Segundo Trabalho Prático de SO2

Ocupação de espaços comerciais

Universidade de Évora
22 de Junho de 2020

Miguel Azevedo nº 36975 e Vasco Crespo nº 37913

Introdução

Para a implementação desta aplicação web recorremos à utilização da ferramenta Maven para a gestão do nosso projeto e à framwork Spring. Para manter os dados dos utilizadores e da aplicação utilizamos uma base de dados postgresql e para a manipulação dessa base de dados no nosso projeto utilizamos a API JPA que facilita bastante todo o processo. Por fim utilizamos o template thymeleaf para fazermos a interface para o utilizador a partir de ficheiros html.

Base de dados

A nossa base de dados divide-se em 3 tabelas diferentes:

Tabela utilizador:
A nossa tabela tem cinco colunas, sendo elas uma String "utilizador"contendo o nome do utilizador que faz o login, uma String "password"que é a password desse utilizador e um Long "id"que será gerado automaticamente para identificar o utilizador. Cada utilizador também tem uma Role para ter acesso apenas a determinadas operações. Nesta tabela o id será a chave primária.

Tabela store:
Esta tabela é utilizada para manter as lojas presentes no nosso programa. Cada elemento desta tabela tem uma String "store_name"que guarda o nome da loja e um double "latitude"e double "longitude"que terão os valores da latitude e longitude dessa loja. Também tem um long "identificador"que tal como na tabela utilizador será a chave primária e única que representa cada loja.

Tabela ocupation:
Esta tabela comtém a informação acerca dos registos de ocupações feitas tendo um Long "store_id"(que é a chave primária da tabela store sendo chave estrangeira nesta tabela), que é o id da loja da qual queremos indicar a ocupação. Tem também um Long "utilizador_id", novamente chave primária mas desta vez da tabela utilizador, que guarda o valor do id único do utilizador que está a registar a ocupação e um int "lotacao"que indica qual a ocupação da loja em questão. Terá também um valor automaticamente posto que é um Long "time_stamp"que é um valor em milisegundos que determina o tempo que já passou desde o dia 1 de Janeiro de 1970. E por fim terá também um long "id"que é a chave primária desta tabela e sendo também um valor único e gerado automaticamente nesta tabela.

Autenticação

Para a realização do login do utilizador utilizamos a framework Spring Security que simplifica o processo todo de autenticação tendo até uma página web já pré implementada. A autenticação será feita através do acesso à nossa tabela utilizador que verificará se o nome e a respetiva password já existem e caso exista o utilizador fica logged in até fazer logout. Para o registo criámos manualmente um form html que recebe o utilizador e a password e coloca-os na base de dados(tendo em conta se o seu username já existe ou não).

Desenvolvimento

Para a utilização da nossa aplicação não é obrigatório existir um Login, contudo o utilizador apenas terá acesso a alguns serviços, como por exemplo consultar o grau de ocupação de um espaço comercial.

Também poderá registar-se ou então se já o tiver feito, pode fazer o login. Ao efetuar o login, o utilizador poderá ser um utilizador normal ou então um admin, sendo as diferenças nas operações à qual tem acesso. O utilizador regular apenas tem acesso às operações correspondentes ao registo de um grau de ocupação, à visualização dos seus registos, à visualização de um grau de ocupação de uma loja e à remoção de um grau de ocupação registado por esse utilizador. Enquanto que o utilizador admin tem acesso a todas estas operações mas também à criação e remoção de lojas e de utilizadores. Podendo assim consultar o grau de ocupação de um espaço comercial, registar o seu grau de ocupação, consultar os seus registos, remover um registo e por fim verificar a loja mais proxima de si.

Para os diferentes serviços de sistema foram criados vários Controllers(que definem o que acontece em cada path recebido) tais como:

- LoginController

  Este Controller tem como objetivo controlar o que acontece quando são recebidos:

  - o default mapping, ou seja *"/"*, onde apenas vai mostrar a nossa página inicial contendo as opções de fazer login, sign up, verificar a ocupação de um espaço ou ir para o menu(apesar de apenas poder aceder ao nosso menu principal após fazer o login)
  - o mapping *"/remUser"* que permite remover um utilizador regular do sistema a partir do seu username
  - o mapping *"/register"* que permite registar um utilizador novo através de um username e uma password desde que o username não exista já na nossa table utilizador da base de dados
  - o mapping *"/menu"* que apresenta o menu principal com as operações à qual aquele utilizador já logged in tem acesso, dependendo se é um utilizador regular ou se é um admin
  - o mapping *"/menuAdmin"* que apresenta o menu do admin, ou seja um menu extra que contém as operações que são de acesso restrito aos admins

- StoreController

  - *"/createStore"* que tem como objetivo criar uma loja. Para isso executa o metodo *findStore* criado nos serviços, que irá verificar se a loja já existe no sistema ou não, e caso não exista, é criada.
  - *"/remStore"* que tem como objetivo remover uma loja do sistema. Para isso executa o metodo *findStore* criado nos serviços, que irá verificar se a loja já existe no sistema ou não, e caso exista, é removida.

- OcupationController

  Este controlador tem como objetivo realizar qualquer tipo de *mapping* relacionado com serviços *ocupation*, tais como:

  - */setOcupation* que tem como objetivo o registo de ocupação de um espaço. Para isso é executado o metodo *createRegister* que irá verficiar se a loja que o utilizador introduziu existe no sistema, criando assim um registo na tabela ocupation caso a loja exista. Caso a loja não exista, o utilizador irá ser direcionado para uma nova página HTML com a informação que a loja não foi encontrada.

  - */getOcupation* que tem como objetivo saber o grau de ocupação de um espaço. Para isso é executado o método *getGrauOcupacao* devolver uma *ArrayList* de inteiros, com os registos efetuados sobre a ocupação daquele espaço na última hora. Caso a loja introduzida pelo utilizador não exista, esta lista terá apenas -1 direcionando o utilizador para uma página HTML que indica que a loja não existe. Caso apenas ainda não existam registos sobre aquela loja, é adicionado 0 a lista, direcionando novamente para uma nova página HTML.

  - */getRegistos* que tem como objetivo mostrar todos os registos efetuados por aquele utilizador. Para isso é executado o método *getRegistos* devolver uma *ArrayList* de objetos, em que cada um irá conter a loja e a lotação do espaço. Para isso é enviado para o frontEnd um objeto DTO que contem apenas a informação necessária.

– */remRegisto* que tem como objetivo remover um registo efetuado por aquele utilizador. Para isso é executado o método *removeRegisto* que irá remover da tabela ocupação o registo com o nome do utilizador, e com o id do registo que o utilizador deseja remover. Caso o id inserido não exista ou exista algum problema na remoção, o utilizador será redirecionado para as respectivas páginas HTML.

– */nearStore* que tem como objetivo devolver a loja mais próxima de cada utilizador. Para isso o utilizador introduz a sua latitude e longitude atual, e através desses dados vamos comparar com todas as lojas existentes no sistema, devolvendo apenas a que se encontra mais próxima.

Bibliografia

- https://spring.io/
- https://spring.io/projects/spring-security
- https://leafletjs.com/