



ТЕХНИЧЕСКИ УНИВЕРСИТЕТ - СОФИЯ
Факултет Компютърни системи и технологии
Катедра „Компютърни системи“

КУРСОВ ПРОЕКТ

по

Програмиране за интернет на тема

**„Проектиране и разработка на уеб приложение за
намиране на работа“**

Студент:

Димитар Аврамов

Фак. номер: 121320051

Група: 223

Специалност: КСИ

Научен ръководител:

доц. д-р Антония Ташева

София, 2021

Съдържание

Съдържание.....	2
Увод.....	4
1. Анализ на темата. Преглед на подобни приложения. Цели и задачи.....	5
1.1 Анализ на темата.....	5
1.2 Преглед на подобни приложения	5
1.2.1 www.jobs.bg	5
1.2.2 www.bg.jooble.org.....	8
1.3 Цели и задачи.....	10
2. Структура (Архитектура) на приложението.....	11
2.1 Обща архитектура	11
2.2 Архитектура според MVC шаблона.....	12
2.3 Как Spring имплементира MVC	14
2.4 Архитектура на уеб приложението.....	16
2.5 Архитектура на Spring Data	17
2.6 Архитектура и механизъм за автентикация и авторизация	18
3 Схема на таблиците и релациите им в Базата данни.....	21
3.1 Таблица jobs	22
3.2 Таблица company	22
3.3 Таблица saved_job.....	23
3.4 Таблица logs	23
3.5 Таблица applications.....	23
3.6 Таблица users	23
3.7 Таблица user_roles	24
3.8 Таблица users_roles.....	24
3.9 EP Диаграма	25
4. Обяснение как са реализирани функционалностите, потребителските роли и др.	26
5. По-детайлно описание (наблягане) на по-интересните и нестандартни функционалности на проекта.....	30
5.1 Организация на потребителите	30
5.2 История на логове	31
5.3 Рейтване на дадена обява	33

6. Screenshot на уебсайта, представящ неговия дизайн.....	34
7. Линк към GitHub	38

Увод

Интернета като един глобален извор на информация все повече и повече става част от нашето ежедневие. Мощността, която той предлага стана основна и незаменима част от живота на всеки един човек. Уеб технологиите се развиват изключително бързо, предлагащи все по-напреднали услуги, качество и характеристики.

Идеята да се създават все по-добри и по-полезни сайтове е от голямо значение. Шансът да се улесни, подобри или замени някой процес от реалността е очакван с широко отворени ръце от страна на хората.

В миналото време когато един човек е тръсил работа, той ходил от компания до компания за да пита за свободни позиции, се оглеждал по вестници, разпитвал свои познати итн. В момента тези неща все още съществуват, но са леко заместени с по-бързи и по-ефикасни такива. Един от най-големите заместници е интернетa. Там вече съществуват огромен брой на софтуерни решения които свързват хората и компаниите. С това те в голяма мера потпомагат за свързване на двете страни. Уникалното решение на повечето такива сайтове позволява с почти един клик да си намериш бъдещата работна позиция.

Относно този проект са разгледани повече вече съществуващи софтуерни решения от които научаваме на кои детайли най-много да наблегнем. В същото време позволяват ни да учим от тяхните грешки и да имплементираме тяхните недостатъци.

Като цяло проекта е все още в почетна фаза и се очаква в продължение да се имплементира в много по-сериозни фази и да обхваща повече детайли.

1. Анализ на темата. Преглед на подобни приложения. Цели и задачи.

1.1 Анализ на темата

С течение на годините технологиите се развиват ежедневно все повече и повече. Броят на софтуерните продукти нараства непрекъснато. Нови и нови системи излизат във всеки един момент, с което даже и умишлено добре направени статистики, които да анализират бройката на нови продукти, трудно би се съобразили за точната бройка. С всеки нов проект, с всяка нова функционалност, идеите стават все по-големи и все по-примамливи. Пазарът е винаги гладен за нови възможности. Хубавите намерения да се създаде продукт който улеснява, помага или върши голяма работа на определен брой хора е добре дошъл в света, който с всеки нов ден е все по-напред и по-напред. Същността на продукта се намира в това, какво искаме да създадем и на кои хора може да „улесним живота“. Така, в последните няколко години според мои лични наблюдения все повече тези софтуерни продукти са предоставяни чрез уеб технологиите. Възможностите на интернет стават все по-огромни, даже можем да кажем че без интернет сме „никой и нищо“. В последно време дори и изискванията на десктоп приложения намаляват и се заменят с такива в уеб платформа.

Нашия уеб продукт не е нещо съвсем ново в днешното ежедневие. Неговата идея е да наблегне върху изискванията на потребителя. С това нашия фокус е да имаме все повече доволни клиенти които непрекъснато ще ползват и препоръчват това приложение.

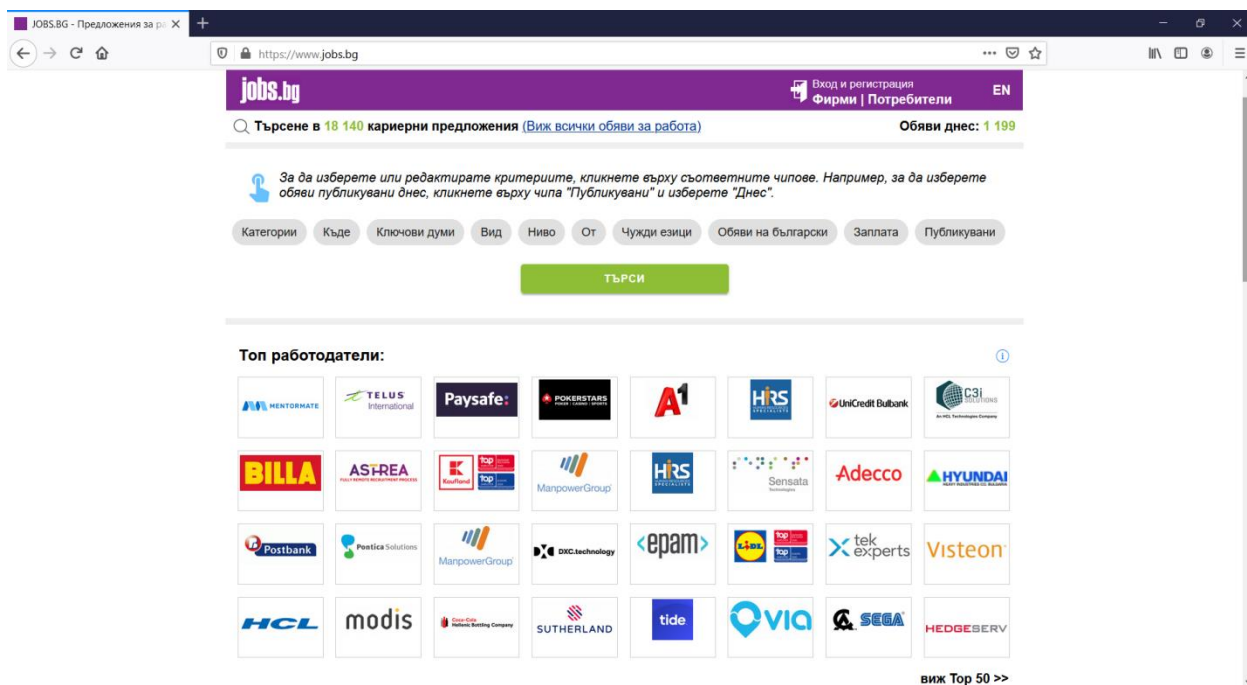
1.2 Преглед на подобни приложения

1.2.1 www.jobs.bg

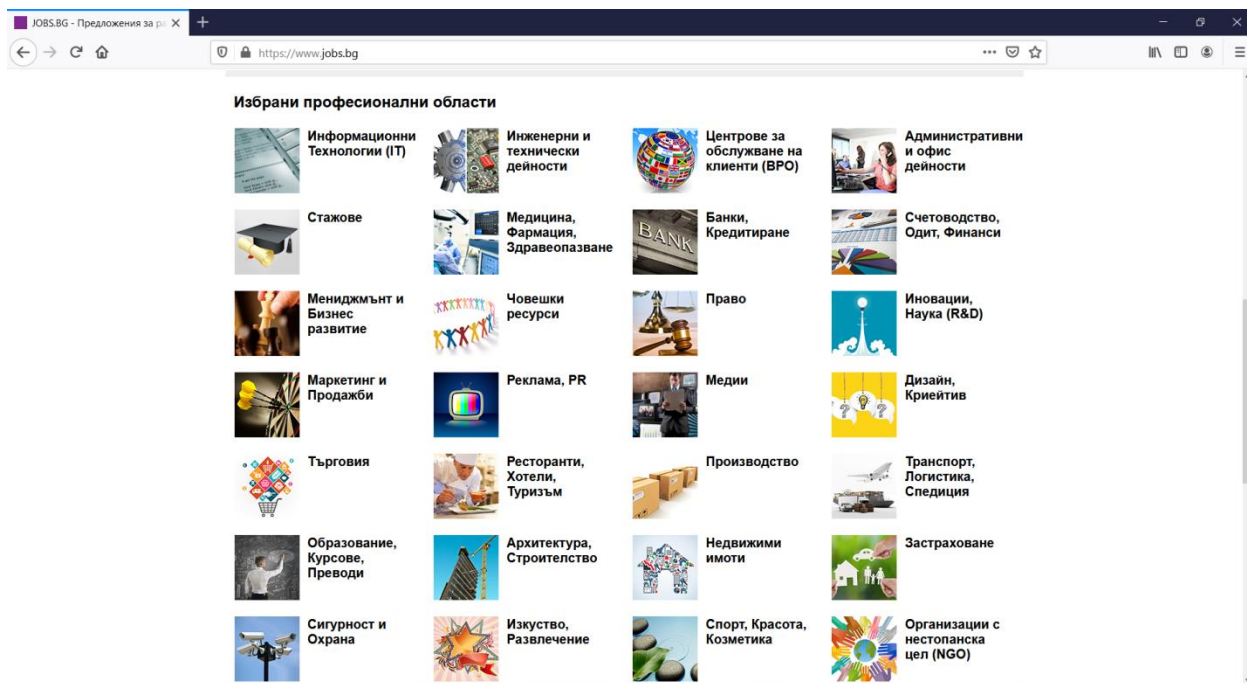
jobs.bg е главния сайт в България където може да се намерат огромен брой на отворени позиции за работа. Уеб приложението предоставя пълна информация за всичките особености на дадена работа.

Част от основните функционалности на приложението са:

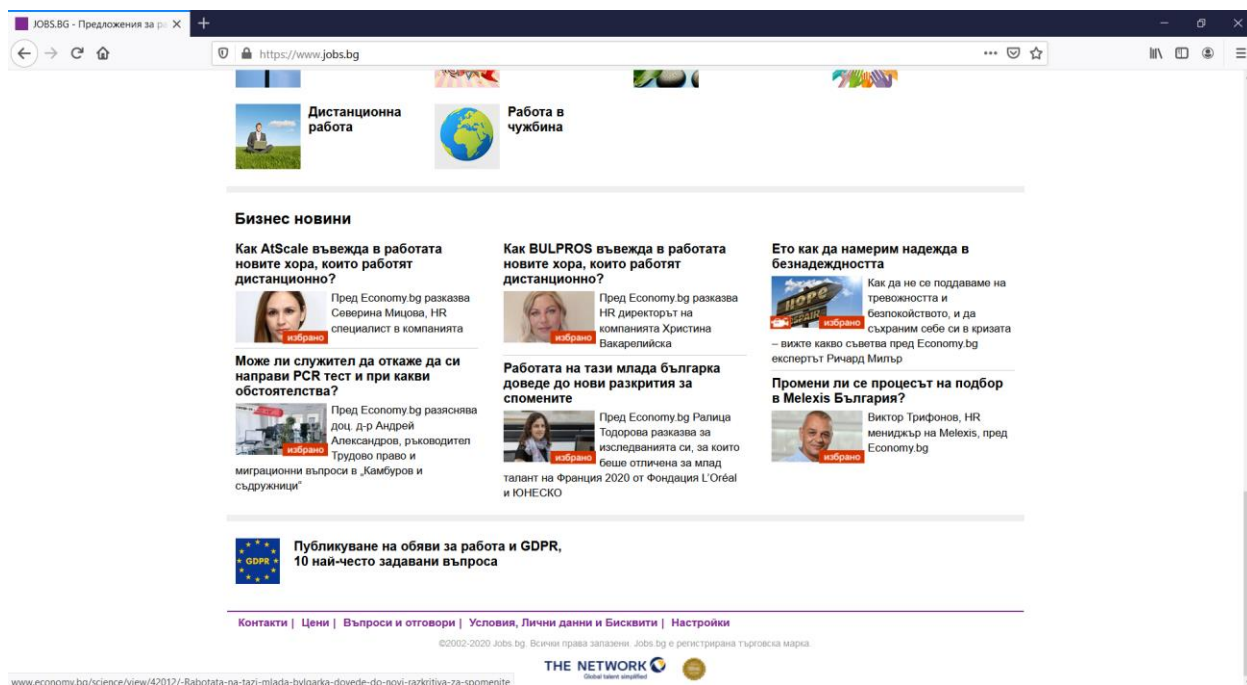
- Преглед на топ работодатели(фиг. 1.1)
- Преглед на избрани професионални области(фиг. 1.2)
- Бизнес новини(фиг. 1.3)
- Преглед на отворени позиции и съществуващи филтри(фиг. 1.4)
- Тръсене на отворена позиция или компания(фиг.1.5)



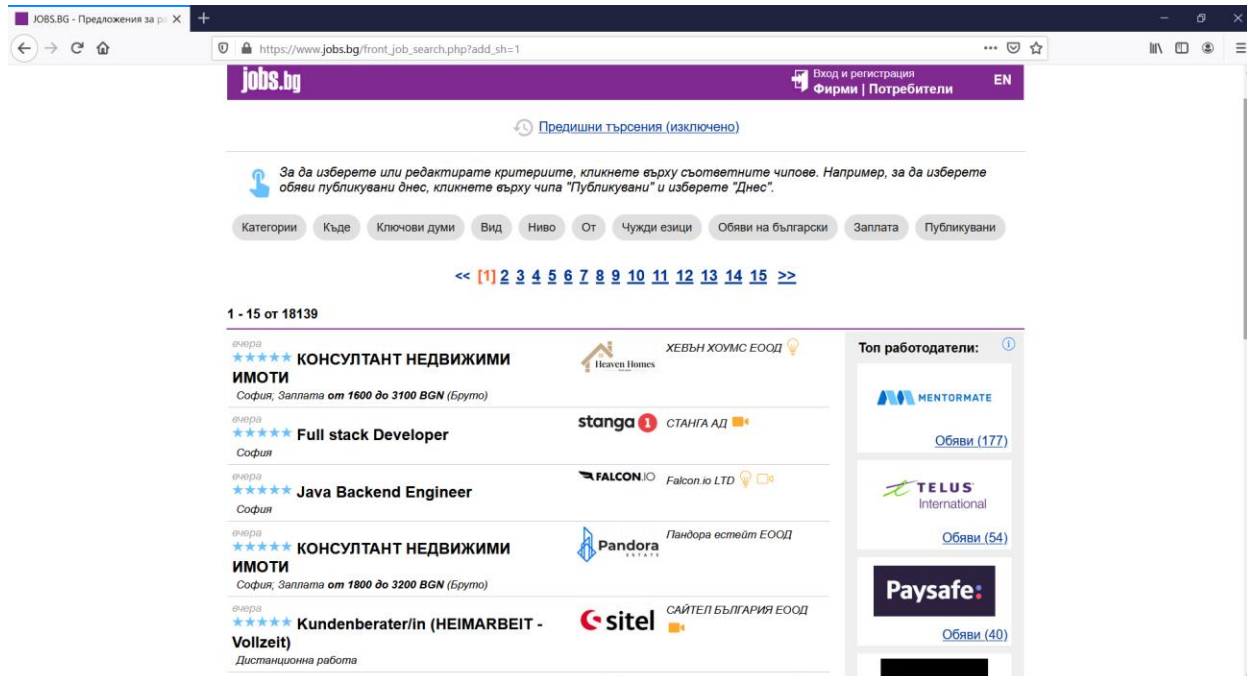
1.1 – Топ работодатели



1.2 - Преглед на избрани професионални области



1.3 – Бизнес новини



1.4 – Отворени позиции

Положителни страни на приложението:

- Много фийчъри са имплементирани
- Доста детайли са запазени
- Широк обхват на нещата
- Бърз при ползване
- Смяна на езика BG-EN
- Ясно разредени секции в сайта

Негативни страни на приложението:

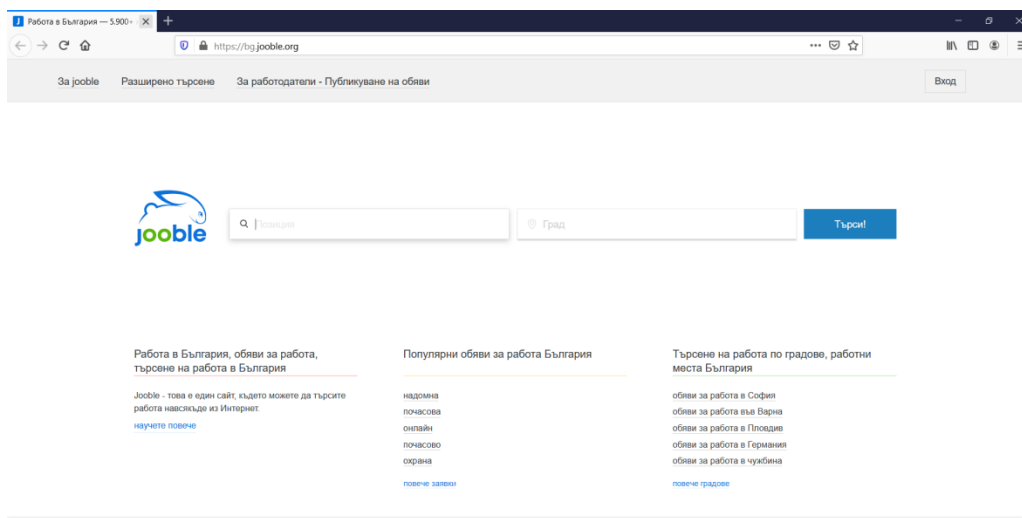
- Не толкова добър графичен интерфейс
- Не многу удобен за ползване
- Изглежда малко сложен

1.2.2 www.bg.jobble.org

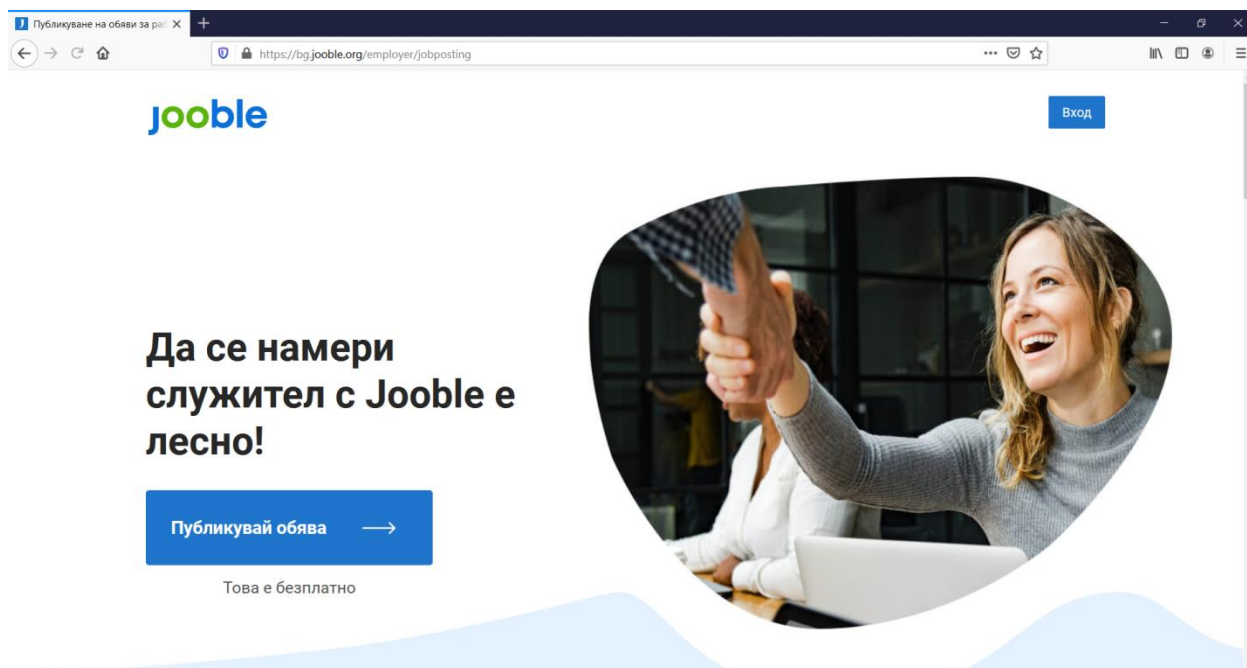
Jooble е световна организация за тръсене на работа. Има я в 71 земя, а е базирана в Киев, Украйна. Предоставя доста прост и хубав интерфйес който ни помага много лесно да намерим това което тръсим.

Част от основните функционалности на приложението са:

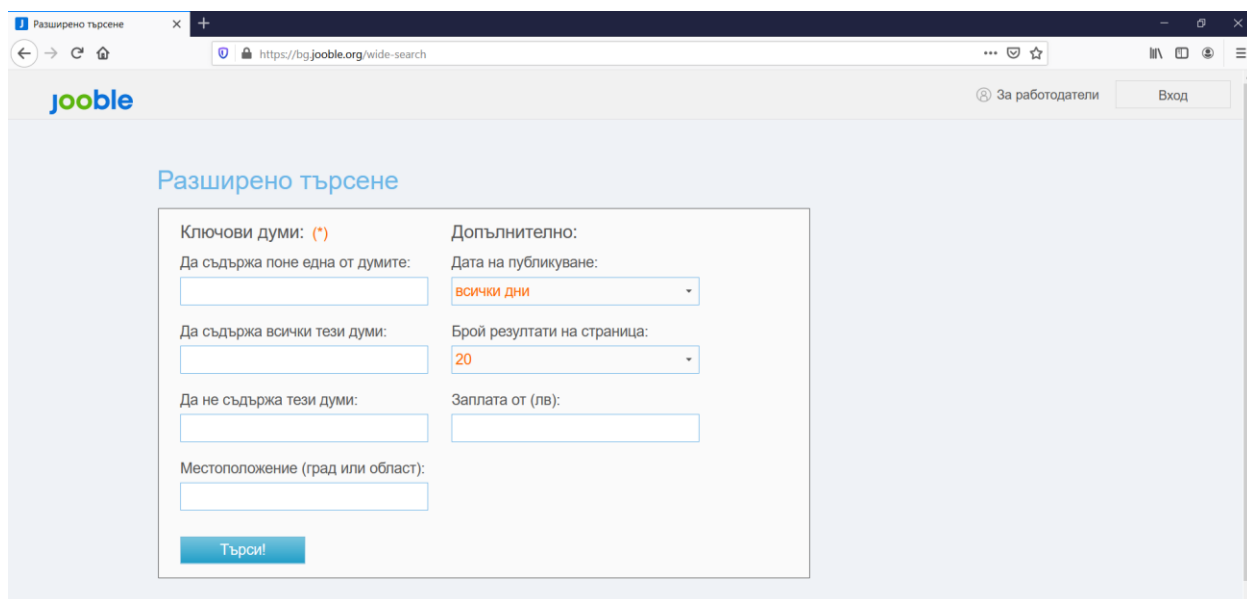
- Тръсене на позиция
- Публикуване на обява за работа
- Разширено тръсене
- Абонат за бюлетина с нови обяви за работа



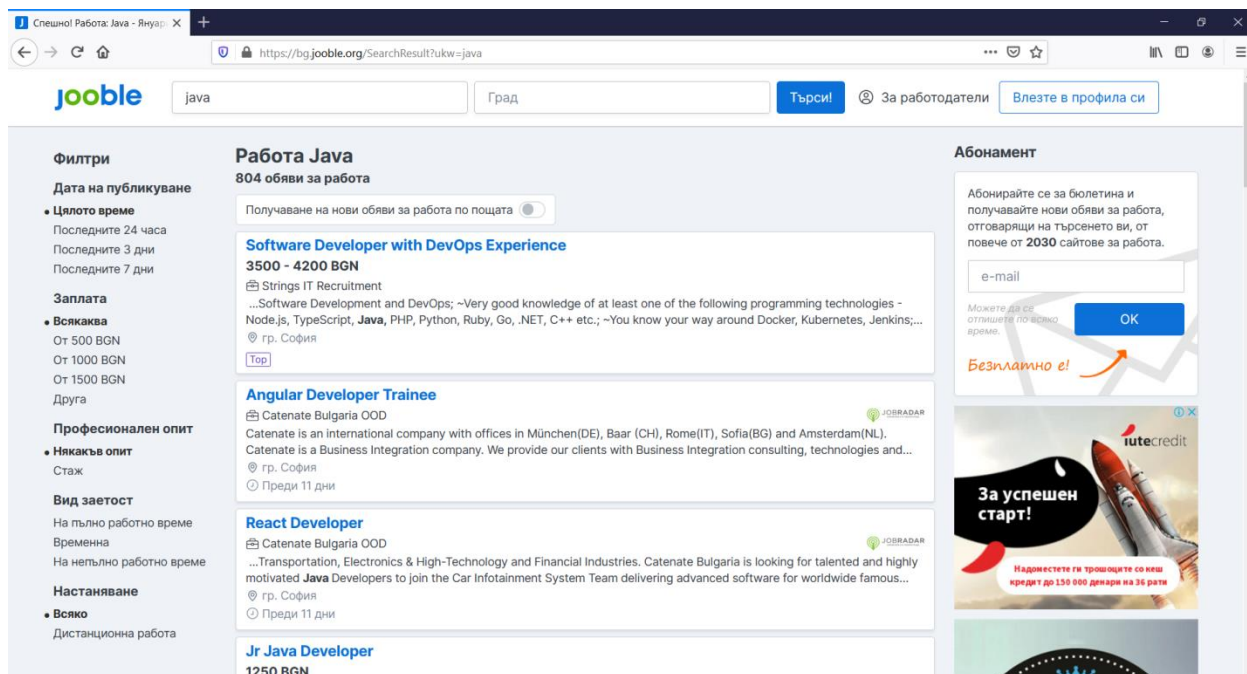
Фиг. 1.5 – Начална страница – приятен графичен интерфейс



Фиг. 1.6 – Публикуване на обява



Фиг. 1.7 – Разширено търсене



Фиг. 1.8 – Резултат от тръсене

Положителни страни на приложението:

- Хубав графичен интерфейс
- Удобен и приятен начин на ползване
- Прост и лесен за разбиране
- Емаил абониране

Негативни страни на приложението:

- Бавен при пренасочването към определена обява
- Малък брой на фийчър

1.3 Цели и задачи

Както споменахме идеята на този проект е да се направи един уникален уеб сайт в който хората ще може да си тръсвят работа, а същевременно работодателите да могат да добавят обяви. Като главната цел е да се базираме върху самите потребители на сайта като цяло. За това, всяка част от сайта трябва да е ясно дефинирана и подредена. Клиента трябва да има възможността супер лесно да достъпва всички функционалности които сайта предлага за него. Същевременно, проекта трябва да притежава силно изградена архитектура която ще ни овъзможата ясна структура и лесен подход при бъдещи

ъпъдейти. При софтуерната имплементация трябва да се спазват Java конвенциите за еднотипно реализиране на кода и неговата подреденост. Да се определи подходяща база данни за сторване на цялата информация отдолу. Да се реализира сигурността на приложението. Да се осигури защита от неправомерен достъп и загуба на данни.

За реализиране на целта на проекта също така си задавам и следващите задачи:

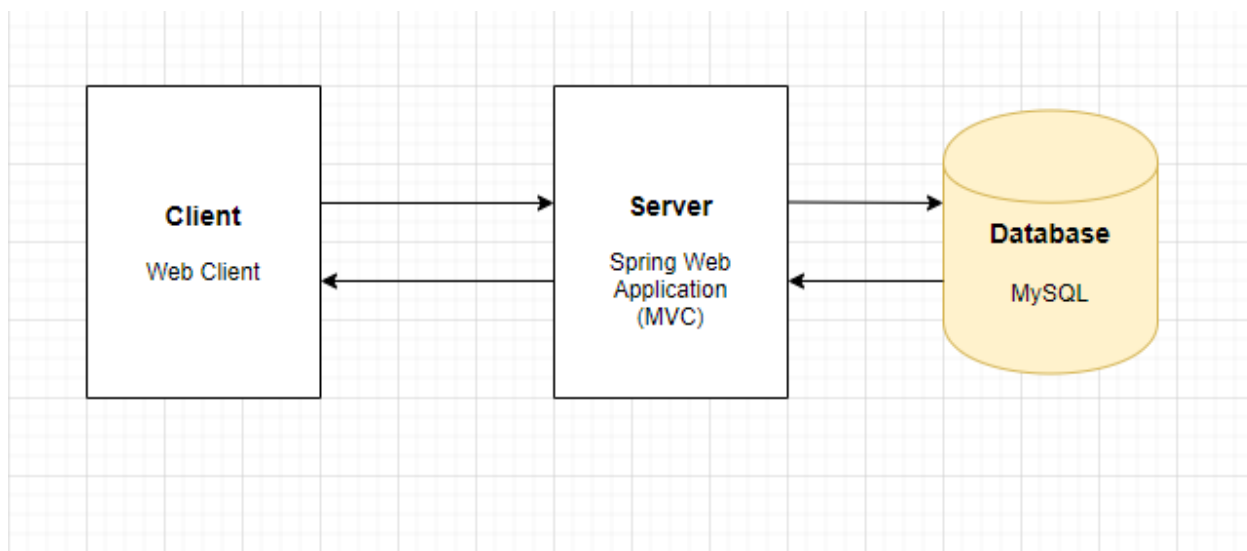
- Възможност за регистриране на потребители
- Възможност за логване в сайта(автентикация)
- Възможност за помнене на логнатия акаунт
- Автентикация на потребителите с потребителско име и парола
- Авторизация на потребителите относно дадената роля
- Сигурност на профилите
- Достъпване само до определени ресурсите от страна на посетителите които не са логнати
- Да се предостави точната функция на потребителя относно неговата роля и нищо повече
- Бързо зареждане помежду отделните страници
- Хубав потребителски интерфейс
- Удобно и лесно за ползване
- Системата да работи успешно с голям набор от информации
- Бърз и лесен достъп до информацията
- Красив feedback за нашите потребители
- Качествени функционалности
- Обективност на публикациите

2. Структура (Архитектура) на приложението

2.1 Обща архитектура

Архитектурата е най-важната част от всяко приложение. Без нея проекта ще е неподреден, труден за разбиране и поддръжка. За тази цял приложението за намиране на работа се базира върху MVC(Model-View-Controller) шаблона който е един от най-популярните и четки структури. Съставен е от модели които реално са класове които реализират таблиците в базата данни. Изгледите които презентират дадените модели на клиента в уеб сайта и контролери които приемат информацията от потребителите и я препращат към моделите (като разбира се тази информация се рендерира чрез сервизите

които са помежду тях). Общо зето, приложението е разделено на две части, front-end и back-end. Front-end-а играе ролята на изгледа - показване на данните получени от бек-енда и изпращане на данни към бек-енда обратно. Докато back-end-а представлява сървър, който процесира данните до базата данни и имплементирайки вътрешната логика на приложението. Тази комуникация се осъществява чрез основните заявки GET, POST, PUT, PATCH, DELETE от които за целите на това уеб приложение ще се ползват основно GET и POST. Глобалната архитектура е показана на фигура 2.1.



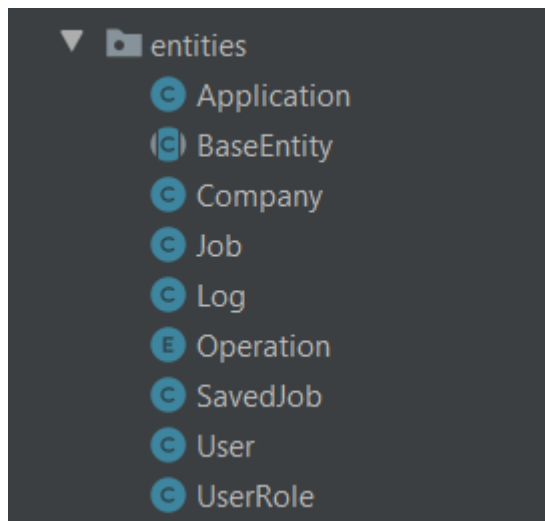
Фиг. 2.1 – Глобална архитектура

2.2 Архитектура според MVC шаблона

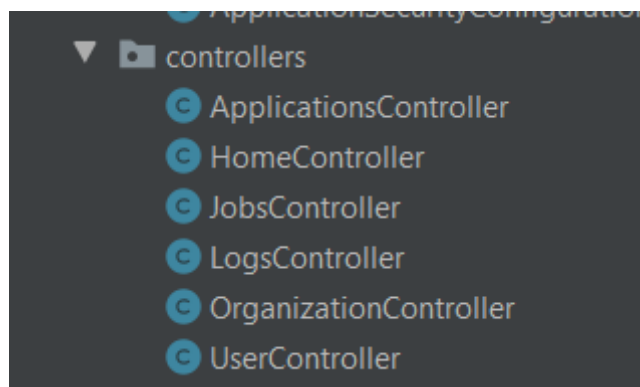
Модел-Изглед-Контролер (Model-View-Controller или MVC) е архитектурен шаблон за дизайн в програмирането, основан на разделянето на бизнес логиката от графичния интерфейс и данните в дадено приложение. За пръв път този шаблон за дизайн е използван в програмния език Smalltalk. Подобни шаблони на MVC са MVVM(Model-View-View-Model), MVP(Model-View-Presenter), MVT(Model-View-Template).

1. Model – това е частта която представя ядрото с данни. Това са обектите които представляват нещо от „реалния свят“. Тези обекти за основните информации кои са директно свързани с таблиците от базата данни чрез използване на класовете от обектно-ориентираното програмиране. Всеки клас си има свои атрибути, методи и конструктори. Модела като една основа на шаблона стои най-отдолу на шаблона MVC и е междинен слой помежду външната база данни и контролера. Неговите данни се взимат от по-горните слоеве, примерно контролера, които ги обработва и праща на view частта. Също така view частта чрез потребителски интерфейс може да промени тези модели които обратно ще се препратят към контролера който с помощ от своите пот-слоеве ще промени

информацията в определените модели и съответно ще се запишат в базата данни. В нашия проект моделите ще се намират в пакета entities. На фигура 2.2 се виждат основните класове които представляват M(Model) от шаблона MVC(Model-View-Controller) и същевременно представляват таблиците в нашата база от данни.



Фиг. 2.2 – Модели

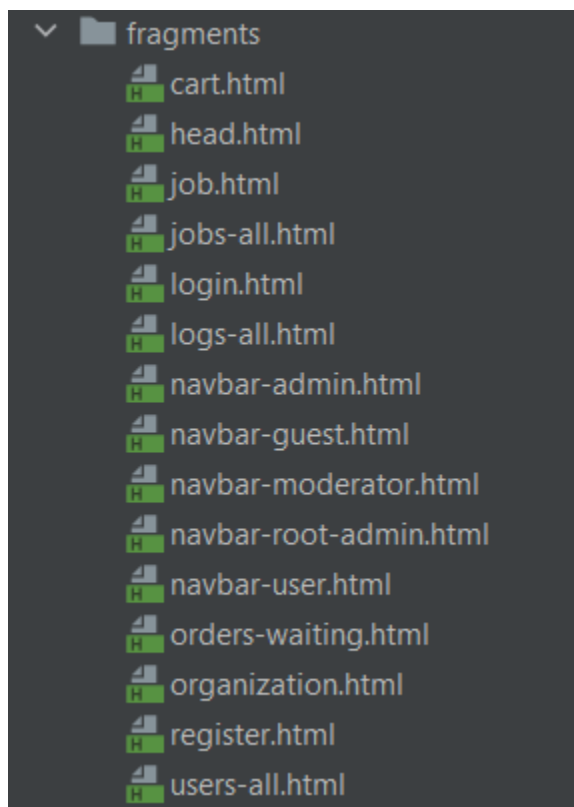


Фиг. 2.3 – Контролери

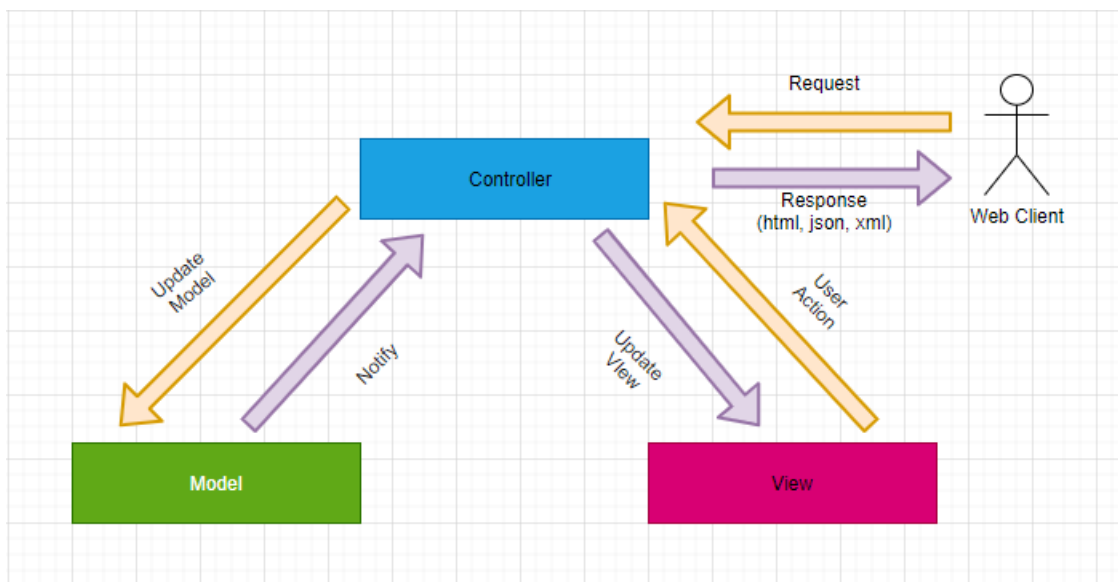
2. **Controller** – секцията която хендълва поискането на клиентите към определена функционалност на приложението с помощ на GET и POST заявки. При тази интеракция контролера актуализира моделите и не води към някой друг изглед(view). Разбира се в неговата част той си има пот-слоеве(примерно services, repository и т.н) които ще разгледаме по-надолу.

В нашия проект контролерите ще се намират в пакета controllers (фиг. 2.3). Ще имаме повече контролери, тъй като искаме да реализираме повече страници и функционалности които да са достъпни за потребителя.

3. **View** – това е частта в която трябва да визуализираме данните от модела. Това е последната част от MVC шаблона която сервира на потребителя резултата от информацията които ги поискал. Тези данни пред да стигнат до изгледа са минали от модела който ги е взел от базата данни, през контролера за на край да се визуализират в някой от view-та. Изгледите в нашия проект са рендерирани с помощта на JavaScript езика който показва процесиращите view-та.



Фиг. 2.4 - Изгледи

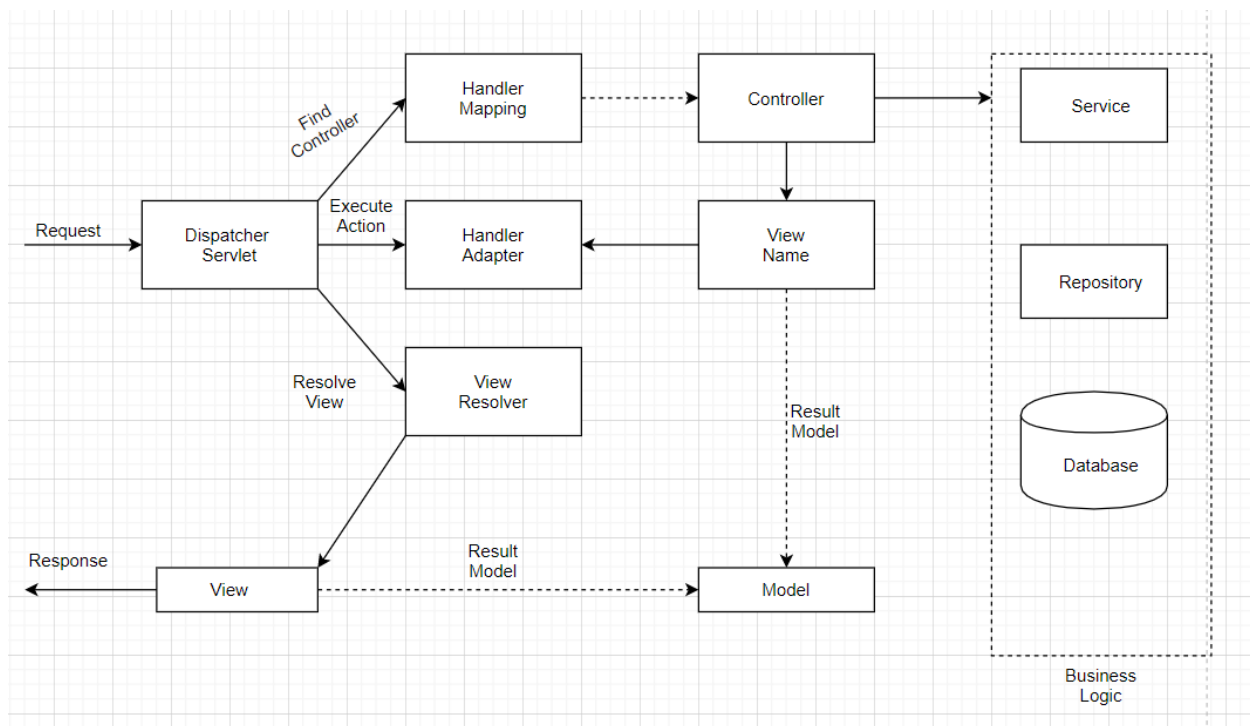


Фиг. 2.5 – MVC - Control Flow

2.3 Как Spring имплементира MVC

MVC framework-а в Spring работи върху т.н Dispatcher сервлета който се деплойма в Каталина или Томкат. Когато дойде един request той първо отива до диспечер сервлета.

Диспечер сервлета е конфигуриран така, че да хваща абсолютно всички request-и. В момента в който намери request той търси контролер. Когато намери контролер търсим дали този контролер има мапинга който като цяло ни е метода който ще се изпълни или т.н контролер action. В момента в който го намерим вземаме контролера, той съответно стартира услуги, репозитория и всичко, което му трябва (всичките му депенденси) и вътре в него изпълнява бизнес логиката. На края след извършване на логиката имаме някакво view, което ни връща хендлер. Този хендлер съответно може в себе си да извършва някакви действия (примерно може да е логин форма, която праща POST request). След извършване на действията трябва всичко да се събере на едно и да се върне като хендлер на диспечер сервлета. В момента, в който тези неща се връщат върху диспечер сервлета изпълнява action-а, който е пратил потребителя. Изпълнява го върху хендлера, прави някакво view или някакъв resolver, който ни дава view-то. Това нещо отива до view-то, нещата се комбинират в един модел (който контролера го пълни) и дава възможността за тези неща да могат да се рендерират в view-то. На края всичките тези неща, комбинирани в едно цяло, се изпращат като response.

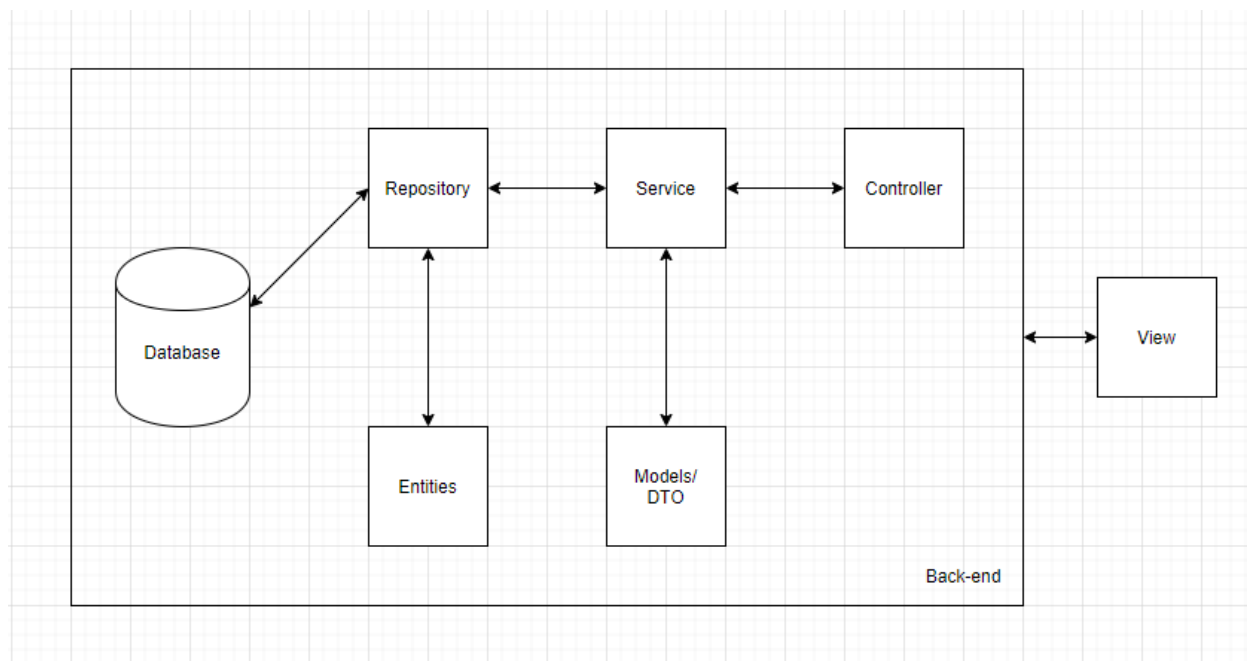


Фиг. 2.6 – Имплементация на MVC в Spring

2.4 Архитектура на бек-енда

Архитектурата на уеб апи-то, гледана само в бек-енда се състои от следващите компоненти:

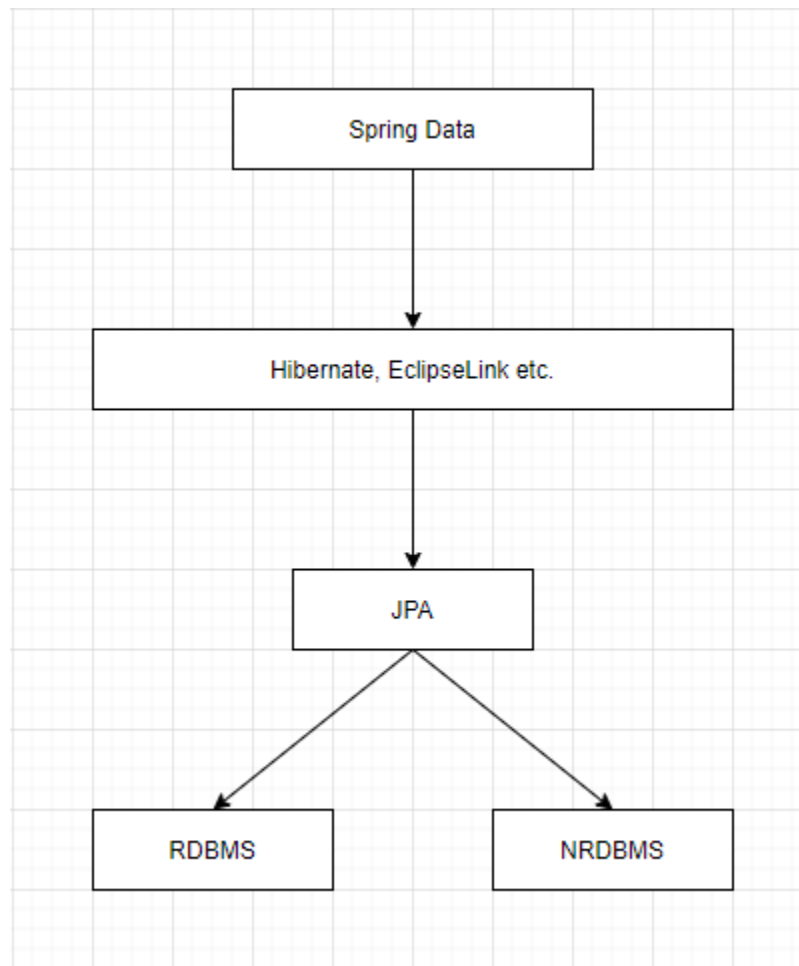
- База от данни – място където ще се съхраняват всичките необходими данни.
- Repository – компонент който ни служи за директна връзка с данните в базата от данни. Тука се очаква да се случва CRUD логиката – добавяне, ъпдейтване, триене, селектиране и всичките спецификации на query-тата които базата данни ни овъзможавя. За по-лесно имплементиране на тези задачи, Spring framework-а предоставя свой компонент наречен Spring Data с който може много по-едноставно да се реализират query-тата.
- Entities – класове които представляват реалните обекти с които ще работим и които ще са запазени директно в базата данни във вида на таблици
- Service – слой който не е задължителен, обаче е добре да съществува за по-ясна и по-качествена архитектура. Той е инжектива Repository-то и в него трябва да се имплементира цялата бизнес логика на приложението. Описване на почти всичките методи на приложение: намиране на топ 3 автори на добри дела за текущия месец, регистрирация на потребител, постване на публична информация и т.н. Сервизите трябва да са инжектнати в контролери през които да им се извикват разните методи. Всеки метод има обръщане към базата от данни и съответните манипулации с данните в нея чрез предишния компонент Repository.
- Models – Компонент който представлява модела (M) от MVC шаблона. Те са същите класове от пакета entities, обаче са създадени за да се ползват в сервизите с цял да не се работи директно върху ентитита които реално са таблиците в базата данни
- Controller – Контролерите са класовете които се занимават с URL request-ите. Техните методи са наконфигурирани с определен мапинг. Съответно, те чакат някакво поискане на тоз мапинг от страна на потребителя, за да извикат съответния метод. Във тях се очаква да бъде включен отреден обект от сервизите чрез който да се извикват методи които са нужни за съответния request. Във тях не трябва да бъде реализирана никаква бизнес логика. Там трябва да има само валидация на данните. Когато свършат с обработката на данни контролерите трябва да връщат определено view което може да бъде html страница или определен отговор(response).



Фиг. 2.7 – Архитектура на уеб приложението

2.5 Архитектура на Spring Data

Spring Data е компонент на Spring framework-а който добавя един допълнителен слой над JPA провайдера. Той надгражда JPA, Mongo и друг. Spring Data намира приложение в компонента Repository. С негова помощ с много малко писане на код получаваме генериран код. Това помага, с това че много лесно може да манипулираме с данните, които са налични в базата от данни.



Фиг. 2.8 – Spring Data

2.6 Архитектура и механизъм за автентикация и авторизация

Процесът на проверката дали потребителското име съвпада с паролата (автентикация) се състои в изпращане на потребителско име и парола от уеб сайта към сървъра, валидиране на данните на сървъра с проверката на съществуващите потребители в базата данни и при успешна валидация се преминава към началната страница на приложението. В противен случай се изписва уведомление за грешно потребителско име или парола. (Фиг. 2.9).

Тези неща са направени с помощта на Spring Security компонента на Spring framework-a който представя една добре структурирана имплементация която изцяло се занимава с потребителите и тяхната сигурност. Вътрешно конфигурираме stateless автентикация JWT(JSON Web Token). В конфигурацията имаме филтър за автентикацията и авторизацията показани на фигура 2.9 и 2.10.

```

public class JwtAuthorizationFilter extends BasicAuthenticationFilter {
    private UserService userService;

    public JwtAuthorizationFilter(AuthenticationManager authenticationManager,
        UserService userService) {
        super(authenticationManager);
        this.userService = userService;
    }

    @Override
    protected void doFilterInternal(HttpServletRequest request, HttpServletResponse
        response, FilterChain chain) throws IOException, ServletException {
        String header = request.getHeader("Authorization");
        if(header == null || !header.startsWith("Bearer ")) {
            chain.doFilter(request, response);
            return;
        }
        UsernamePasswordAuthenticationToken token = this.getAuthentication(request);
        SecurityContextHolder.getContext().setAuthentication(token);
        chain.doFilter(request, response);
    }

    private UsernamePasswordAuthenticationToken getAuthentication(HttpServletRequest
        request) {
        String token = request.getHeader("Authorization");
        UsernamePasswordAuthenticationToken usernamePasswordAuthenticationToken =
        null;

        if (token != null) {
            String username = Jwts.parser()
                .setSigningKey("Secret".getBytes())
                .parseClaimsJws(token.replace("Bearer ", ""))
                .getBody()
                .getSubject();

            if (username != null) {
                UserDetails userData = this.userService
                    .loadUserByUsername(username);

                usernamePasswordAuthenticationToken
                    = new UsernamePasswordAuthenticationToken(
                        username,
                        null,
                        userData.getAuthorities()
                    );
            }
        }
        return usernamePasswordAuthenticationToken;
    }
}

```

Фиг. 2.9 – Филтър за авторизация

```

public class JwtAuthenticationFilter extends UsernamePasswordAuthenticationFilter {
    private final AuthenticationManager authenticationManager;

    public JwtAuthenticationFilter(AuthenticationManager authenticationManager) {
        this.authenticationManager = authenticationManager;
    }

    @Override
    public Authentication attemptAuthentication(HttpServletRequest request,
        HttpServletResponse response) throws AuthenticationException {
        try {
            UserLoginBindingModel loginBindingModel = new ObjectMapper()
                .readValue(request.getInputStream(),
                    UserLoginBindingModel.class);

            return this.authenticationManager.authenticate(
                new UsernamePasswordAuthenticationToken(
                    loginBindingModel.getUsername(),
                    loginBindingModel.getPassword(),
                    new ArrayList<>())
            );
        } catch (IOException ignored) {
            return null;
        }
    }

    @Override
    protected void successfulAuthentication(HttpServletRequest request,
        HttpServletResponse response, FilterChain chain, Authentication authResult) throws
        IOException, ServletException {
        User user = ((User) authResult.getPrincipal());

        String authority = null;

        if(user.getAuthorities().stream().filter(o -> o.getAuthority().equals("ROOT-
ADMIN")).findFirst().isPresent() == true)
        {
            authority = "ROOT-ADMIN";
        }
        else if(user.getAuthorities().stream().filter(o ->
o.getAuthority().equals("ADMIN")).findFirst().isPresent() == true)
        {
            authority = "ADMIN";
        }
        else if(user.getAuthorities().stream().filter(o ->
o.getAuthority().equals("MODERATOR")).findFirst().isPresent() == true)
        {
            authority = "MODERATOR";
        }
        else if(user.getAuthorities().stream().filter(o ->
o.getAuthority().equals("ROLE_USER")).findFirst().isPresent() == true)
        {
            authority = "ROLE_USER";
        }
    }
}

```

```

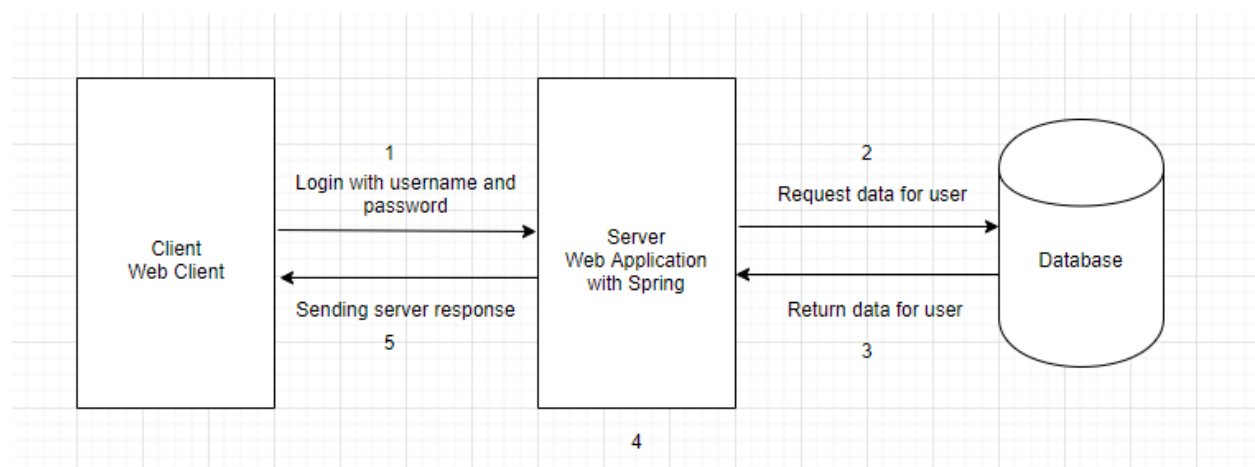
String token = Jwts.builder()
    .setSubject(user.getUsername())
    .setExpiration(new Date(System.currentTimeMillis() + 12000000))
    .claim("role", authority)
    .signWith(SignatureAlgorithm.HS256, "Secret".getBytes())
    .compact();

response.getWriter()
    .append("Authorization: Bearer " + token);

response.addHeader("Authorization", "Bearer " + token);
}
}

```

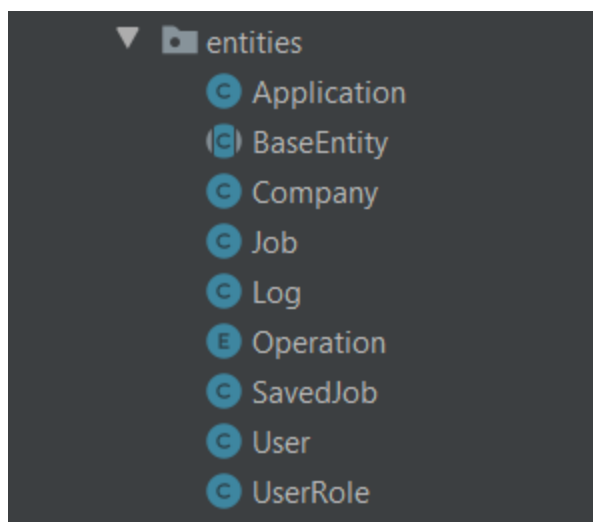
Фиг. 2.10 – Филтър за автентикация



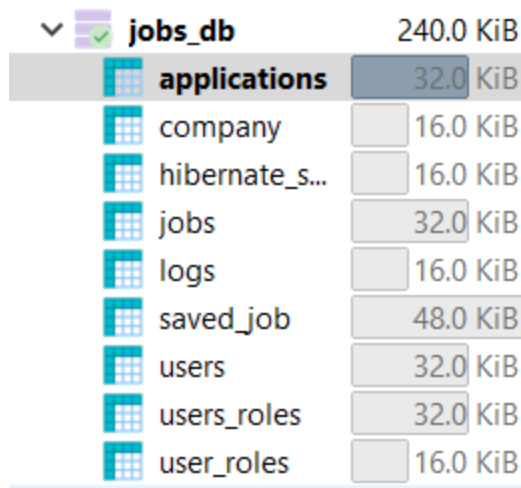
Фиг. 2.11 – Автентикация и Авторизация на потребител

3 Схема на таблиците и релациите им в Базата данни

Като база данни в този проект ползваме MySQL която е доста удобна и лесна за разбиране. Нашата таблица е наречена **jobs_db**. Всичките таблици са ни пряко свързани с класовете които имат конфигурация @Entity(M от MVC) в пакета entities.



Фиг. 2.12 – Ентитита



Фиг. 2.13 – Реалните таблици

3.1 Таблица jobs

Тази таблица ще съдържа цялата информация която се отнася на секцията за работа

Име на колона	Тип	Описание
id(PRIMARY_KEY)	int	ID на работата
description	varchar	Описание
likes	int	Харесвания
location	varchar	Локация на фирмата
num_of_employees	int	Номер на служители
position	varchar	Позиция
rating_sum	double	Рейтинг
salary	double	Заплата
times Rated	int	Колко пъти е рейтната
company_id	int	От коя компания е обявата
img1	varchar	Снимка 1
img2	varchar	Снимка 2

3.2 Таблица company

Име на колона	Тип	Описание
id(PRIMARY_KEY)	int	ID на компанията
name	varchar	Име на компания

3.3 Таблица saved_job

Име на колона	Тип	Описание
id(PRIMARY_KEY)	int	ID на любимата работа
jobs_id	int	ID на работата
user_id	varchar	ID на потребителя

3.4 Таблица logs

Име на колона	Тип	Описание
id(PRIMARY_KEY)	varchar	ID на лога
date	varchar	ID на датума
operation	varchar	ID на операцията
user	varchar	Потребителя който креирал лога
table_name	varchar	В коя таблица е направена модификацията

3.5 Таблица applications

Име на колона	Тип	Описание
id(PRIMARY_KEY)	int	ID на лога
city	varchar	Град
delivered	bit	Дали е приета апликацията от модератора
description	varchar	Описание
email	varchar	Е-маил
name	varchar	Име
number	varchar	Брой
job_id	int	С коя обява за работа се свързва

3.6 Таблица users

Таблица която съдържа всичките потребители и информацията необходима за един потребител във уеб или друго подобно приложение.

Име на колоната	Тип	Описание
id(PRIMARY_KEY)	varchar	UUID на потребителите
username	varchar	Потребителско име
password	varchar	Парола на потребителя
email	varchar	Емайл на потребителя
enabled	bit	Дали акаунта е активиран
account_non_expired	bit	Дали акаунта не е стар
account_non_locked	bit	Дали акаунта не е локнат
credentials_non_expired	bit	Дали крeденшлите не са стари

Колоните от типа на дали е акаунта е активиран, локнат или изминат срок на потребителя са допълнителни проверки върху нашите клиенти. За момента, в нашия проект всичките тези ще са true, което ще ни означава те ще винаги ще са активни, никога няма да бъдат локнати или с изминал срок.

3.7 Таблица user_roles

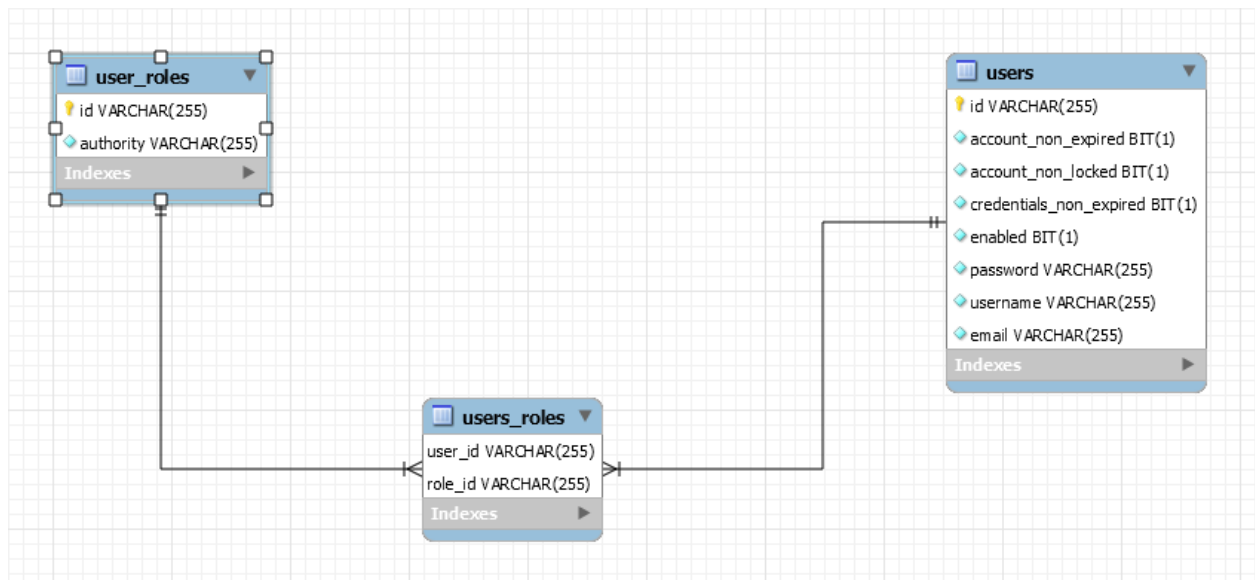
Допълнителна таблица, която ще дефинира различните роли на потребителите. По принцип, тези роли могат да са включат като String(varchar) в таблицата на user-ите. Обаче, тази логика е създадена за по-добро проектиране на базата данни. С това ние се предпазваме и намаляваме си проблемите в бъдещето. Един пример за това е ако след известно време искаме да променим всички роли от тип модератор към роля админ или някоя друга, да не ходим във всички редове от данни за потребители и навсякъде променяме колоната за ролята. При голям брой на потребители, това ще ни донесе големи проблеми и губене на време.

За това, прилагаме нова таблица която да описва отделните роли и с това имплементираме част от функционалното изискване което гласи – **нормализация на базата данни**.

Име на колона	Тип	Описание
id(PRIMARY_KEY)	varchar	UUID на ролята
authority	varchar	Име на ролята

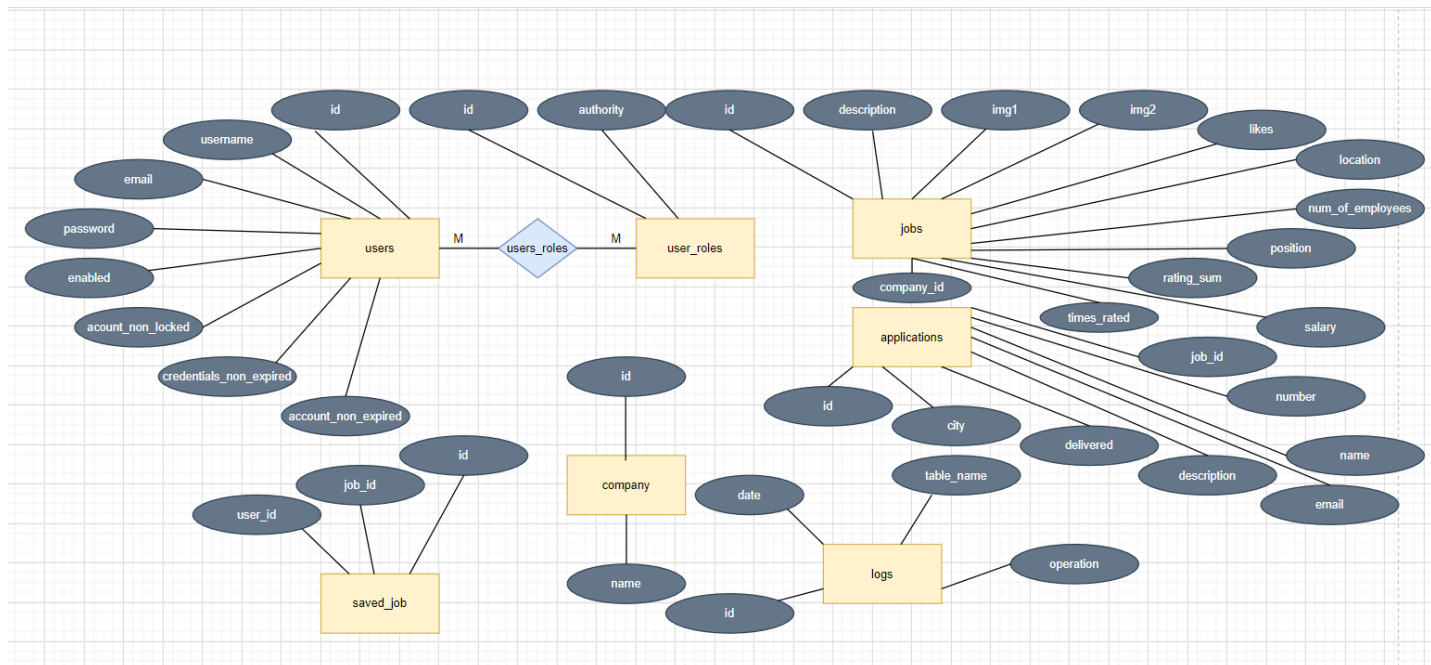
3.8 Таблица users_roles

На края трябва по някакъв начин да свържем таблицата с потребителите(users) и таблицата с ролите(user_roles). Тази връзка ще бъде ManyToMany защото един потребител може да има повече роли, също така и една роля може да бъде присвоена на повече потребители.



Фиг. 3.14 – Many-To-Many Relationship

3.9 ER Диаграма



Фиг. 3.15 – ER Диаграма на базата данни

4. Обяснение как са реализирани функционалностите, потребителските роли и др.

Реализацията на основните функционалности е направена с помощта на езиците: Java(за бек-енда), JavaScript и jQuery(за фронт-енда). За изгледа за ползване HTML, CSS и Bootstrap. Основните функционалности се намират в сервизите. Там, по принцип може да се бъдат всички алгоритми които процесират данните. За интеракция с базата данна се ползва Spring Data и нейния интерфейс JpaRepository който ни позволява да пишем чист код който всъщност да представлява реална заявка и ни връщи комуникацията с определената таблица в базата от данни. Тези неща в нашия проект ги имплементираме в отделни интерфейси (репозиторита) в които наследяваме JpaRepository-то. На фиг 4.1 е приказан едно наше repository.

```
public interface JobsRepository extends JpaRepository<Job, Integer>
{
    Optional<Job> findById(int s);
    Optional<Job> findByPosition(String s);

    @Modifying
    @Query("update Job u set u.timesRated = :timesRated where u.id = :id")
    void updateRating(@Param(value = "id") int id, @Param(value = "timesRated") int
timesRated);
}
```

Фиг 4.1 – Repository относно jobs таблицата

От този пример се вижда хубаво че с много малко код може да направим реална заявка към MySQL-а. Тези методи са само дефинирани, докато тяхната имплементация ни е предоставена директно от JPA. Всички тези функционалности са използвани в нашите сервизи, където с помощта на @Autowired ние вмъкваме обект от repository-то. В следващото парче код е показано как е направен обект от JobsRepository с име jobsRepository с който натам ще ползваме функционалностите дефинирани в него.

```
public class JobServiceImpl implements JobService
{
    private final JobsRepository jobsRepository;
    private final ModelMapper modelMapper;
    private final CompanyRepository companyRepository;
```

```

@Autowired
public JobServiceImpl(JobsRepository jobsRepository, ModelMapper modelMapper,
CompanyRepository companyRepository)
{
    this.jobsRepository = jobsRepository;
    this.modelMapper = modelMapper;
    this.companyRepository = companyRepository;
}

```

Фиг 4.2 – Injection чрез @Autowired

```

@Override
public String updateAvgRating(int id, int rating)
{
    Job job = this.jobsRepository.findById(id).orElse(null);
    job.setTimesRated(job.getTimesRated()+1);
    job.setRatingSum(job.getRatingSum()+rating);
    this.jobsRepository.save(job);
    DecimalFormat df = new DecimalFormat("####0.00");

    double avgRating = (double)job.getRatingSum() / (double)job.getTimesRated();

    return df.format(avgRating);
}

```

Фиг 4.3 – Извикване на методите от репозиторито

На фигура 4.3 добре се вижда как се извиква тръсенето на работа чрез метода findById който проверява в базата от данни и връща обект от Job. Също така може да забележим метода save който е предназначен за сторване на информация в базата данни.

Контролерите в приложението са от типа на REST контролери. Вътре за момента се ползват GET и POST заявки които слушат на определени end-point-и. Един контролер е показан на фиг. 4.4.

```

@RestController
//@CrossOrigin("http://localhost:63343")
@RequestMapping(value = "/jobs", consumes = "application/json", produces =
"application/json")
public class JobsController
{
    private final JobService jobService;
    private final ModelMapper modelMapper;
    private final LogServiceImpl logService;

    @Autowired
    public JobsController(JobService jobService, ModelMapper modelMapper,
LogServiceImpl logService)
    {
        this.jobService = jobService;
    }
}

```

```

        this.modelMapper = modelMapper;
        this.logService = logService;
    }

    @PostMapping("/change-rating")
    public ResponseEntity partialUpdateSweatshirt(@RequestBody JobUpdateRating
jobUpdateRating) throws URISyntaxException
    {
        String avgRating = this.jobService.updateAvgRating(jobUpdateRating.id,
jobUpdateRating.rating);

        return ResponseEntity.created(new URI("/sweatshirts/change-
rating")).body(avgRating);
    }

    @PostMapping("/add")
    public ResponseEntity addSweatshirt(@RequestBody JobAdd jobAdd) throws
URISyntaxException
    {
        boolean result = this.jobService.addJob(jobAdd);

        return ResponseEntity.created(new URI("/sweatshirts/add")).body(result);
    }

    @GetMapping("/product")
    public Job getSingleSweatshirt(@RequestParam(name = "id") int id)
    {
        Job job = this.jobService.getSweatshirt(id);
        return job;
    }
}

```

Фиг 4.4 – Контролер за Jobs

От тук виждаме че един контролер е клас който е дефиниран с анотацията @Controller чийто default-ен първ end-point е /jobs който работи по json формат. Вътре има имплементирано различни методи където всеки отделен слуша на уникален end-point и приема входа на потребителя на този адрес, като след това го процесира чрез извикване на функционалности от сервизите и дава съответния резултат.

Докато същите тези енд-поинти в фронт-енда са описани по следния начин с помощ на JavaScript и jQuery.

```

app.router.on("#/logs/all", null, function () {

    $.ajax({
        type: 'GET',
        url: constants.serviceUrl + '/logs/all',
        headers: {
            'Authorization': app.authorizationService.getCredentials()
        }
    })
}

```

```
}).done((data) => {  
  app.templateLoader.loadTemplate('.app', 'logs-all', function () { ... }  
})
```

Потребителските роли са реализирани по йерархичен начин. Като всяка следваща роля наследява функционалностите на предишната и има допълнителни права. 5 главни роли в това приложение са:

- ROOT-ADMIN
- ADMIN
- MODERATOR
- ROLE-USER
- Guest

Приложението е отворено и достъпно за всеки един. За това по принцип, Guest не е роля, ами е посетител на сайта който не се е логнал. Основната роля при всеки един регистриран потребител е ROLE-USER. Това е обикновена роля която в случая има шанса да кандидатства директно за дадената работа, да си прави листа от любими обяви за работа, да request-ва обяви за работа. Следващата роля е модератора който е с идеята да поддържа сайта. В неговите права са посочени: да приема или отхвърля обяви за работа, да създава акаунти, да редактира акаунти, да гледа логове. Админа който е една степен по-висока роля има всичките права както обикновения потребител и модератора. Допълнително на това той притежава права за триене на потребители, триене на логове, частта за организацията където може да променя ролите на останалите потребители(без възможност да променя своята роля). И най-главния потребител е ROOT-ADMIN-а който всъщност е owner-а на приложението. Създаден е с идеята да няма админи които ще изтрият функцията на притежателя на приложението и откраднат приложението на много прост начин. Така, по принцип в този проект ролята ROOT-ADMIN се възлага на първия регистриран потребител който се очаква да сме самите ние(основателите).

5. По-детайлно описание (наблягане) на по-интересните и нестандартни функционалности на проекта.

Едни от по-интересните и нестандартни функционалности на проекта, според мен са:

5.1 Организация на потребителите

Show entries Search:

#	First name	Last name	Username	Role	Change Role
1	Tijana	Avramova	tijana	MODERATOR	<button>Promote</button> <button>Demote</button>
2	Petar	Najdov	peco	ROLE_USER	<button>Promote</button>
3	Dimitar	Avramov	dimitaravramov	ROOT-ADMIN	<button>Own Unchangeable</button>

First name Last name Username Role Change Role

Showing 1 to 3 of 3 entries Previous 1 Next

Refresh

Фиг 5.1 – Организация

От тук много лесно може да сменяме ролята на потребителите. Тази функционалност е достъпна само за потребителите с минимум роля на админ. Така, с един клик може да доделим роля на всеки потребител освен нашата и на главния админ. Всичко това е листнато в една таблица където с много прост изглед може да правим предложените функционалности. Същевременно таблицата ни предлага бързо фронт-енд сортиране на всеки един от редовете, както и тръсене по ключова дума итн.

Алгоритъма на тази функционалност е описан в UserServiceImpl.java класа.

```
@Override
public boolean promoteUser(String id) {
    User user = this.userRepository
        .findById(id)
        .orElse(null);

    if(user == null) return false;

    String userAuthority = this.getUserAuthority(user.getId());
    System.out.println("BEFORE PROMOTE USER AUTHORITY: " + userAuthority);
    Set<UserRole> authorities = new HashSet<>();

    switch (userAuthority) {
        case "ROLE_USER":
            authorities.add(this.userRoleService.findRole("MODERATOR"));
    }
```

```

        authorities.add(userRoleService.findRole("ROLE_USER"));
        user.setAuthorities(authorities);
//        user.setAuthorities(this.getAuthorities("MODERATOR"));
        break;
    case "MODERATOR":
        authorities.add(userRoleService.findRole("ADMIN"));
        authorities.add(userRoleService.findRole("MODERATOR"));
        authorities.add(userRoleService.findRole("ROLE_USER"));
        user.setAuthorities(authorities);
        break;
    default:
        throw new IllegalArgumentException("There is no role, higher than
ADMIN");
    }
    System.out.println("AFTER PROMOTE USER AUTHORITY: " +
this.getUserAuthority(user.getId()));
    this.userRepository.save(user);
    return true;
}

```

Фиг 5.2 – Промотване на потребител

Тук като параметър се подава id на даден потребител, като след това се тръси в базата данни. Ако не е намерен метода връща null и спира. Докато, ако е намерен потребителя, тогава се проверява коя роля притежава. След това добавя с една нагоре и запазва новата му роля която по йерархия е главната.

Също така постои и демотване на потребител т.е намаляване на неговата роля за едно. Това става по подобен принцип както в горния метод за promote.

5.2 История на логове

All Logs				
Show 10 entries		Search: <input type="text"/>		
Username	Operation	Modified table	Time	
Guest	Create	Users	13/01/2021 04:17:58	
dimitaravramov	Promote	Organization	13/01/2021 16:01:41	
Guest	Create	Users	13/01/2021 16:02:57	
Guest	Create	Users	13/01/2021 03:34:57	
Username	Operation	Modified table	Time	
Showing 1 to 4 of 4 entries				
		Previous	1	Next
Clear selected		Refresh		
		Clear All		

Фиг 5.3 – Логове

Тази функционалност според мен е една от най-важните в всички проекти. Тук се пази цялата история която се е случвала в живота на това приложение. Така, може да си правим разни статистики от типа кога приложението е най-активно, кой потребител е най-активен, кои части са най-популярни итн. Същевременно, може да се ползва за да намерим кой потребител направил нещо което е против правилата на сайта, кой модератор или админ направил грешка, кой модератор или админ си върши най-много работата итн. Имплементацията на това нещо се слуга в LogServiceImpl.java класа където разните методи реализират функционалностите който са достъпни в логовете.

```
@Override
public boolean insertLog(Log log)
{
    return this.logsRepository.save(log) != null;
}

@Override
public void removeLogsById(List<String> list)
{
    for(String uuid : list)
    {
        Log log = this.logsRepository.findLogById(uuid);
        this.logsRepository.delete(log);
    }
}

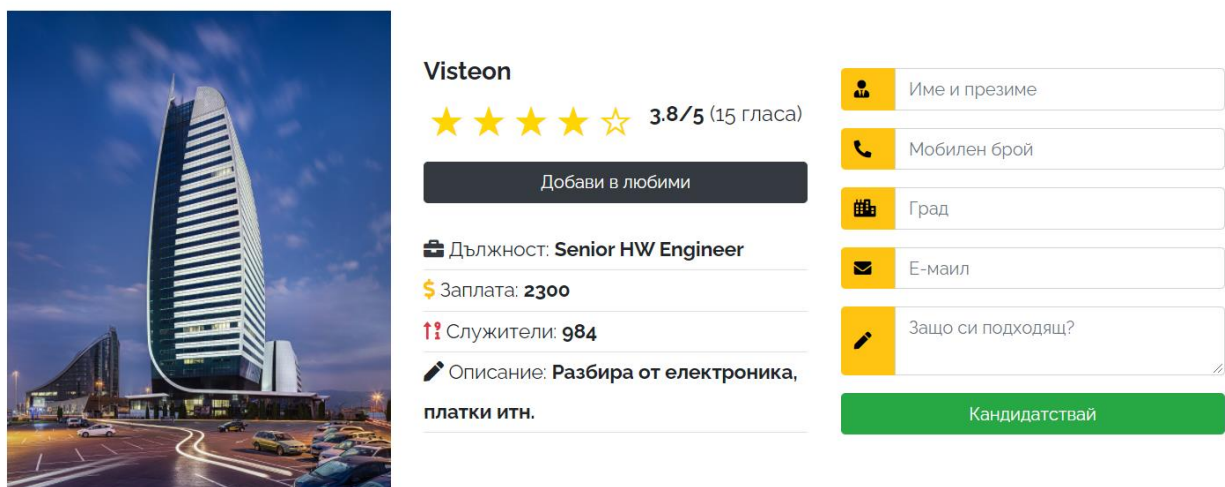
@Override
public List<Log> getAll()
{
    return
this.logsRepository.findAll().stream().collect(Collectors.toUnmodifiableList());
}

@Override
public void removeAllLogs()
{
    this.logsRepository.deleteAll();
}
```

Фиг 5.4 – Логове реализация

5.3 Рейтване на дадена обява

Senior HW Engineer



Visteon

★★★★☆ 3.8/5 (15 гласа)

Добави в любими

Длъжност: Senior HW Engineer

Заплата: 2300

Служители: 984

Описание: Разбира от електроника, платки итн.

Име и презиме

Мобилен брой

Град

Е-маил

Защо си подходящ?

Кандидатствай

Фиг 5.5 – Рейтване на обява

С функционалността за рейтване на дадена обява може да дадем нашата оценка базирана на наше изкуство, познаване и чуване на препоръки за дадената обява и фирма. Така, може да предоставим ясен приказ на останалите хора в сайта относно обявата. Те ще имат горе-долу познаване за това колко препоръчвана е тази обява от тяхните „виртуални колеги“. Функционалността на този фийчър е реализиран в JobServiceImpl.java класа.

```
@Override
public String updateAvgRating(int id, int rating)
{
    Job job = this.jobsRepository.findById(id).orElse(null);
    job.setTimesRated(job.getTimesRated()+1);
    job.setRatingSum(job.getRatingSum()+rating);
    this.jobsRepository.save(job);
    DecimalFormat df = new DecimalFormat("####0.00");

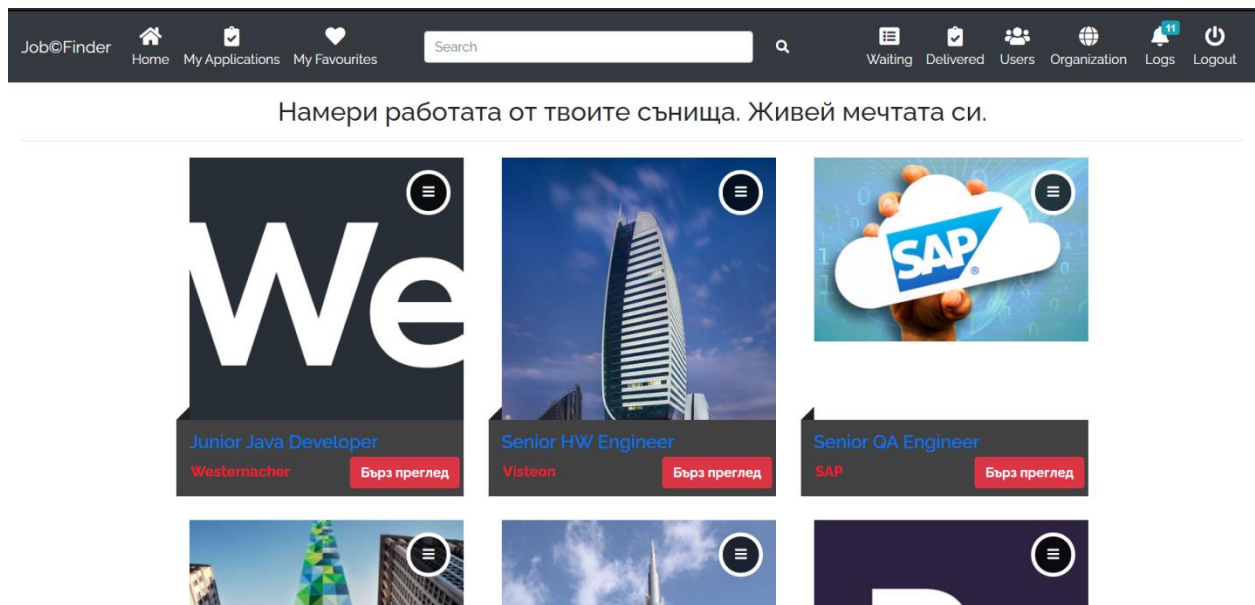
    double avgRating = (double)job.getRatingSum() / (double)job.getTimesRated();

    return df.format(avgRating);
}
```

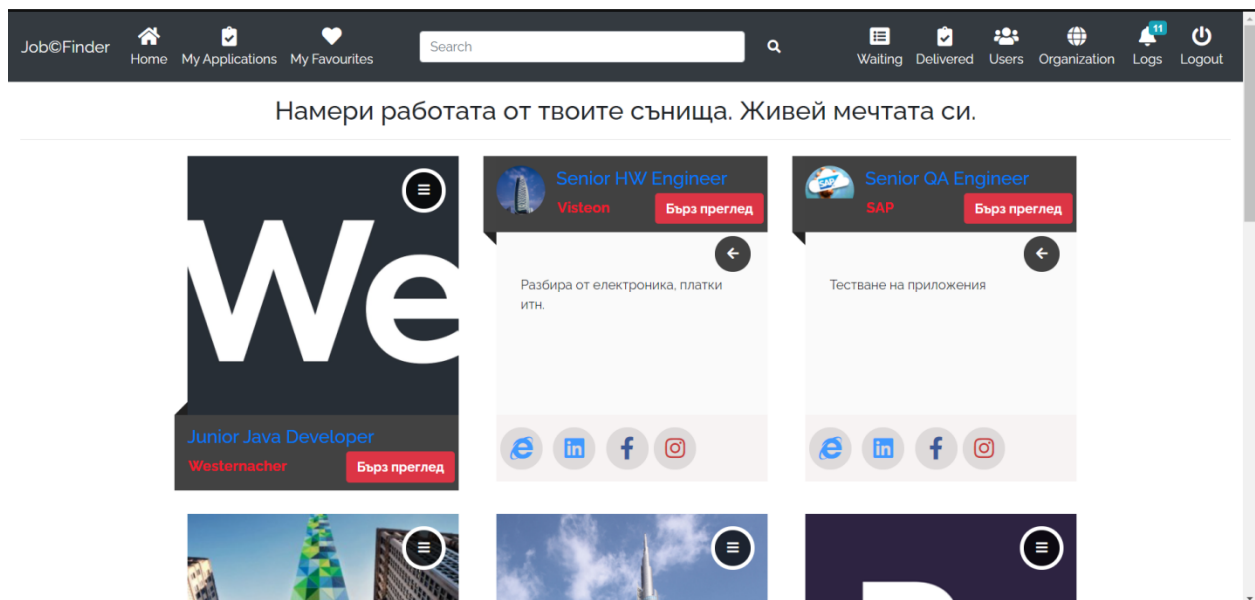
Фиг 5.6 – Реализация на рейтване

Тук отново се подава id-то на обявата, тръси се в базата данни и се връща цял обект, разбира се ако е достъпен. Увеличава се числото което означава колко пъти е рейтнато и се добавя новия инпут за рейтинг към текущата сума на рейтингзи. Така, на края се получава средния рейтинг и се връща, форматиран.


6. Screenshot на уебсайта, представящ неговия дизайн



Фиг 6.1 – Начална страница



Фиг 6.2 – Начална страница



UniCredit Bulbank

☆☆☆☆☆

Добави в любими

Длъжност: **Database SQL Engineer**

Заплата: **2400**

Служители: **312**

Описание: **Минимум 5 години опит с Oracle Database**

Име и презиме

Мобилен брой

Град

Е-маил

Защо си подходящ?


Кандидатствай

Фиг 6.3 – Бръз преглед на обява

Job@Finder Home My Applications My Favourites Search

Waiting Delivered Users Organization Logs Logout

Junior Java Developer



☆☆☆☆☆

Добави в любими

Длъжност: **Junior Java Developer**

Заплата: **2800**

Служители: **50**

Описание: **Основни познания по MVC, Spring Boot, HTML5, CSS3, Database, Rich faces. Flat структурата е ежедневното отличие на нашата компания**

Име и презиме

Телефонен номер

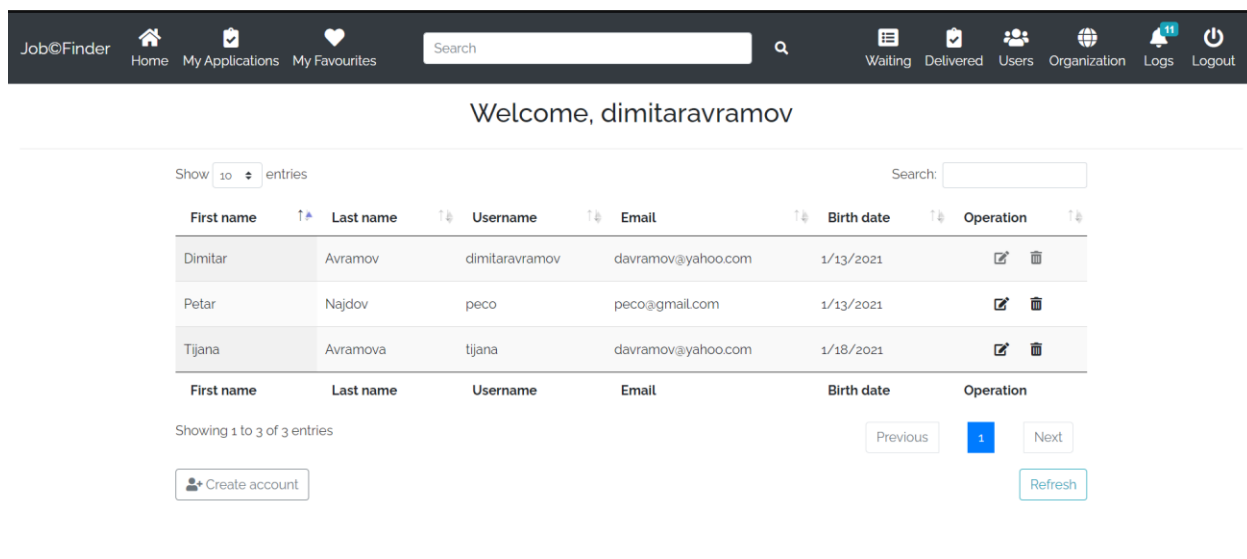
Град

Е-маил

Защо си подходящ?

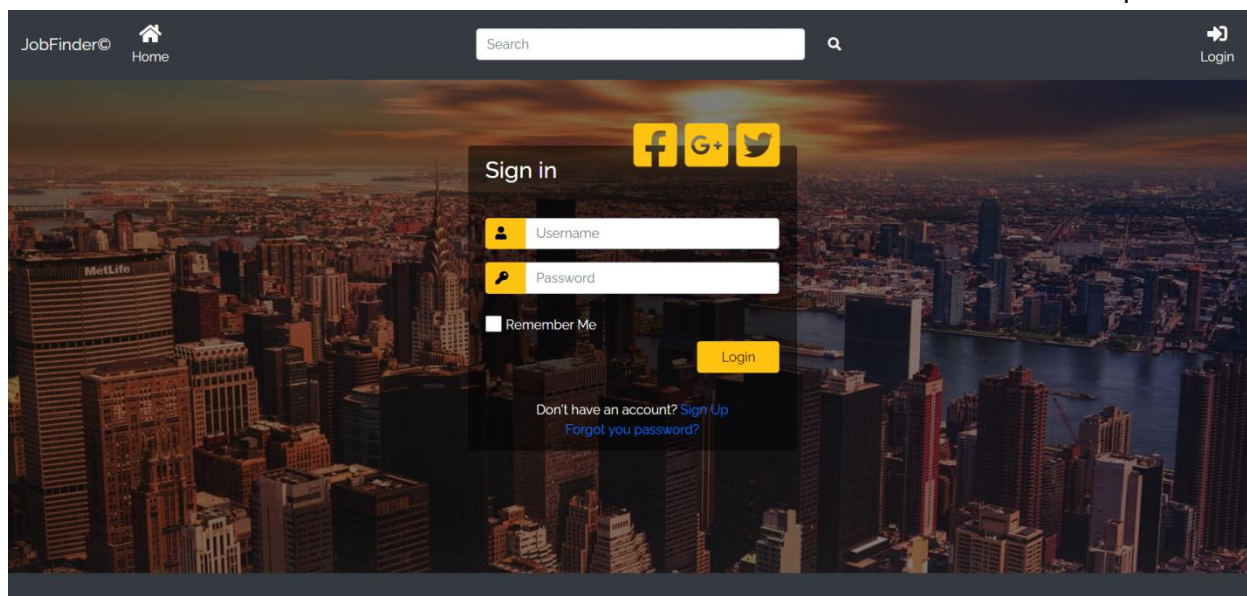
Кандидатствай

Фиг 6.4 – Отделна страница на обява



Фиг 6.5 – Листа на всички потребители

Фиг 6.5 – Листа на всички потребители



Фиг 6.6 – Логиране на потребител

The image shows a web browser window with the JobFinder website. The background is a high-angle photograph of a city skyline at dusk. A dark, semi-transparent overlay contains a 'Sign Up' form. The form has a title 'Sign Up' at the top. Below it are seven input fields, each with a yellow icon on the left: a person icon for 'First name', a person icon for 'Last name', a person icon for 'Username', an envelope icon for 'Mail', a calendar icon for 'Date of birth', a key icon for 'Password', and a key icon for 'Confirm password'. At the bottom of the form are two yellow buttons: 'Login' and 'Register'. The top of the browser window shows the 'JobFinder' logo, a 'Home' link with a house icon, a search bar with the text 'Search', and a 'Login' link with a person icon.

JobFinder® Home Search Login

Sign Up

First name

Last name

Username

Mail

Date of birth

Password

Confirm password

Login Register

Фиг 6.7 – Регистриране на потребител

7. Линк към GitHub

<https://github.com/avramov97/find-job-pi>