цикъл. Като пример, в уеб приложение е обикновено да има разделени връзки с базата данни с различни, несвързани EntityManager инстанции за всяка HTTP заявка.

Главната роля на **EntityManagerFactory** инстанцията, която наследява EntityManager е да си комуникира само с една специфична база данни чрез менежирането на информацията ефикасно тоест примерно при (pool of sockets).

Това позволява ефикасен метод да се създават множество ЕntityManager инстации или множество връзки с базата данни. Тоест при смяна на базата данни не е нужно да се преработва цялостно кода на приложението а само да се добави нов EntityManagerFactory клас.

Операциите, които модифицират информацията в базата данни изисква активни транзакции. Тези транзакции се обработват от EntityTransaction инстанция на EntityManager класа.

За използването на куерита към базата данни се използва така наречения Query класа, което е инстанция също на EntityManager.



Фигура 2.4.4.1

## **Структура на базата данни**

## **Таблица Users**

Таблица, описваща как изглежда един потребител в базата данни.

|  |  |
| --- | --- |
| users | |
| Поле | Описание |
| **id** BIGINT(20) PRIMARY KEY | уникален идентификатор |
| **email** VARCHAR(255) | имейл |
| **first\_name** VARCHAR(255) | собствено име |
| **last\_name** VARCHAR(255) | фамилно име |
| **password** VARCHAR(255) | парола |
| **username** VARCHAR(255) | титулувано име |

## **Таблица Role**

Таблица, която показва как изглежда една роля в базата данни.

|  |  |
| --- | --- |
| Role | |
| Поле | Описание |
| **id** BIGINT(20) PRIMARY KEY | уникален идентификатор |
| **name** VARCHAR(255) | име обозначаващо ролята |

## **Таблица Users\_Roles**

Таблица, която показва как изглежда връзката между роля и потребител в базата данни.

|  |  |
| --- | --- |
| users\_roles | |
| Поле | Описание |
| **user\_id** BIGINT(20) FOREIGN KEY | уникален идентификатор |
| **role\_id** BIGINT(20) FOREIGN KEY | уникален идентификатор |

## **Таблица File\_Мodel**

Таблицата, която показва как изглежда модела на един файл.

|  |  |
| --- | --- |
| file\_model | |
| Поле | Описание |
| **id** BIGINT(20) PRIMARY KEY | уникален идентификатор |
| **name** VARCHAR(255) | име за обозначаващия файл |
| **created\_at** DATETIME | време на създаване |
| **updated\_at** DATETIME | време на променяне |
| **mimetype** VARCHAR(255) | Тип |
| **pic LONGBLOB** | Данни |

## **Таблица Post**

Таблицата, която показва как изглежда един пост.

|  |  |
| --- | --- |
| Post | |
| Поле | Описание |
| **id** BIGINT(20) PRIMARY KEY | уникален идентификатор |
| **text** VARCHAR(255) | Съдържание |
| **topic** VARCHAR(255) | тема |

## **Таблица Message**

Таблицата, която показва как изглежда създаденото потребителско съобщение в базата данни.

|  |  |
| --- | --- |
| Message | |
| Поле | Описание |
| **id** BIGINT(20) PRIMARY KEY | уникален идентификатор |
| **content** VARCHAR(255) | Съдържание |

## **Таблица Candidacy**

Таблицата, която репрезентира потребителската кандидатура за работа в базата данни.

|  |  |
| --- | --- |
| Candidacy | |
| Поле | Описание |
| **id** BIGINT(20) PRIMARY KEY | уникален идентификатор |
| **description** VARCHAR(255) | съдържание |
| **creation\_date** Date | дата на създаване |
| **state** Text | състояние |
| **post\_id** Integer | връзка с пост |
| **user\_id** Int | връзка с потребител |

**Трета глава**

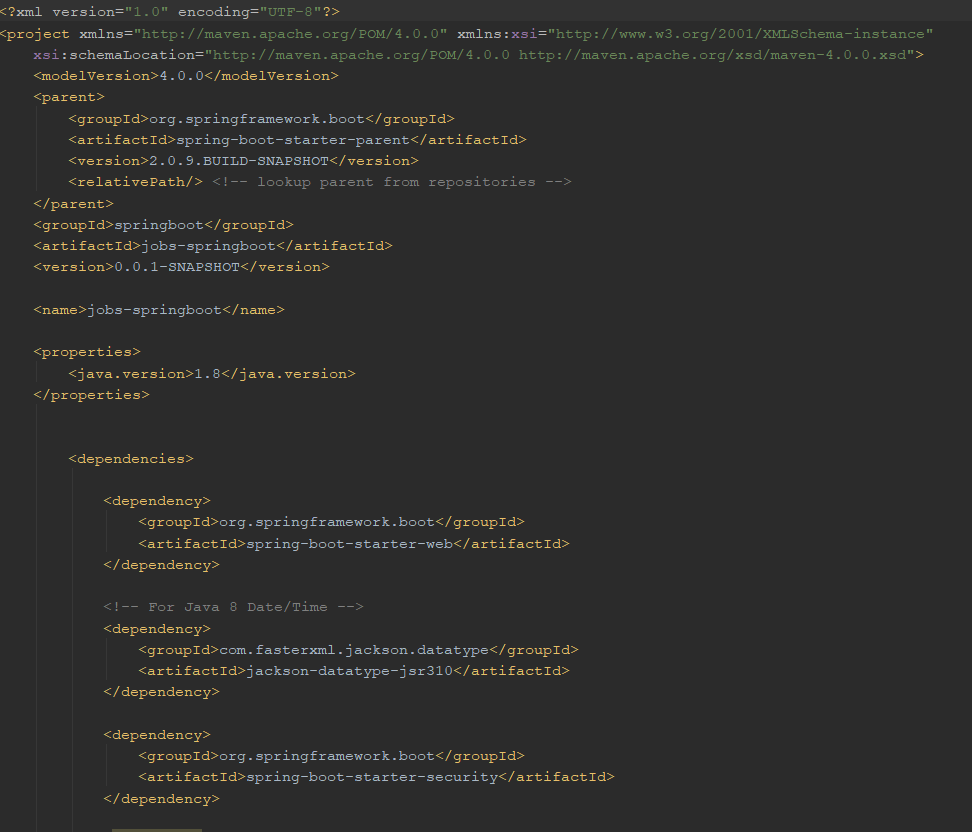
## **Реализация на проекта**

## **Създаване на POM**

Spring Framework осигурява цялостен модел на програмиране и конфигуриране за съвременни Java-базирани корпоративни приложения - на всякакъв вид платформа за внедряване. Spring Boot е малко повече от набор от библиотеки, които могат да бъдат използвани от системата за изграждане на всеки проект. За удобство, framework-а предлага и интерфейс на командния ред, който може да се използва за изпълнение и тестване на Boot приложения.

## **Създаване на Classpath Dependencies**

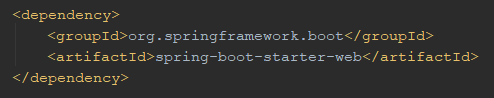
За да се опрости dependency графиката, функционалността на Boot е модулирана на групи. Така зависимостите като примерно външни библиотеки или плъгини могат да бъдат въведени в проект чрез импортиране на така наречените "стартови" модули на Boot. За лесно управление на версиите за използване на конфигурацията по подразбиране, рамката представя родителски POM файл, който се наследява от проекта. Maven – тул, който билдва приложението и създава jar файл, който може да се рънне. Изтегля плъгините и ги добавя в build path-а. Създава се Maven **pom.xml** файл.



Фигура 3.1.1.1

На *Фигура 3.1.1.1* е показан част от POM.xml файла.

За проекти, които не са изградени с CLI, Boot осигурява хост от "стартерни" модули, които дефинират набор от зависимости, които могат да бъдат въведени в система за изграждане, за да се разрешат специфичните библиотеки, необходими от рамката и нейната основна платформа. Като пример за това е spring-boot-starter-web (*Фигура 3.1.1.2*), като целта е да се получи по-бързо конфигурирано и работещо приложение.



Фигура 3.1.1.2

Когато зависимостта “spring-boot-starter-web” е добавена, тя автоматично се конфигурира с bootstrap и Tomcat container, където се създават микросървиси. Tomcat мапва всички крайни точки, микро-услуги, като информация за сървъра, метрики на приложенията и подробности за околната среда.

Освен това, когато се добави „spring-boot-starter-security“, автоматично се конфигурира Spring Security, за да осигури на приложението основна идентификация и други разширени функции за сигурност.

## **Създаване на Micro-Services**

Като се има предвид опростяването на Boot за разработването на приложения Spring, чрез предоставената му възможност за импортиране на зависимости по модулен начин, framework-a очевидно притежава огромна полезност при разработването на микро – услуги. Микро-услугите са все по-популярна приложна архитектура в корпоративната инфраструктура, тъй като позволяват бързо създаване и развитие на по-малки кодови бази и интеграция на приложението.

## **Връзка с базата данни**

Микро-услугите могат да бъдат изградени за различни цели, но една гаранция е, че повечето ще се нуждаят от възможност за четене и запис в база данни. Spring Boot прави интеграцията на базата данни тривиална задача с възможността да конфигурира автоматично Spring Data за достъп до базата данни. Чрез просто включване на модула spring-boot-starter-data-jpa като част от проекта, така проекта ще разбере, че се изисква достъп до базата данни и ще създаде необходимите звена в контекста на приложението. Така с предварително подготвени модели за връзка с базата данни.

## **Modeling Entities (Моделиращи единици)**

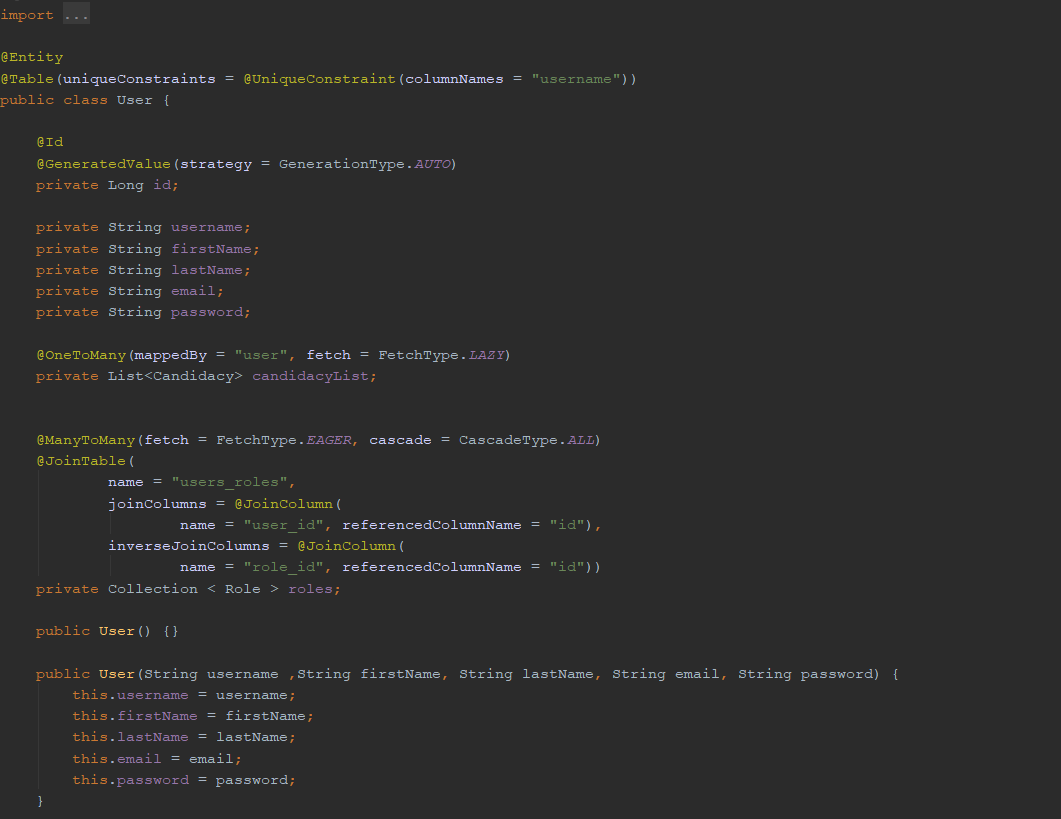
Това са класове, които репрезентират информацията на обектите към базата данни. Трябва да съдържат анотацията *@Entity* или @*Document,* която дефинира, че един клас може да бъде съпоставен с таблица. И това е, просто е маркер, като например интерфейса **Serializable**.

## **User**

От User моделa се съхранява и извлича информация за потребителите на приложението от таблицата user в базата данни. Създава се връзка много към много с ролите в таблицата users\_roles.

Когато се създава нов обект, трябва да направият поне две неща:

* да се анотира с @Entity
* да се създаде ID поле и да бъде анотирано с @Id



Фигура 3.3.1.1

За да вкараме някои общи данни, които предствляват User може да създадем файл с името schema.sql или data.sql в папката на нашия проект и да напишем заявката, която ще качи предварително някакви данни.

## **Role**

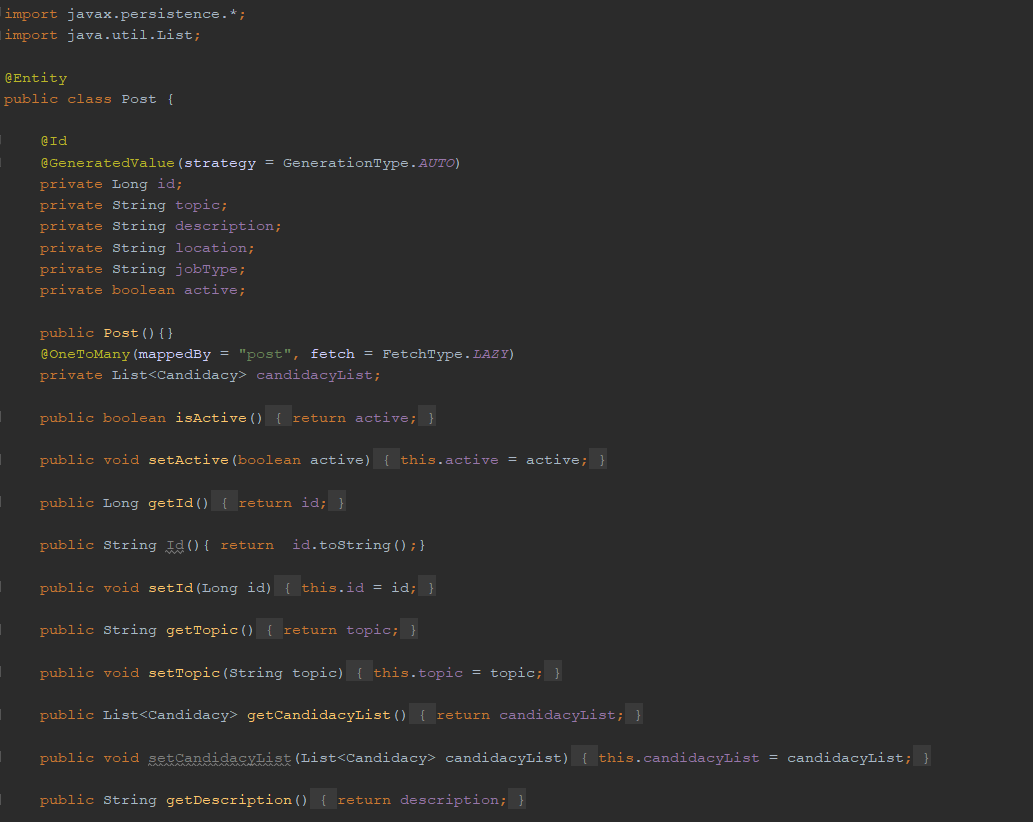
Role моделът чрез него се съхранява и извлича информация за ролята на потребителите на приложението от таблицата role в базата данни. Анотацията @GeneratedValue трябва да конфигурира начина на увеличаване на посочената колона (поле). Например, когато използваме Mysql, можете да укажете auto\_increment в дефиницията на таблицата, за да го направите увеличаващ се, и след това да използваме:



Фигура 3.3.2.1

## **Post**

От Post моделa се съхранява и извлича информация за потребителите на приложението от таблицата post в базата данни.



Фигура 3.3.3.1

## **Message**

Моделът Message съхранява данните за обекта съобщение в базата данни.



Фигура 3.3.4.1

При стартиране, предоставения SQL код ще бъде изпълнен. Сега когато микро услугата разполага с данните, с които да започне.

## **Candidacy**

Candidacy модела съхранява данните за една кандидатура. Съдържа като външни ключове този на обявите и този на потребителите, кандидатствали за конкретната работа. Притежава член променлива State, която е ENUM и оказва активността на дадена кандидатура. Работодателят има възможността да удобрява или да отказва дадена кандидатура на потребител и по този начин тя става неактивна.



Фигура 3.3.5.1

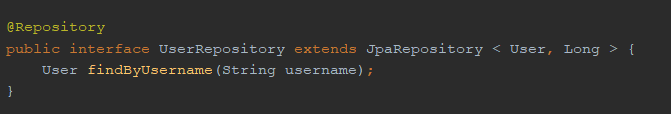
## **Релациите**

Реализацията на релационните връзки между моделите в контекста на Spring се случва посредством анотациите:

* @*OneToMany* или @*ManyToOne* - дефинират едно-към-много или много към едно взаимоотношение между 2 единици.
* @*ManyToMany* – дефинира взаимоотношение много-към-много.
* @*JoinColumn* показва, че обектът е собственик на връзката, а съответната таблица има колона с външен ключ към друга таблица.
* *mappedBy* показва, че обектът е част от връзка.

## **Репозитори**

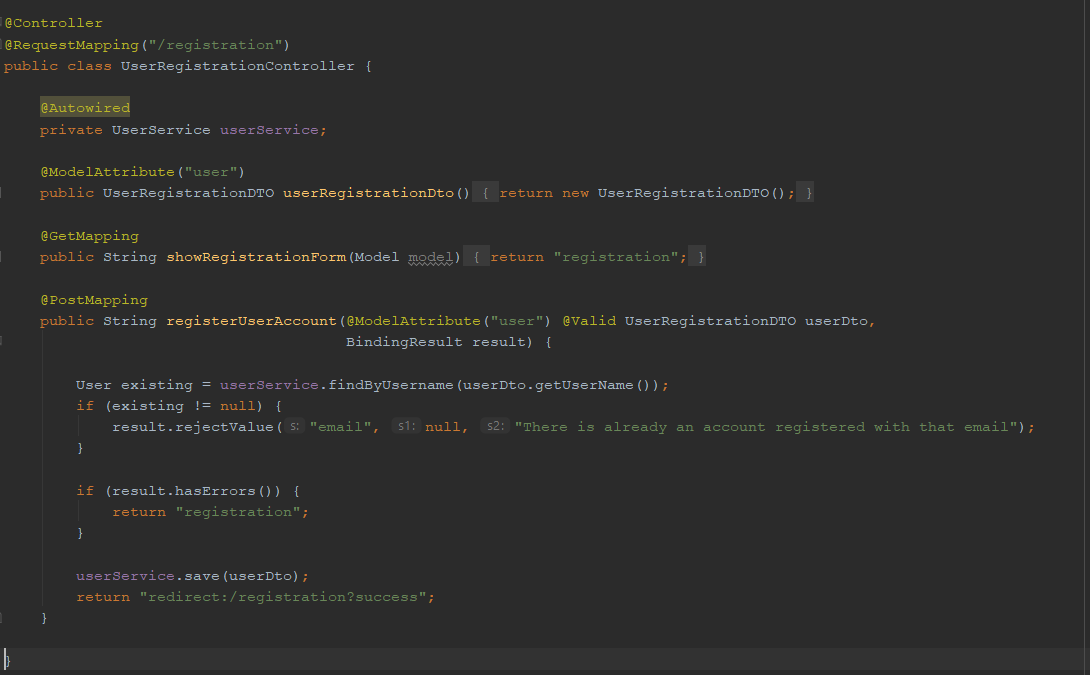
На *Фигура 3.5.1* е изпобразено Spring data’pattern, който предствлява интерфейса UserRepository, приемащ данни като Data Access Object за User entity класа.



Фигура 3.5.1

В него има един метод, който се имплементира от сървиза като целта е да се получи разделене на слоевете. JpaRepository предоставя някои общи интерфейсни методи за създаване, извличане, актуализиране и изтриване на обекти и колекции от обекти.

След създаването на интерфейса UserRepository, Spring Boot spring-data-jpa слоя ще намери го във вътрешността на проекта и ще го въведе в контекста на приложението, което го прави готов за използване от контролерите и сървизите.



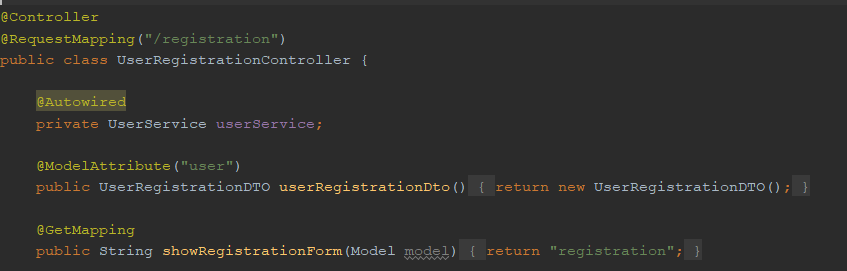
Фигура 3.5.2

## **Анотации**

Анотацията *@****Autowired*** осигурява по-фин контрол върху това къде и как трябва да се извърши автоматичното свързване. @***Autowired*** анотацията може да бъде използвана за автоматична настройка на метода на сетера, подобно на @***Required*** анотацията. Тази автоматична конфигурация се появява само когато Boot приложението притежава @***EnableAutoConfiguration*** анотацията.

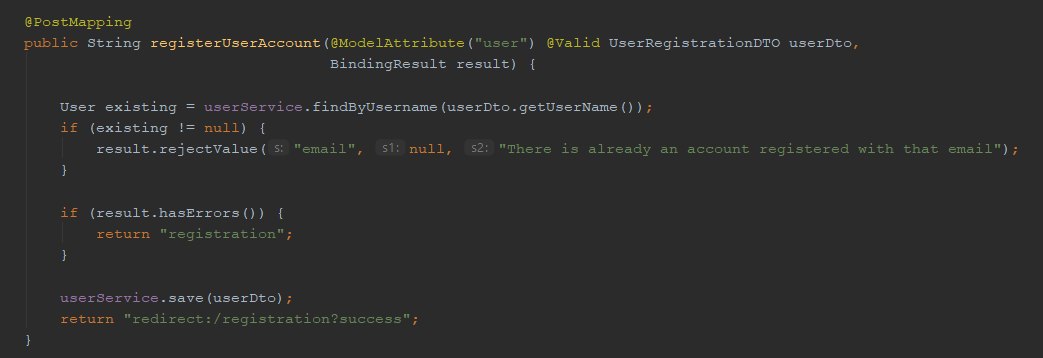
## **Контролери**

В контролерите като PostController, UserRegistrationController, JobApplicationController или UploadFileController се отговаря на конкретните заявки на потребителя.



Фигура 3.7.1

Когато потребителят на дадена микро-услугата изпълни HTTP POST заявка към крайната точка на приложението, Spring ще създаде инстанция примерно на User модела и след това ще използва UserRepository за съхраняване на обекта в релационната базата данни MySql.



Фигура 3.7.2

В случая POST заявката, която се обработва от UserRegistrationController взима получения модел, валидира го и ако няма грешки запазва резултата в userService, в който е инжектнато репозиторито.

По интересна част от кода е заявката при създаване на кандидатура в JobApplicationController, където се подава уникалния индентификатор на поста като @PathVariable и се проверява дали има съществуваща кандидатура и ако няма се създава такава с предварително поставени връзки с поста и потребителя.

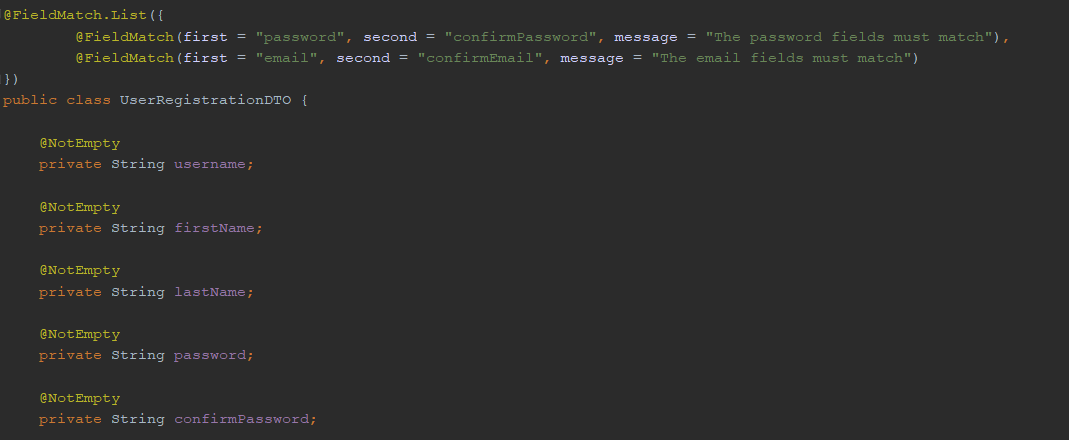


Фигура 3.7.3

Обикновено получената информация се валидира, чрез UserRegistrationDTO класа или подобен клас.

## **Data Transfer Object**

DTO е обект, който пренася данни между процесите. Tрябва да се намали броя на заявките. Решението е да се създаде обект за прехвърляне на данни, който може да съдържа всички данни за повикването. Той трябва да може да се сериализира. Обикновено съдържа в себе си член променливи, гетъри и сетъри.



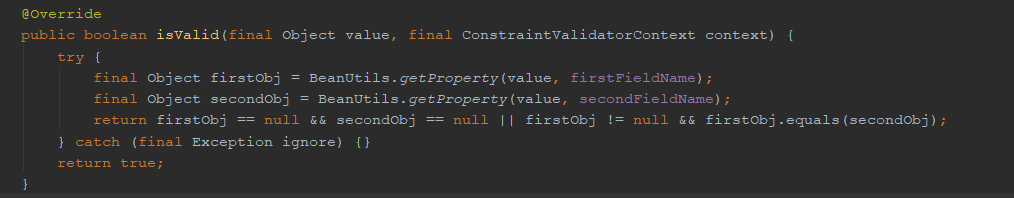
Фигура 3.8.1

## **Валидация**

Валидирането на информацията се извършва посредством тaзи **FieldMatch** анотация. Това е анотация на ниво клас, където можем да сравним две полета за равенство и да преминем в незадължително съобщение, за да се покаже на потребителя, ако валидирането на ограничението не е успешно.

**FieldMatch** анотацията се валидира чрез FieldMatchValidator класа, който прочита двете полета в случая (*Фигура 3.8.1)* полетата - first и second.

Методът **isValid()** е извикван повреме на бийн валидацията. Този метод прочита двете полета и ги сравнява чрез common-beanutils. Когато първото поле не съвпада с второто валидацията връща съобщение за грешка.



Фигура 3.9.1

## **Хеширане на паролите**

Паролите винаги трябва да бъдат кодирани с помощта на защитен алгоритъм за хеширане, предназначен за тази цел (а не стандартен алгоритъм като SHA или MD5). Това се поддържа от елемента <password-encoder>. С кодирани пароли bcrypt оригиналната конфигурация на доставчика на удостоверяване ще изглежда така:



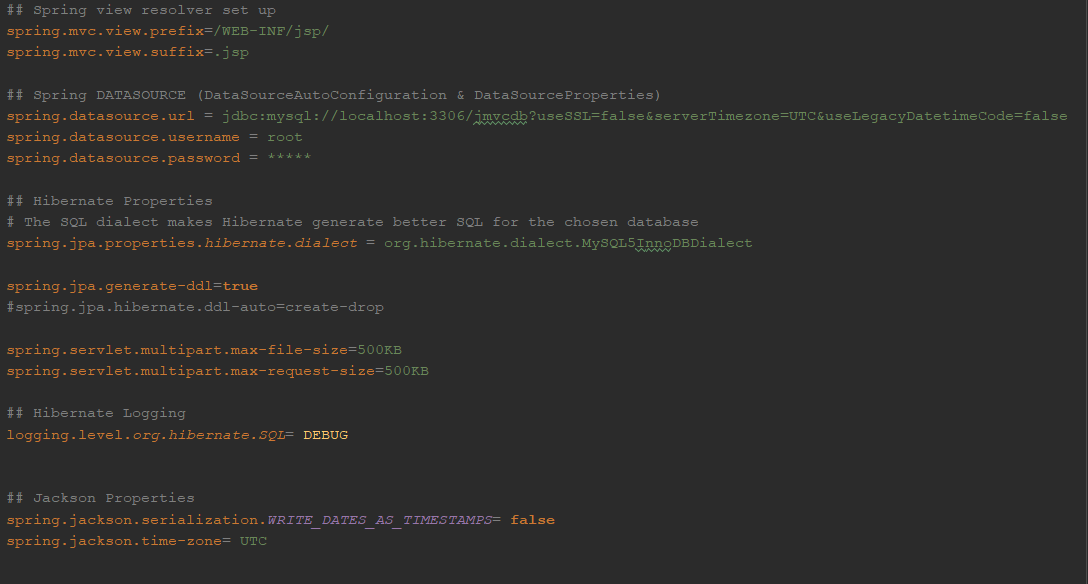
Фигура 3.10.1

Bcrypt е добър избор за повечето случаи, освен ако нямате система, която ви кара да използвате различен алгоритъм. Ако използвате прост алгоритъм за хеширане или, още по-лошо, съхраняване на обикновени текстови пароли, това няма да бъде достатъчно надеждно.

## **Конфигурации**

Spring Boot с огед към противното поммага мног със своя лесен достъп, когато трябва да се замени нещо от настройките по подразбиране. По подразбиране конфигурацията на приложения може да бъде дефинирана с помощта на файл наречен application.properties.

Предпочитан подход, обаче, е да се използва YAML конфигурация, която дава структура и дълбочина на вложената конфигурация.



Фигура 3.11.1

Конфигурация на релационния път за връзката между controller и view се извършва в MySimpleUrlAuthenticationSuccessHandler класа, който имплементира от AuthenticationSuccessHandler. В някои сценарии може да искате да пренасочите различни потребители към различни страници в зависимост от задачите, възложени на потребителите.

Например, може да искате потребителите с роля USER да бъдат пренасочени към началната страница, докато потребителите с роля ADMIN да бъдат пренасочени към страницата за добавяне на служител.



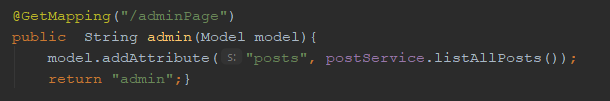
Фигура 3.11.2

## **Spring модел атрибутите**

Spring MVC извиква части от данни, които могат да бъдат достъпни по време на изпълнението на изгледите. Еквивалентният термин в езика Thymeleaf е контекстните променливи.

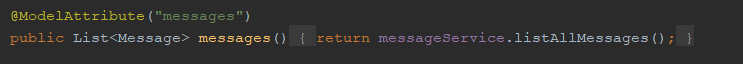
Има няколко начина за добавяне на атрибути на модела към изглед през Spring MVC. По-долу ще намерите някои често срещани случаи:

Добавете атрибут към Модела чрез неговия метод *addAttribute*:



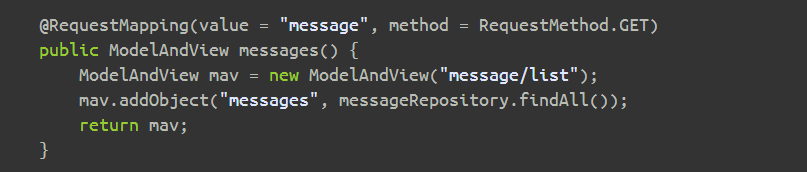
Фигура 3.12.1

Разкрива общите атрибути чрез методи, отбелязани с *@ModelAttribute*:



Фигура 3.12.2

Връща ModelAndView с включени атрибути на модела:

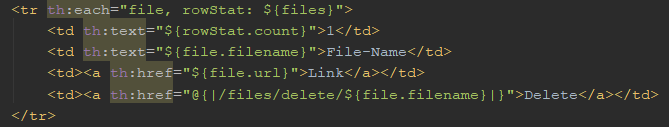


Фигура 3.12.3

Във всички горепосочени случаи атрибутът на съобщенията се добавя към модела и ще бъде достъпен в изгледите на Thymeleaf.

В Thymeleaf тези атрибути на модела (или контекстните променливи) могат да бъдат достъпни със следния синтаксис: $ {attributeName}, където атрибутName в нашия случай е съобщения.

Това е Spring EL израз. Накратко, Spring EL (Spring Expression Language) е език, който поддържа запитвания и манипулиране на график на обект по време на изпълнение.



Фигура 3.12.4

## **WebSecurityConfigurerAdapter**

Анотацията @*EnableWebSecurity* и **WebSecurityConfigurerAdapter** работят заедно, за да осигурят уеб базирана сигурност. Разширявайки WebSecurityConfigurerAdapter и само няколко реда код, ние можем да направим следното:

* Да създадем изискване от потребителя да бъде аутентикиран преди да получи достъп до някой URL адрес в приложението ни.
* Предварително да се създаде потребител с потребителското име „user“, паролата „passoword“ и ролята на „Emploee“
* Активира аутентикация чрез HTTP Basic и Form
* Автоматично ще рендерира login и logout?success страниците

## **Заявки на контролерите**

## **Get Заявки**

* **/post/view/{id}** – връща формата за кандидатсване за обява

Параметри: id – уникалния индентификатор на пост

Описание: Търсят се поста и текущия потребител от базата данни като ако не може да бъде намерен се създава нов потребител и се подават на модела, който ще бъде променян в изгледа.

* **/index\_user** – връща началната страница на потребителя с роля на служител.
* **/index -** връща началната страница на потребителя с роля на employer като подава лист от постовете в модела, използван от изгледа.
* **/adminPage** - връща началната страница на потребителя с роля на администратор като подава колекции от постовете, кандидатурите, съобщенията и потребителите в модела, използван от изгледа.
* **/userSettings –** връща форма, чрез която да се променят данните на потребителя, тоест настройки на потребителския профил.

Описание: Подава се текущия логнат потребител в модела на изгледа чрез SecurityContextHolder. Ако не бъде намерен, потребителя се връща в началната страница.

* **/post/delete/{id} –** изтрива записа на поста

Описание: От подадения уникален индентификатор се намират данните на поста и се изстриват.

* **/posts –** показва създадената пост форма
* **post/close/{id} -** редиректва до основната страница (index).

Параметри: id – уникалния индентификатор на пост.

Описание: Променя състоянието на поста от активно в неактивно като не трие запи от базата данни.

* **post/update –** връща страница с форма за промяна на поста.
* **list/{id} –** връща страница с листнатите апликанти за дадена обява.

Параметри: id – уникален индентификатор за постовете

Описание: Показва всички кандидатсвали за дадения пост.

* **/registration** – връща страницата за регистрация на потребител

## **Post Заявки**

* **/registration** – връща началната страница като казва дали регистрацията е била успешна.

Параметри: User представен през UserRegistrationDTO

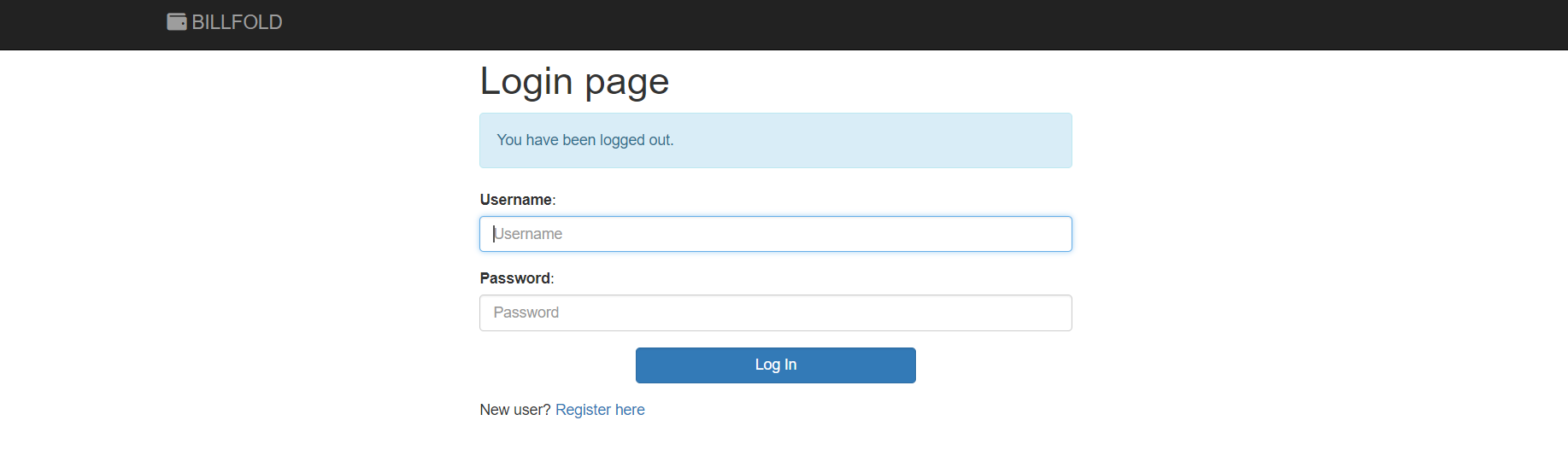
Описание: Валидира подадения обект и след това проверява дали има подобен потребител.

* **/posts –** запазва пост като го валидира преди това
* **/update/user –** промяна на текущите данни на потребителя

## **Четвърта глава**

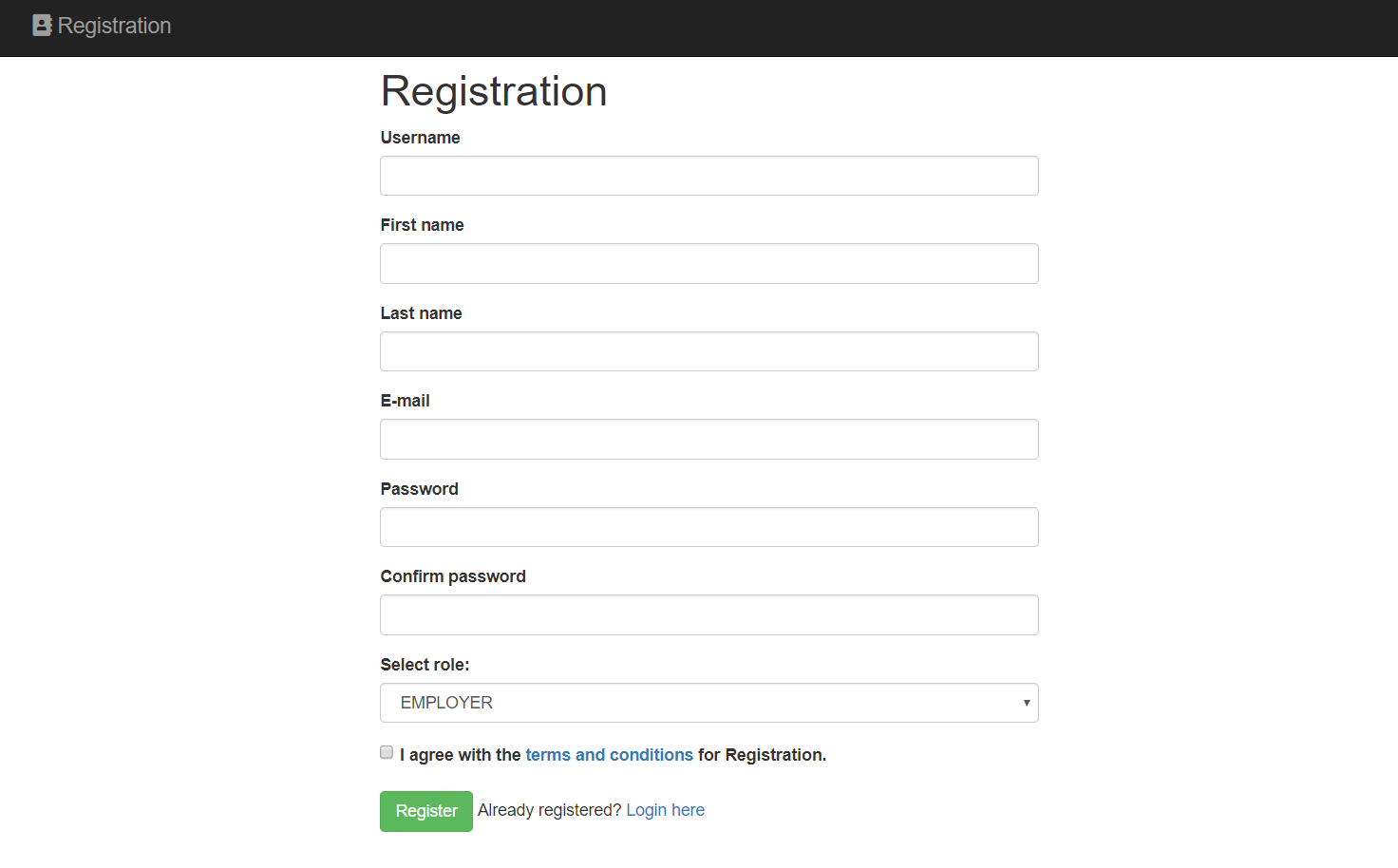
## **Ръководство на потребителя**

В началната страница на приложението се показва login форма и навигация, на която ще има полезна информация относно платформата. Потребителят трябва да попълни необходимите полета за да влезе в системата като са нужни потребителско им, име



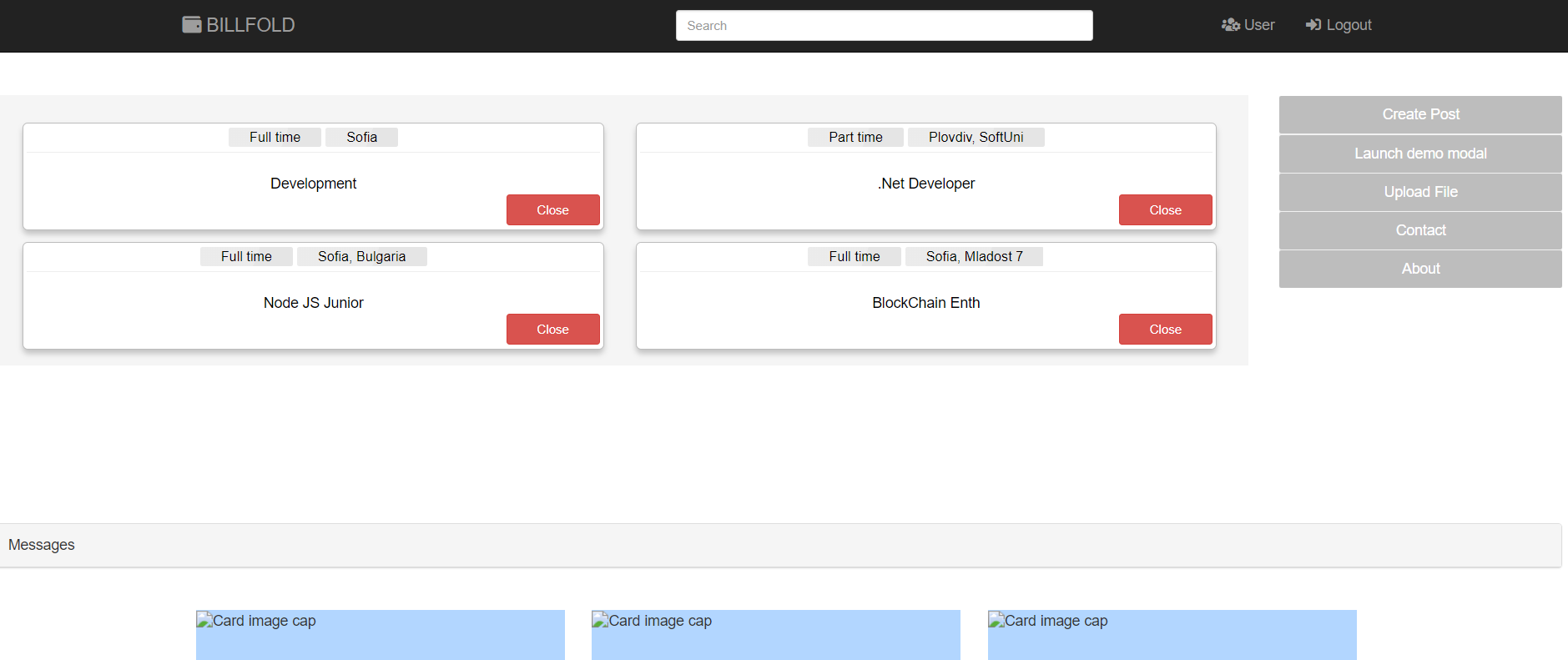
Фигура 4.1

Ако потребител липсва, то той може да се регистрира на оставения линк.



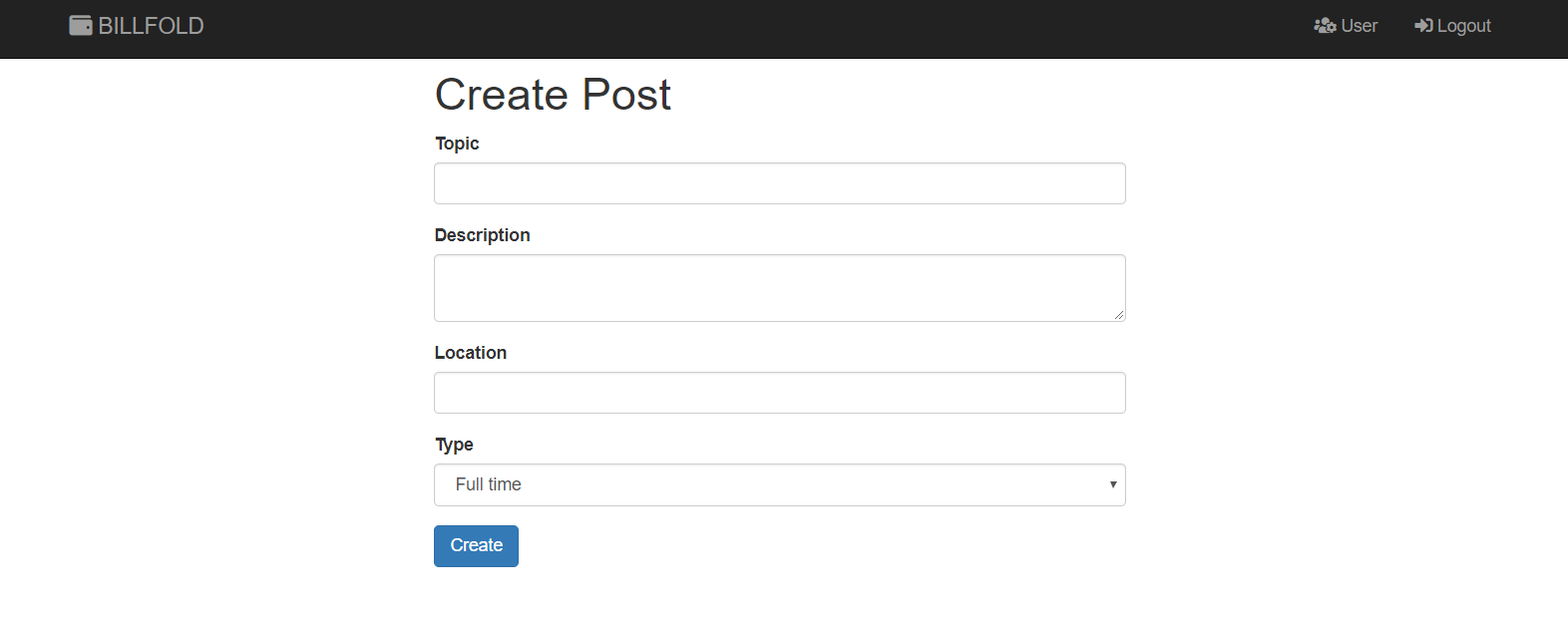
Фигура 4.2

В навигацията отгоре потребителя прави регистрация поради лиспа на профил. След успешен вход приложението се отваря главния екран с всички създадени обяви и съответната информация за тях. От ролята на потребителя се отварят различни изгледи.

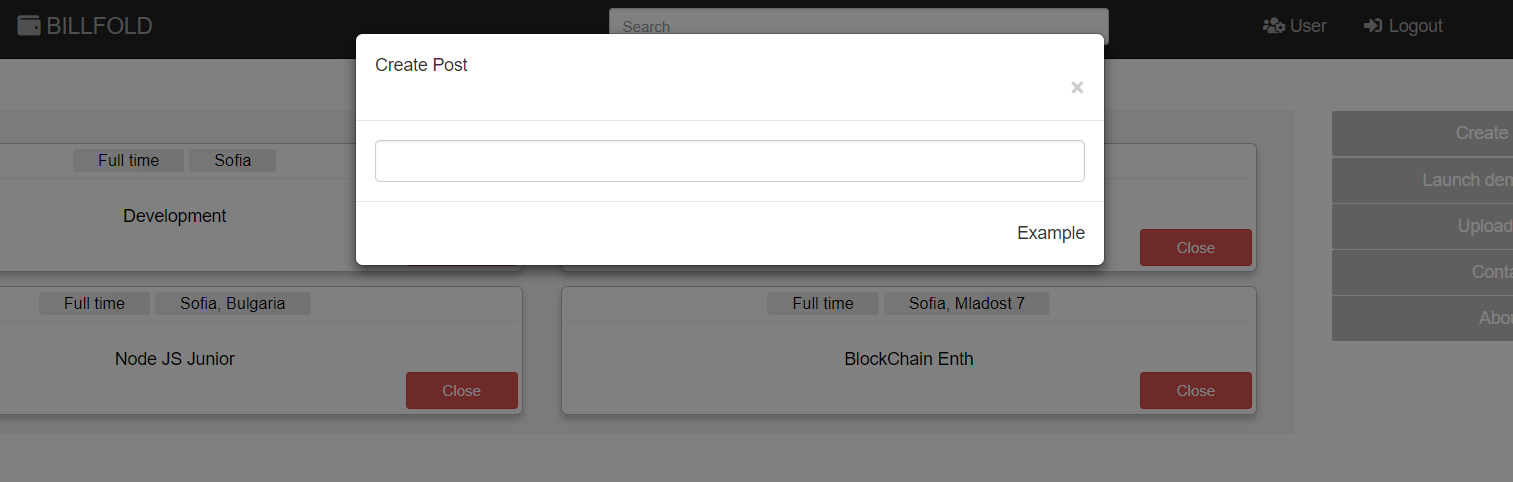


Фигура 4.3

Като работодател имаш правото да създаваш обява, по която да се кандидатсва.



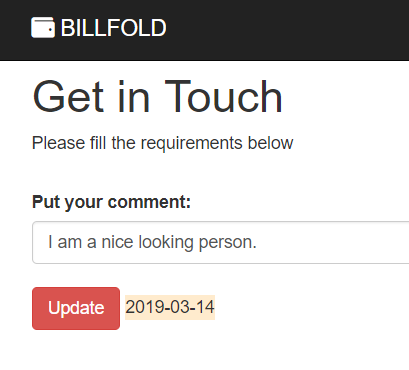
Фигура 4.4



Фигура 4.5

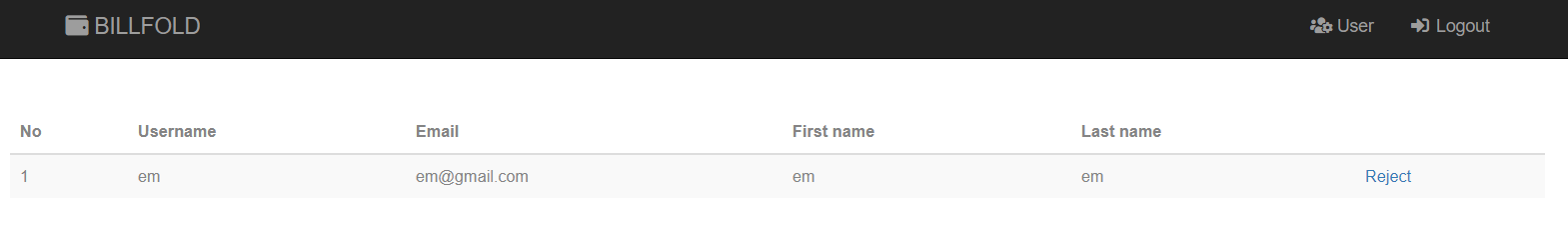
Отваря се модал, в който може да се създаде пост.

Когато потребителя натисне някоя обява, се отваря екран с информация за създателя на обявата, информацията за самата обявата и бутон за кандидатстване за обявата.



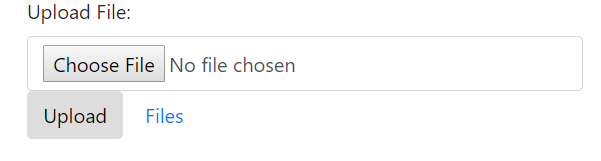
Фигура 4.6

Когато потребителя в ролята на служител кандидатсва по обява, той може да въведе кратко описание и да качи своето CV в базата данни. Текущите данни ще бъдат листнати както и датата на поселдната промяна.

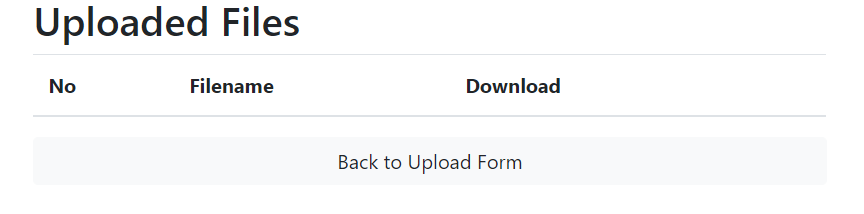


Фигура 4.7

Потребителят може да поиска да качи файл с личните си данни.

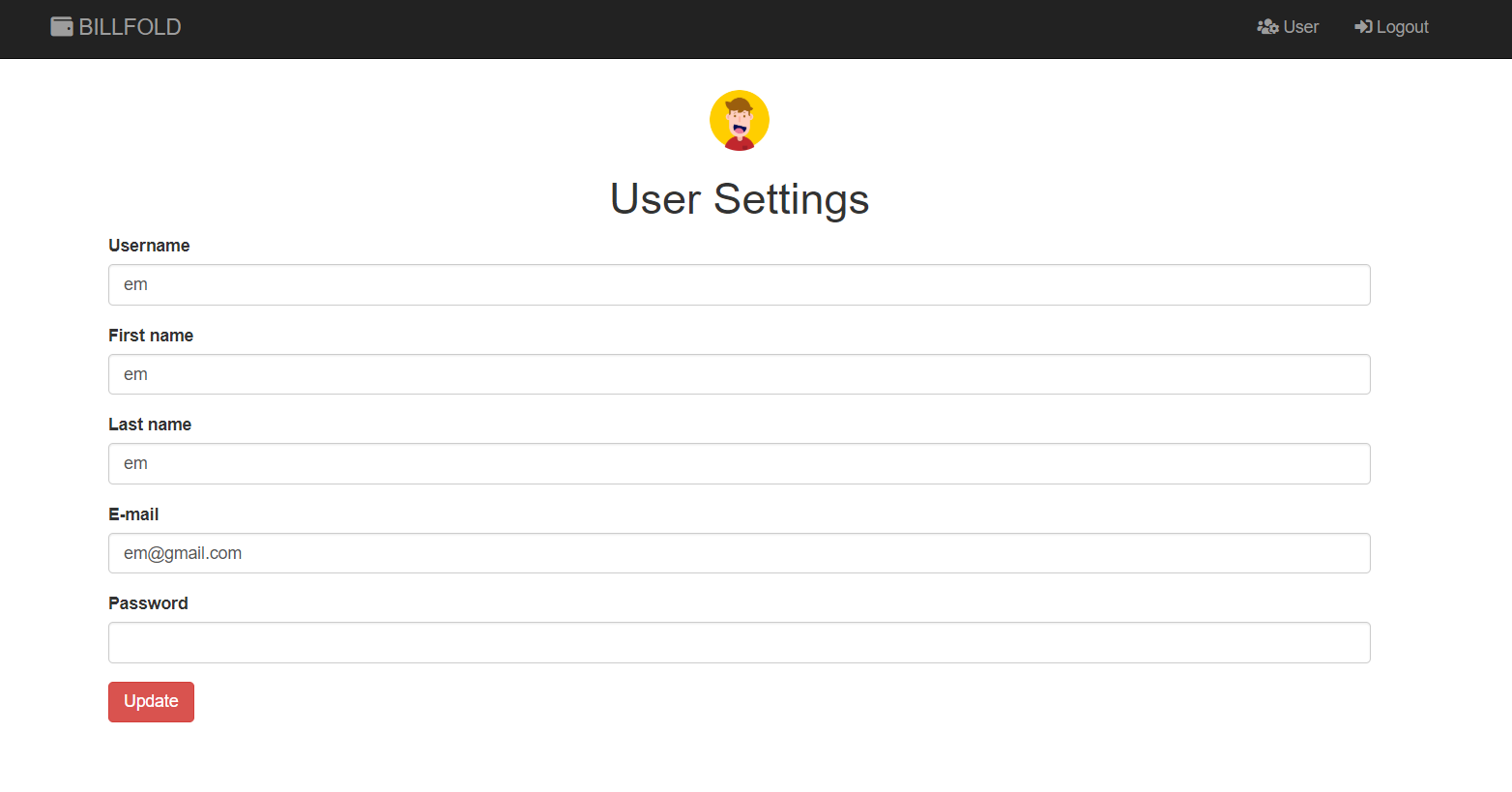


Фигура 4.8



Фигура 4.9

Потребителят може да реши да промени личните си данни, затова когато избере настройки се показва следната страница.



Фигура 4.10

## **Заключение**

В настоящата дипломна работа бе разработено уеб приложение, което предоставя възможност на потребителите да кандидатсват и да наемат хора за почасова работа.

Приложението покрива функционалните изисквания, следвайки стандартните на качество на кода. Идеята на този уеб базиран портал е именно да се улеснят взаимоотношенията на работодател със служител като се използва уеб среда и онлайн комуникация.

## **Бъдещо развитие**

За бъдещо развитие на приложението се очакват следните неща:

* Плащането да става през приложението, така че да може да се следи за коректността на потребителите.
* Подобряване на дизайна на приложението
* Чат среда за говорене между създателя на обявата и служителите

## **Използвана литература**

DZONE

<https://dzone.com/articles/configuring-spring-mvc-ajax>

Spring Framework Documentation

<https://docs.spring.io/spring/docs/3.0.x/spring-framework-reference/html/>

Spring Security

<https://spring.io/projects/spring-security>

MySQL

<https://www.mysql.com/>

JavaEE

<https://docs.oracle.com/javaee/6/tutorial/doc/bnbpz.html>

Hibernate

<http://hibernate.org/orm/>

Thymeleaf

<https://www.thymeleaf.org/doc/tutorials/2.1/thymeleafspring.html>

**Съдържание**

**[Увод](#_Toc3452574)** [3](#_Toc3452574)

**[ПЪРВА ГЛАВА](#_Toc3452575)** [4](#_Toc3452575)

**[1.](#_Toc3452576)****[МЕТОДИ И ТЕХНОЛОГИИ ЗА РЕАЛИЗИРАНЕ НА WEB ПРИЛОЖЕНИЯ](#_Toc3452576)** [4](#_Toc3452576)

**[1.1.](#_Toc3452577)****[Основни принципи, технологии и развойни среди за реализиране на web приложения](#_Toc3452577)** [4](#_Toc3452577)

**[1.1.1.](#_Toc3452578)****[Развойни средства и среди](#_Toc3452578)** [4](#_Toc3452578)

**[1.1.1.1.](#_Toc3452579)****[IntelliJ IDEA](#_Toc3452579)** [4](#_Toc3452579)

**[1.1.1.2.](#_Toc3452580)****[Eclipse](#_Toc3452580)** [5](#_Toc3452580)

**[1.2.](#_Toc3452581)****[Съществуващи подобни WEB проложения](#_Toc3452581)** [6](#_Toc3452581)

**[1.2.1.](#_Toc3452582)****[Jobs Tiger](#_Toc3452582)** [7](#_Toc3452582)

**[1.2.2.](#_Toc3452583)****[Zaplata.BG](#_Toc3452583)** [7](#_Toc3452583)

**[1.2.3.](#_Toc3452584)****[JOBS BG](#_Toc3452584)** [8](#_Toc3452584)

**[1.2.4.](#_Toc3452585)****[Buljobs.bg](#_Toc3452585)** [9](#_Toc3452585)

**[Втора глава](#_Toc3452586)** [10](#_Toc3452586)

**[2.](#_Toc3452587)****[Функционални изисквания към системата](#_Toc3452587)** [10](#_Toc3452587)

**[2.1.1.](#_Toc3452588)****[Управление на потребители:](#_Toc3452588)** [12](#_Toc3452588)

**[2.1.2.](#_Toc3452589)****[Управление на публикуваните обяви:](#_Toc3452589)** [12](#_Toc3452589)

**[2.1.3.](#_Toc3452590)****[Регистрация на потребител:](#_Toc3452590)** [13](#_Toc3452590)

**[2.1.4.](#_Toc3452591)****[Специално гласуване](#_Toc3452591)** [13](#_Toc3452591)

**[2.1.5.](#_Toc3452592)****[Търсене по обява](#_Toc3452592)** [13](#_Toc3452592)

**[2.2.](#_Toc3452593)****[Съображения за избор на програмни средства и развойната среда](#_Toc3452593)** [13](#_Toc3452593)

**[2.2.1.](#_Toc3452594)****[Избор на език за програмиране](#_Toc3452594)** [13](#_Toc3452594)

**[2.2.2.](#_Toc3452595)****[Избор на технологична рамка](#_Toc3452595)** [14](#_Toc3452595)

**[2.2.3.](#_Toc3452596)****[Библиотеки](#_Toc3452596)** [15](#_Toc3452596)

**[Logging Libraries](#_Toc3452597)** [15](#_Toc3452597)

**[2.3.](#_Toc3452598)****[Архитектура на приложението](#_Toc3452598)** [16](#_Toc3452598)

**[2.3.1.](#_Toc3452599)****[Spring MVC](#_Toc3452599)** [17](#_Toc3452599)

**[2.3.1.1.](#_Toc3452600)****[Spring MVC архитектура](#_Toc3452600)** [20](#_Toc3452600)

**[2.3.1.2.](#_Toc3452601)****[Spring’s Dispatcher Servlet](#_Toc3452601)** [21](#_Toc3452601)

**[2.3.1.3.](#_Toc3452602)****[Handler Mapping](#_Toc3452602)** [21](#_Toc3452602)

**[2.3.1.4.](#_Toc3452603)****[Controller](#_Toc3452603)** [21](#_Toc3452603)

**[2.3.1.5.](#_Toc3452604)****[View Resolver](#_Toc3452604)** [21](#_Toc3452604)

**[2.3.1.6.](#_Toc3452605)****[Registration View](#_Toc3452605)** [22](#_Toc3452605)

**[2.3.1.7.](#_Toc3452606)****[Index page View](#_Toc3452606)** [23](#_Toc3452606)

**[2.3.2.](#_Toc3452607)****[Hibernate](#_Toc3452607)** [23](#_Toc3452607)

**[2.4.](#_Toc3452608)****[Слоеве на приложението](#_Toc3452608)** [24](#_Toc3452608)

[2.4.1.](#_Toc3452609) **[Presentation Layer:](#_Toc3452609)** [24](#_Toc3452609)

[2.4.2.](#_Toc3452610) **[Controller Layer:](#_Toc3452610)** [24](#_Toc3452610)

**[2.4.3.](#_Toc3452611)****[Service Layer:](#_Toc3452611)** [25](#_Toc3452611)

**[2.4.4.](#_Toc3452612)****[Persistence layer](#_Toc3452612)** [25](#_Toc3452612)

**[2.5.](#_Toc3452613)****[Структура на базата данни](#_Toc3452613)** [27](#_Toc3452613)

**[2.5.1.](#_Toc3452614)****[Таблица Users](#_Toc3452614)** [27](#_Toc3452614)

**[2.5.2.](#_Toc3452615)****[Таблица Role](#_Toc3452615)** [27](#_Toc3452615)

**[2.5.3.](#_Toc3452616)****[Таблица Users\_Roles](#_Toc3452616)** [27](#_Toc3452616)

**[2.5.4.](#_Toc3452617)****[Таблица File\_Мodel](#_Toc3452617)** [27](#_Toc3452617)

**[2.5.5.](#_Toc3452618)****[Таблица Post](#_Toc3452618)** [28](#_Toc3452618)

**[2.5.6.](#_Toc3452619)****[Таблица Message](#_Toc3452619)** [28](#_Toc3452619)

**[2.5.7.](#_Toc3452620)****[Таблица Candidacy](#_Toc3452620)** [28](#_Toc3452620)

**[3.](#_Toc3452621)****[Реализация на проекта](#_Toc3452621)** [29](#_Toc3452621)

**[3.1.](#_Toc3452622)****[Създаване на POM](#_Toc3452622)** [29](#_Toc3452622)

**[3.1.1.](#_Toc3452623)****[Създаване на Classpath Dependencies](#_Toc3452623)** [29](#_Toc3452623)

**[3.2.](#_Toc3452624)****[Създаване на Micro-Services](#_Toc3452624)** [31](#_Toc3452624)

**[3.2.1.](#_Toc3452625)****[Връзка с базата данни](#_Toc3452625)** [31](#_Toc3452625)

**[3.3.](#_Toc3452626)****[Modeling Entities (Моделиращи единици)](#_Toc3452626)** [32](#_Toc3452626)

**[3.3.1.](#_Toc3452627)****[User](#_Toc3452627)** [32](#_Toc3452627)

**[3.3.2.](#_Toc3452628)****[Role](#_Toc3452628)** [33](#_Toc3452628)

**[3.3.3.](#_Toc3452629)****[Post](#_Toc3452629)** [34](#_Toc3452629)

**[3.3.4.](#_Toc3452630)****[Message](#_Toc3452630)** [35](#_Toc3452630)

**[3.3.5.](#_Toc3452631)****[Candidacy](#_Toc3452631)** [36](#_Toc3452631)

**[3.4.](#_Toc3452632)****[Релациите](#_Toc3452632)** [37](#_Toc3452632)

**[3.5.](#_Toc3452633)****[Репозитори](#_Toc3452633)** [38](#_Toc3452633)

**[3.6.](#_Toc3452634)****[Анотации](#_Toc3452634)** [39](#_Toc3452634)

**[3.7.](#_Toc3452635)****[Контролери](#_Toc3452635)** [40](#_Toc3452635)

**[3.8.](#_Toc3452636)****[Data Transfer Object](#_Toc3452636)** [42](#_Toc3452636)

**[3.9.](#_Toc3452637)****[Валидация](#_Toc3452637)** [42](#_Toc3452637)

**[3.10.](#_Toc3452638)****[Хеширане на паролите](#_Toc3452638)** [43](#_Toc3452638)

**[3.11.](#_Toc3452639)****[Конфигурации](#_Toc3452639)** [43](#_Toc3452639)

**[3.12.](#_Toc3452640)****[Spring модел атрибутите](#_Toc3452640)** [45](#_Toc3452640)

**[3.13.](#_Toc3452641)****[WebSecurityConfigurerAdapter](#_Toc3452641)** [47](#_Toc3452641)

**[3.14.](#_Toc3452642)****[Заявки на контролерите](#_Toc3452642)** [48](#_Toc3452642)

**[3.14.1.](#_Toc3452643)****[Get Заявки](#_Toc3452643)** [48](#_Toc3452643)

**[3.15.](#_Toc3452644)****[Post Заявки](#_Toc3452644)** [49](#_Toc3452644)

**[Четвърта глава](#_Toc3452645)** [50](#_Toc3452645)

**[4.](#_Toc3452646)****[Ръководство на потребителя](#_Toc3452646)** [50](#_Toc3452646)

**[Заключение](#_Toc3452647)** [55](#_Toc3452647)

**[5.](#_Toc3452648)****[Бъдещо развитие](#_Toc3452648)** [55](#_Toc3452648)

**[5.1.](#_Toc3452649)****[Използвана литература](#_Toc3452649)** [56](#_Toc3452649)