Universidade de Évora
Departamento de Informática
Sistemas Operativos II
Ano letivo 2019 - 2020

Ocupação de Espaços Comerciais (Aplicação Web)

Alunos:
Luís Ressonha - 35003
Rúben Teimas - 39868

Docente:
José Saias

7 de Julho de 2020

Indice

1 Introdução **1**

2 Desenvolvimento da aplicação **2**
2.1 Back-End 2
2.1.1 *Spring MVC* 2
2.1.2 Persistência de dados 3
2.1.3 Acesso e manipulação de dados 4
2.1.4 Receção e resposta a pedidos 4
2.2 Front-End 5
2.2.1 Renderização das páginas 5
2.2.2 Apresentação e interatividade 6

3 Execução do trabalho **7**

4 Conclusão **8**

5 Referências **9**

1 Introdução

Neste trabalho, da *UC* de *SOII*, pretende-se que seja desevolvida uma *aplicação web* que seja útil à comunidade permitindo aos utilizadores consultar e registar os niveis de ocupação numa superficie comercial.

A *aplicação web* deve ser desenvolvida usando, maioritariamente, as tecnologias abordadas na *UC*.
É também pretendido que a aplicação faço uso de replicação, o que permite suportar falhas de comunicação ou na obtenção de dados.

Para a implementação da aplicação utilizámos a *framework Spring Boot*, sendo bastante matura e utilizada a nivel empresarial.

Para a persitência dos dados a *Spring* oferece a funcionalidade *Spring Data JPA*, que permite ter uma camada de abstração em cima de *JPA*.

Outra funcionalidade, disponibilizada pela *framework*, que optámos por utilizar foi a *Spring Security*. Esta funcionalidade permite-nos tratar de aspetos de, como o nome sugere, segurança. Foi com esta funcionalidade que implementámos a autorização e autênticação da nossa aplicação.
Ainda no que toca a segurança, a nossa aplicação comunica com o *browser* utilizando o protocolo *HTTPS* o que permite comunicar numa ligação encriptada, contrariamente a *HTTP*.

Usando a *framework Spring* a nossa aplicação está alojada num servidor *Tomcat* embutido nesta.

No *front-end* da aplicação utilizámos a *template engine Thymeleaf* bem como a *framework Bootstrap* (para a apresentação).

Utilizámos também algum *Javascript* juntamente com *JQuery* e *AJAX* para adicionar alguma interatividade à aplicação bem como establecer algumas comunicações entre o cliente e o servidor.

Para a compilação da aplicação é utilizado *Gradle*.

2 Desenvolvimento da aplicação

Dada a extensão da *Aplicação Web* é necessário dividi-la em pequenas porções.

A principal divisão a ser feita é a distinção entre as operações executadas no servidor(*back-end*) e as operações executadas no cliente/*browser*(*front-end*).

2.1 Back-End

O *back-end* é responsável por processos como acesso e manipulação dos dados, reencaminhar pedidos para o *endpoint* correto bem como responder a esses mesmos pedidos e renderização de algumas *views*, sendo que o *Thymeleaf* faz uso de server-side rendering.

É também responsável pela segurança da aplicação e pela lógica do negócio da mesma.

2.1.1 *Spring MVC*

A nossa *aplicação web* é uma aplicação monolitica, contrariamente a micro-serviços, e optámos por utilizar o *Design Pattern MVC*.

Este divide-se em 3 partes:

- ***Model:*** estão incluindos os *DAO*'s e os *POJO*(*Plain Old Java Object*) que guardam os dados vindos da base de dados.
- ***View:*** as views são o conteúdo com o qual o utilizador interage. Estas são renderizadas no servidor.
- ***Controller:*** responsáveis pela comunicação entre a *views* e *models*, recebendo os pedidos e reencaminhando-os para o *endpoint* correto bem como respondendo aos mesmo.

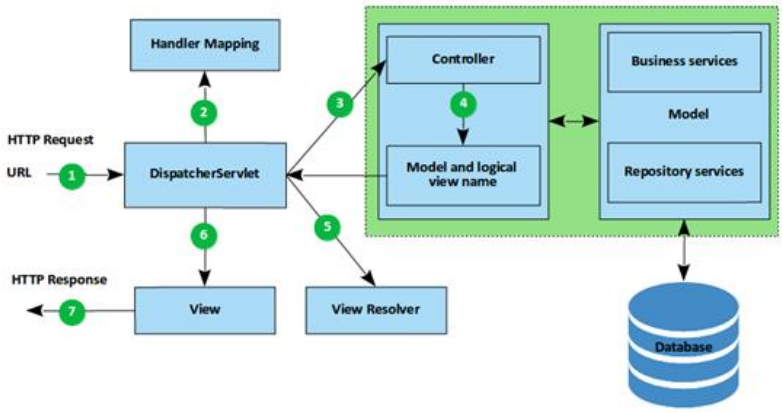

Figura 1: Arquitetura *Spring MVC*

2.1.2 Persistência de dados

A persistência dos dados é alcançada utilizando a *Spring Data JPA*, o que facilitou bastante o nosso trabalho, dispensando a escrita de comandos *SQL* (tanto instruções *DDL* como a criação de *querys*).

A utilização de JPA permite-nos criar as tabelas e relações utilizando *Java* com algumas anotações. Este tipo de objectos é designado por *DAO*(*Data access object*).

Para além das vantagens acima mencionadas, a *JPA* permite, através da sua camada de abstração, uma migração simplificada de um sistema de base de dados, para outro.

Neste caso utilizámos uma base de dados *Postgres*, contudo, se em algum momento decidissemos migrar para uma base de dados *Oracle* poderiamos fazê-lo sem ter de re-escrever *querys* e comandos de criação de tabelas.

A nossa *BD* é constituída por 5 tabelas, sendo que 2 delas dizem respeito à autenticação e as restantes aos dados da aplicacação.

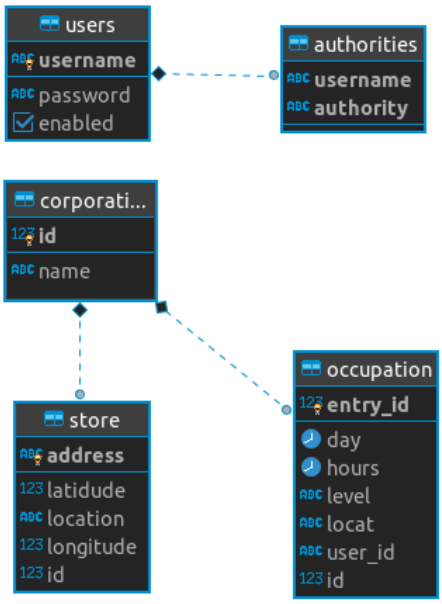

Figura 2: Modelo E-R

As 3 tabelas de dados assentam na ideia de que uma loja(*store*) pertece a uma companhia (*corporation*) e uma companhia pode ter várias lojas espalhadas. Os niveis de

ocupação(*occupation*) são registados numa loja que pertence a uma companhia.

Para as tabelas de autorização utilizámos *JDBC* em vez de *JPA*, simplesmente porque a *Spring Security* já tem uma pré-configuração de autenticação utilizando *JDBC*.

Nessa pré-configuração existe uma tabela (authorities) à qual não demos uso, embora inicialmente tivessemos pensado dar. A nossa ideia seria que os utilizadores tivessem apenas autorização às operações pedidas, para além disso iriamos criar uma autoridade *admin* que teria autorização também à operação de adicionar companhias e lojas. Ainda que seja relativamente simples, não foi implementado.

As passwords são guardadas na *BD* de forma encriptada pois é mais seguro do que utilizando *plain-text*.

2.1.3 Acesso e manipulação de dados

As classes dedicadas ao acesso e manipulação de dados (chamadas à base de dados) bem como à lógica das operações encontram-se na diretória "services".

Essas classes são iniciadas com a anotação *@Service* e são posteriormente chamadas nos *Controllers* através de injeção de dependências (utilizando a anotação *@Autowired*).

2.1.4 Receção e resposta a pedidos

É na diretória "controllers"que se encontram as classes responsáveis por receber e responder a pedidos. Estas classes são iniciadas com a anotação *@Controller* ou *@Rest-Controller*, sendo a última um tipo especifico de controlador.

A nossa aplicação faz uso de 3 *controllers*:

- ***ApiController:*** responsável por redirecionar os pedidos para os *endpoints* com prefixo */api*. Envia a resposta em formato *JSON*, um formato adequado para uma arquitetura *REST*.
- ***HomeController:*** responsável por todos os pedidos feitos a *endpoints* que sejam públicos (não requerem autenticação e autorização). As respostas enviadas são *views* processadas no servidor.
- ***UserController:*** resposável por todos os pedidos feitos para os *endpoints* com prefixo */user*. As respostas enviadas são, à semlhança do controlador anterior, *views* processadas no servidor.

Como é percetivel pela descrição dos controladores, a nossa aplicação disponibiliza uma *API* para algumas operações.

Endpoint	***HTTP Verb***	**Descrição**
/api/localizacao/{id}	*GET*	Devolve localizações das lojas de uma empresa.
/api/procurar-ocupacao	*GET*	Devolve nivel de ocupação de uma loja.
/api/coordenadas	*GET*	Devolve as coordenadas de uma loja.
/api/registar-ocupacao	*POST*	Cria um novo registo de ocupação numa loja.
/api/remover-registo/{id}	*DELETE*	Remove o registo de ocupação de uma loja.

Tabela 1: API da aplicação

Os pedidos da *API* são executados a partir do cliente mediante a ocorrência de enventos, como por exemplo submeter um formulário.

2.2 Front-End

O *front-end* da nossa aplicação é aquilo que o utilizador consegue ver que, ainda que seja bonito, não é estonteante!

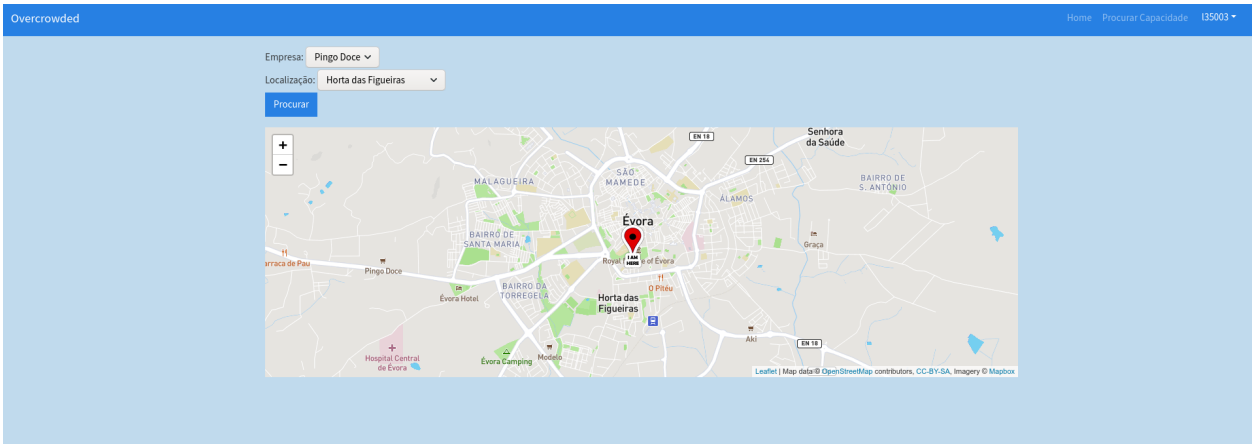

Figura 3: Página de procura de niveis de ocupação

2.2.1 Renderização das páginas

Para a renderização das nossas *views* utilizámos a *template engine Thymeleaf* devido à sua simplicidade e boa documentação.

Esta ferramente permitiu-nos, através de pequenas alterações em código *html*(*tags* próprias do *Thymeleaf*), obter páginas *web* dinâmicas.

Para cada uma das *views* criámos uma página *html*. Em todas as páginas são reutilizados 2 elementos: a *head* do documento *html* e a barra de navegação que podemos

observar em todas as páginas.

Estes pedaços de código reutilizáveis são designados de *fragments*.

2.2.2 Apresentação e interatividade

Para que a aplicação se tornasse, visualmente, mais apelativa usámos a *framework Bootstrap*.

Através da mesma conseguimos dar um melhor aspeto à nossa aplicação produzindo muito pouco código *css*.

Para além de *Bootstrap* utilizámos *JQuery* e *AJAX* para adicionar alguma interatividade e dinamismo à nossa página.

É utilizando *AJAX* que obtemos os dados da *API* que protamente têm efeito na *view* sem que seja preciso recarregar à página. Achamos que este tipo de dinamismo é interessante para operações simples e que o utilizador pode querer fazer várias vezes de seguida, como por exemplo, remover um dos seus registos.

3 Execução do trabalho

Para a execução do trabalho é necessário ter uma instância de uma base de dados *Postgres*.

Por simplicidade nós optámos por fazer um *dump* da nossa base de dados sendo apenas colocar os dados do ficheiro *bd.sql* na instância da base de dados, ficando assim com as tabelas criadas e preenchidas com dados.

As configurações da base de dados, como o nome da mesma e utilizador, podem ser encontradas no ficheiro *application.properties* que se encontra da diretória *src/main/resources/*.

Após a criação da base de dados a aplicação pode ser compilada executando o comando, na pasta base do projecto, *gradle build*.

O *deployment* é conseguido executado o comando *gradle bootRun*.

Para **instruções mais detalhadas** acerca de como criar a Base de dados ou executar a aplicação pode seguir os passos do ficheiro **readme.md** na pasta anterior à pasta base do projeto.

4 Conclusão

Este trabalho foi realmente bastante importante na medida que nos permitiu trabalhar com várias tecnologias *web* que, para além de bastante interessantes, são recentes e utilizadas no mundo empresarial (algo que nem sempre aconteceu ao longo da licenciatura).

Para além das tecnologias *web* tivemos também a oportunidade de programar usando anotações, algo que também era completamente novo para nós.

Ainda assim, este trabalho foi bastante desafiante na medida em que existia muita liberdade e várias tecnologias novas.

Uma das maiores dificuldades inicais foi compreender a hierarquia da *framework Spring*. Felizmente, esta possui uma boa documentação, o que se tornou bastante útil.

Tivemos também algumas duvidas em como mostrar o resultado das nossas operações ao utilizador (*front-end*).
Inevitavelmente acabamos por pesquisar acerca de conceitos como *Server-side rendering VS Client-side rendering*, SPA's entre outros.

A lógica das operações pedidas era bastante simples sendo todo o ecossistema do trabalho a parte desafiante e interessante.

A componente base do trabalho foi totalmente implementada, a componente adicional 2 foi parcialmente implementada (ficando a faltar a 5ª operação) e a componente adicional 1 (replicação) não foi implementada.

Tivemos problemas a obter as coordenadas do utilizador utilizando o *browser Firefox*, mas o mesmo não sucedeu utilizando o *Brave*.

Apesar dos problemas acima descritos o trabalho foi, parcialmente, concluído com sucesso!

5 Referências

- *Spring JPA docs;*
- *Spring Security Default User Scheme;*
- *HTML Geolocation API;*
- *StackOverflow: distance between 2 coordinates;*