JavaFX

A aplicação JavaFX funciona com base em três componentes principais, sendo eles Controllers, Models e Views.

No nosso caso, definimos as seguintes views:

- Login – Mostra uma interface simples de login ao utilizador.
- Main – Componente comum a todas as views. Permite visualizar o utilizador atualmente autenticado.
- ChooseDelegate – Permite escolher um delegado e um tema ao qual o associar.
- CreateProject – Permite criar projetos de lei.
- ProjectList – Permite visualizar uma lista com todos os projetos de lei ativos.
- ProjectDetail – Permite visualizar os detalhes de um projeto de lei selecionado.
- VoteList – Permite visualizar uma lista com todas as votações ativas.
- VoteDetail – Permite visualizar os detalhes de uma votação.
- UseCases – Permite selecionar o caso de uso a executar.

Estes componentes permitem a interação do utilizador com o sistema, através do uso de Controllers. Cada view tem um controller associado que permite efetuar operações com base no clique de botões, alterar o conteudo de elementos da view e registar dados correntes utilizados por outras views.

Para partilhar informação entre views recorremos a um objeto DataModel. Este objeto foi criado inicialmente na classe principal da aplicação, sendo depois passado a cada um dos controllers aquando da sua inicialização. O objeto DataModel permite a partilha de informação entre os vários controllers, sendo usado no nosso caso específico para armazenar qual o utilizador autenticado, qual o voto ou projeto de lei selecionado pelo utilizador e também obter listas “mock” de delegados e temas de projeto de lei. Estes dados são armazenados no DataModel através de propriedades de forma a permitir o uso de observers sobre os mesmos. Os objetos propriedade nada mais são do que wrappers sobre os modelos definidos por nós. Estes modelos são usados para efetuar representações mais simples dos objetos do modelo de domínio da aplicação. No nosso caso decidimos criar os seguintes modelos:

- Person – Representa utilizadores, contendo informações sobre se o utilizador é ou não um delegado.
- Project – Representa um projeto de lei.
- Theme – Reprsenta um tema.
- Vote – Representa uma votação.

Para obter as informações utilizadas nestes modelos recorremos a classes Service. Cada modelo (com a excessão de Theme) tem uma classe Service associada, sendo esta responsável por efetuar pedidos http à api rest disponibilizada pelo servidor. Os pedidos feitos à api retornam strings JSON que necessitam de ser convertidas em objetos Java.

Para tal, recorremos à bilbioteca Gson. De forma a simplificar a conversão, utilizamos ainda objetos DTO intermédios. O resultado final consiste em obter a string JSON do pedido, utilizar o Gson para a converter no respetivo DTO e por fim utilizar essa informação para criar os modelos. De notar que a única forma que conseguimos que o Gson funcionasse corretamente foi através do uso de atributos públicos nos objetos DTO.

Em suma, aplicação JavaFX utiliza várias vistas para mostar informação ao utilizador, cujas partes dinâmicas são controladas por classes Controller, que partilham informação através de uma classe DataModel. A informação do DataModel é armazenada em objetos modelo, criados com base na informação obtida do servidor através de classes Service.

API REST

A API REST utilizada pela aplicação JavaFX está desenvolvida com 3 componentes: RestControllers, Services e DTOs.

Os RestControllers representam a própria API REST em si, definindo os diferentes endpoints HTTP. Cada um dos Controllers define os respetivos endpoints necessários para cumprir os casos de uso definidos no enunciado. Dependendo do que é pedido, estes endpoints vão ser pedidos GET ou POST e devolvem um DTO ou um ResponseEntity. Podem também incluir dados no endereço do endpoint ou como corpo no pedido. Neste projeto, foram definidos 3 Controllers diferentes:

1) RestCidadaoController, com o endpoint seguinte:

    a) **GET "/obterCidadao/{numCC}":** Dado um número de Cartão de Cidadão, se não houver um cidadão com esse CC devolve uma resposta de código 404 e se houver devolve o respetivo CidadaoDTO.

2) RestProjetoLeiController, com os endpoints seguintes:

    a) **GET "/projetosLei":** Lista os projetos de lei abertos.
    b) **GET "/projetosLei/{id}":** Dado um id, se não existir um projeto de lei com esse id devolve uma resposta de código 404 e se existir devolve o respetivo ProjetoLeiDTO.
    c) **POST "/projetosLei/{projId}/apoiar/{numCC}"**: Dado um id de projeto de lei e um número de Cartão de Cidadão, se houver um projeto de lei com esse id e um cidadão com esse CC, adiciona o apoio desse cidadão. Se não houver algum dos elementos, devolve um código 404.

3) RestVotacaoController, com os endpoints seguintes:

    a) **GET "/votações"**: Lista as votações em curso.
    b) **GET "/votacao/{votId}/defaultVote/{numCC}"**: Dado um id de votação e um número de Cartão de Cidadão, se houver uma votação com esse id e um cidadão com esse CC, e o cidadão tiver um voto automático, devolve um int que representa o voto (1: SIM, 0: NÃO). Se não houver algum dos elementos, ou o cidadão não tiver delegado associado ao tema da votação, devolve -1.
    c) **POST "/votacao/{votId}/vote/{numCC}"**: Dado um id de votação, um número de Cartão de Cidadão e um VotoDTO no corpo do pedido, se houver uma votação com esse id e um cidadão com esse CC, se o VotoDTO for válido e se o cidadão ainda não tiver votado nessa votação, atribui o voto nesse DTO (1: SIM, 0: NÃO). Se não houver algum desses elementos, se o VotoDTO for inválido ou se o cidadão já tiver votado, devolve um código 409.

Para poder dividir responsabilidades e para que a camada da API em si seja o mais fina possível, os Controllers simplesmente recebem o pedido e devolvem uma resposta adequada, mas a lógica por detrás do pedido é atribuída aos Services. Os Services ficam responsáveis de duas funções: chamar as funções necessárias dos Handlers, para que a camada de negócio da aplicação faça com que os pedidos tenham efeito, e transformar os objetos da lógica de negócio em DTOs. Neste projeto foram implementados os seguintes Services: CidadaoService, DelegadoService, FilePDFService, ProjetoLeiService, TemaService, VotacaoService, sendo o FilePDFService o único que apenas serve para transformar os objetos PDF em DTO.

Para poder transferir a informação transmitida através dos pedidos HTTP realizados à API, recorremos ao padrão Data Transfer Object para aquelas entidades necessárias. O seu uso também é importante para enviar apenas a informação necessária, e não enviar informação a mais que esteja no objeto original da camada de negócio. Neste projeto são usados os seguintes DTOs: CidadaoDTO, DelegadoDTO, FilePDFDTO, ProjetoLeiDTO, TemaDTO, VotacaoDTO e VotoDTO.

Aplicação WEB

A aplicação WEB funciona com base em 4 componentes principais, sendo eles WebControllers, DTO's, Services e Views.

Alguns dos componentes, nomeadamente os Services e os DTO's são partilhados com a API REST, logo, não serão explicados aqui em detalhe.

De modo a desenvolver os casos de uso propostos, foram desenvolvidos os seguintes endpoints HTTP, divididos pelos respetivos Controllers:

1) WebCidadaoController, com os seguintes endpoints:

   a) **GET "/":** Devolve a página inicial (neste caso o login);
   b) **POST "/login":** Redireciona o utilizador para a dashboard cujo identificador é número de cartão de cidadão inserido na página anterior;
   c) **GET "/{cc_number}/dashboard":** Devolve a página da Dashboard cujo identificador é o número de cartão de cidadão do utilizador. Nesta página é possível ver as votações em curso, ver os projetos de Lei em curso, apresentar um projeto de Lei e também escolher um delegado para um determinado tema;
   d) **GET "/{cc_number}/escolherDel":** Devolve a página onde se pode escolher um delegado para um determinado tema também à escolha;
   e) **POST "/{cc_number}/escDel":** Dados o número do cartão de cidadão (explícito no endpoint) e escolhidos um delegado e um tema, associa o delegado dado ao tema dado para o cidadão corrente (definido pelo número cc);

2) WebFilePDFController, com os seguintes endpoints:

   a) **GET "/consultarProjsLei/{id}/pdf/{filename}":** Devolve o PDF do projeto de Lei com o id dado.

3) WebProjetoLeiController, com os seguintes endpoints:

   a) **GET "/{cc_number}/apresentarProjLei":** Devolve a página onde o delegado com o número de cartão de cidadão **cc_number** pode apresentar um projeto de Lei. Somente delegados podem aceder a esta página a partir da dashboard.
   b) **POST "/{cc_number}/apresProjLei":** Dados o título, a descrição, um ficheiro PDF, um tema e uma data de validade referentes a um projeto de Lei, cria o respetivo projeto de Lei e associa-o ao delegado corrente.

c) **GET "/{cc_number}/consultarProjsLei":** Devolve a página onde se podem consultar os projetos de Lei que ainda estão em curso. Nesta página é possível aceder a um projeto em específico para ver os seus detalhes.
d) **GET "/{cc_number}/consultarProjsLei/{id}":** Devolve a página onde se podem ver os detalhes de um projeto especificado pelo id **id**.
e) **POST "/{cc_number}/consultarProjsLei/{id}":** Caso o utilizador decida apoiar o projeto, é verificado se já o fez anteriormente. Caso o tenha feito, é redirecionado para uma página de erro criada especificamente com uma mensagem referente ao erro. Caso contrário, o utilizador é redirecionado para a sua dashboard.

4) WebVotacaoController, com os seguintes endpoints:

a) **GET "/{cc_number}/listarVot":** Devolve a página onde se podem ver as votações em curso. Nesta página o utilizador consegue aceder a uma votação em específico para ver os seus detalhes.
b) **GET "/{cc_number}/listarVot/{id}":** Devolve a página onde se podem ver os detalhes de uma votação em específico, nomeadamente, o tema da votação e o voto do delegado do cidadão corrente, caso este tenha escolhido um delegado, e caso o respetivo delegado já tenha votado nesta votação. O utilizador pode também submeter o seu próprio voto, caso não concorde com o voto do seu delegado.
c) **POST "/{cc_number}/votarProposta/{id}":** Recebe o voto do cidadão com número cc dado, registando-o na votação com o dado id. Caso o cidadão já tenha votado anteriormente nesta votação, é redirecionado para uma página de erro onde é informado que já votou. Caso contrário, o utilizador é redirecionado para a sua dashboard.
d) **GET "/{cc_number}/votarPropostaError":** Devolve a página de erro referente ao caso em que o cidadão já tenha votado anteriormente numa determinada votação.

Para cada um dos pedidos GET apresentados foi criada uma view (página HTML) que difere na funcionalidade dependendo de se o utilizador tem privilégios de cidadão ou de delegado (somente os delegados podem apresentar projetos de Lei nesta parte desenvolvida).

Cada um dos Controllers depende dos Services explicados na secção da API REST para poder aceder à lógica de negócio e converter os objetos do domínio em DTO's.

Em suma, a aplicação Web utiliza várias vistas para mostrar informação ao utilizador, cujas partes dinâmicas são controladas por classes WebController, que acedem a classes Service para fornecer e obter informação da camada de negócio. Estas vistas consistem em páginas html estáticas, com partes dinâmicas feitas recorrendo ao Thymeleaf.