Relatório Projeto - Fase 2

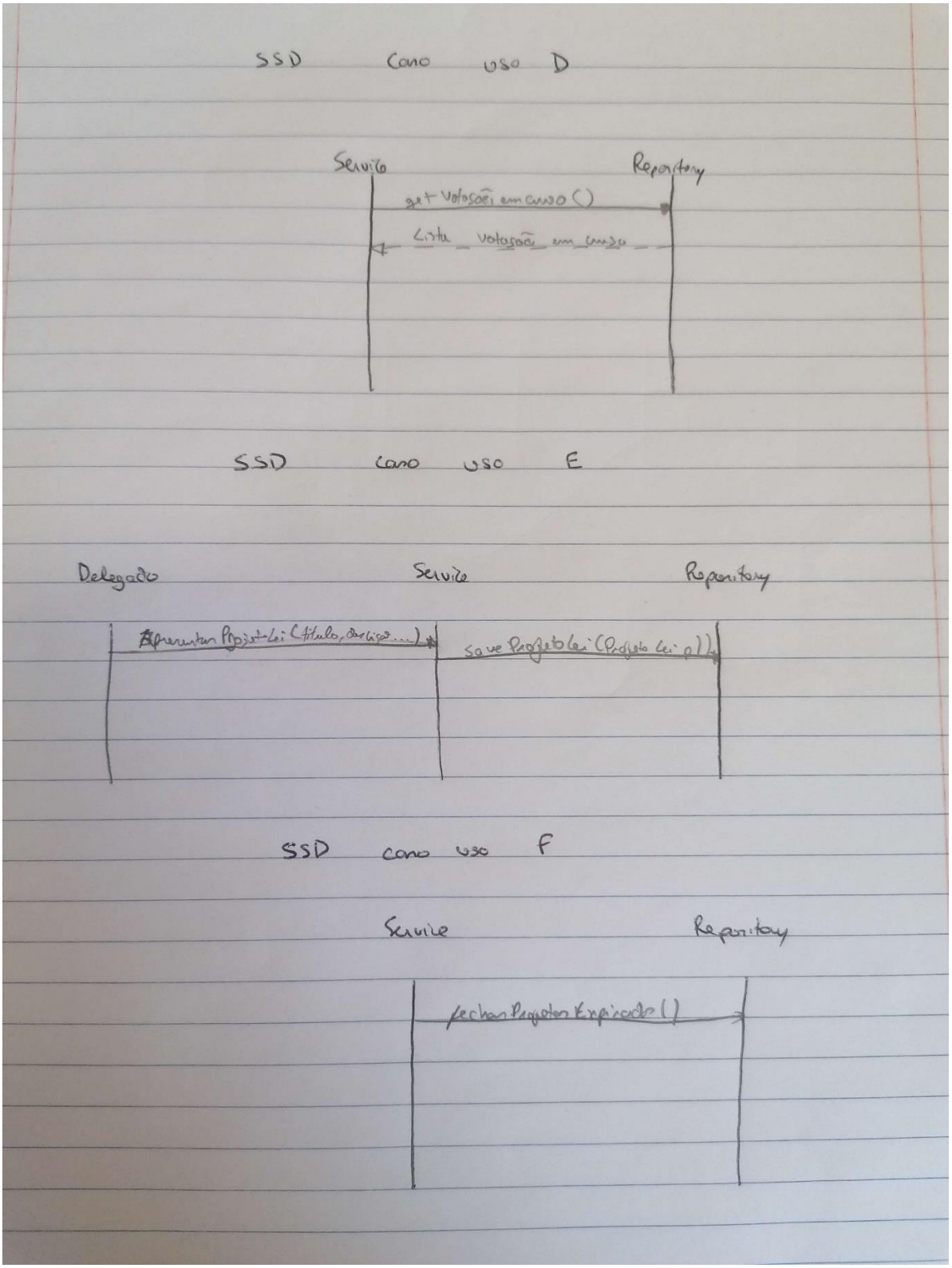
Trabalho Realizado por:

FC55172 Tiago Pinto

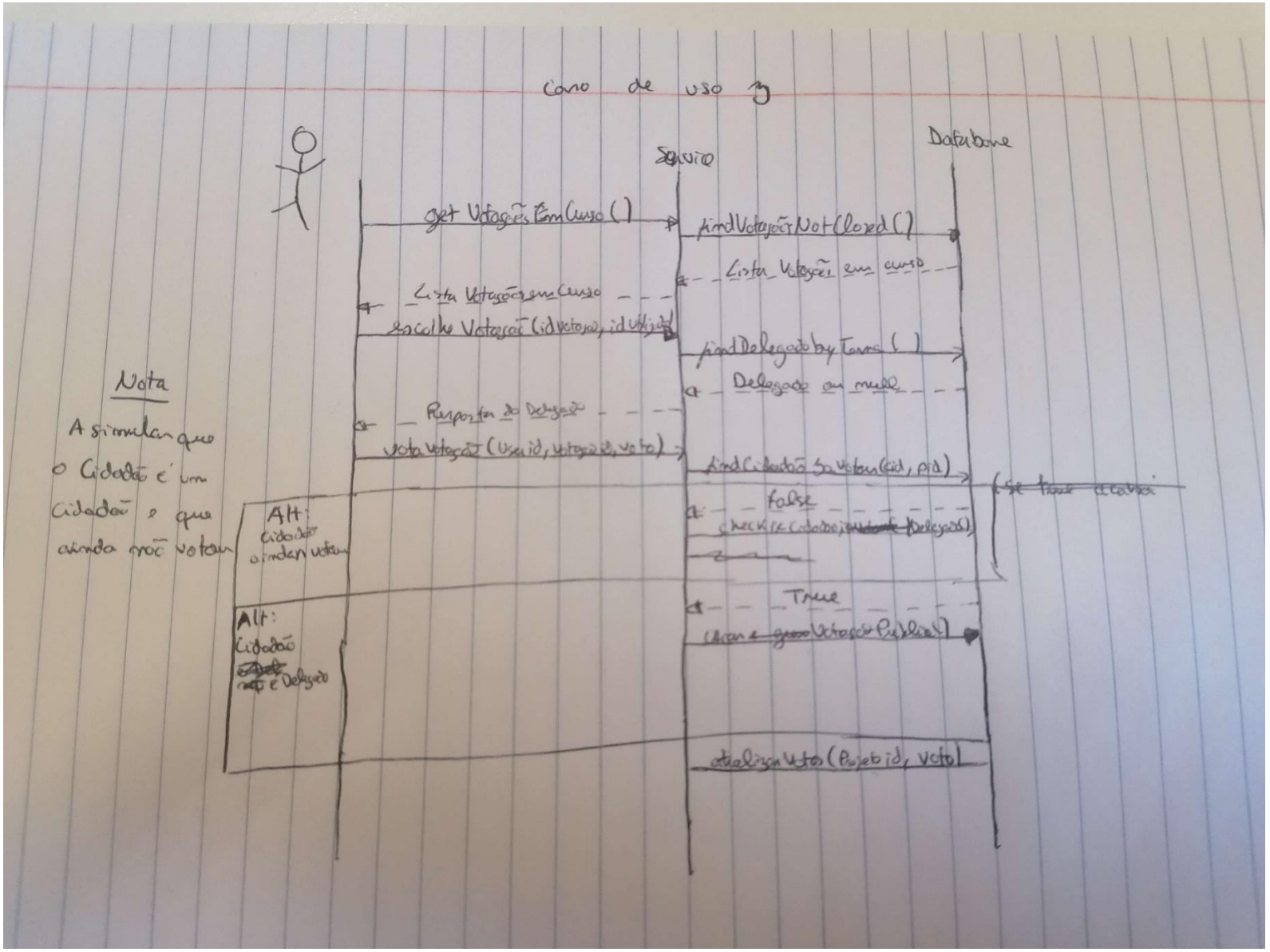
FC54446 Ricardo Soares

FC54409 Miguel Reis

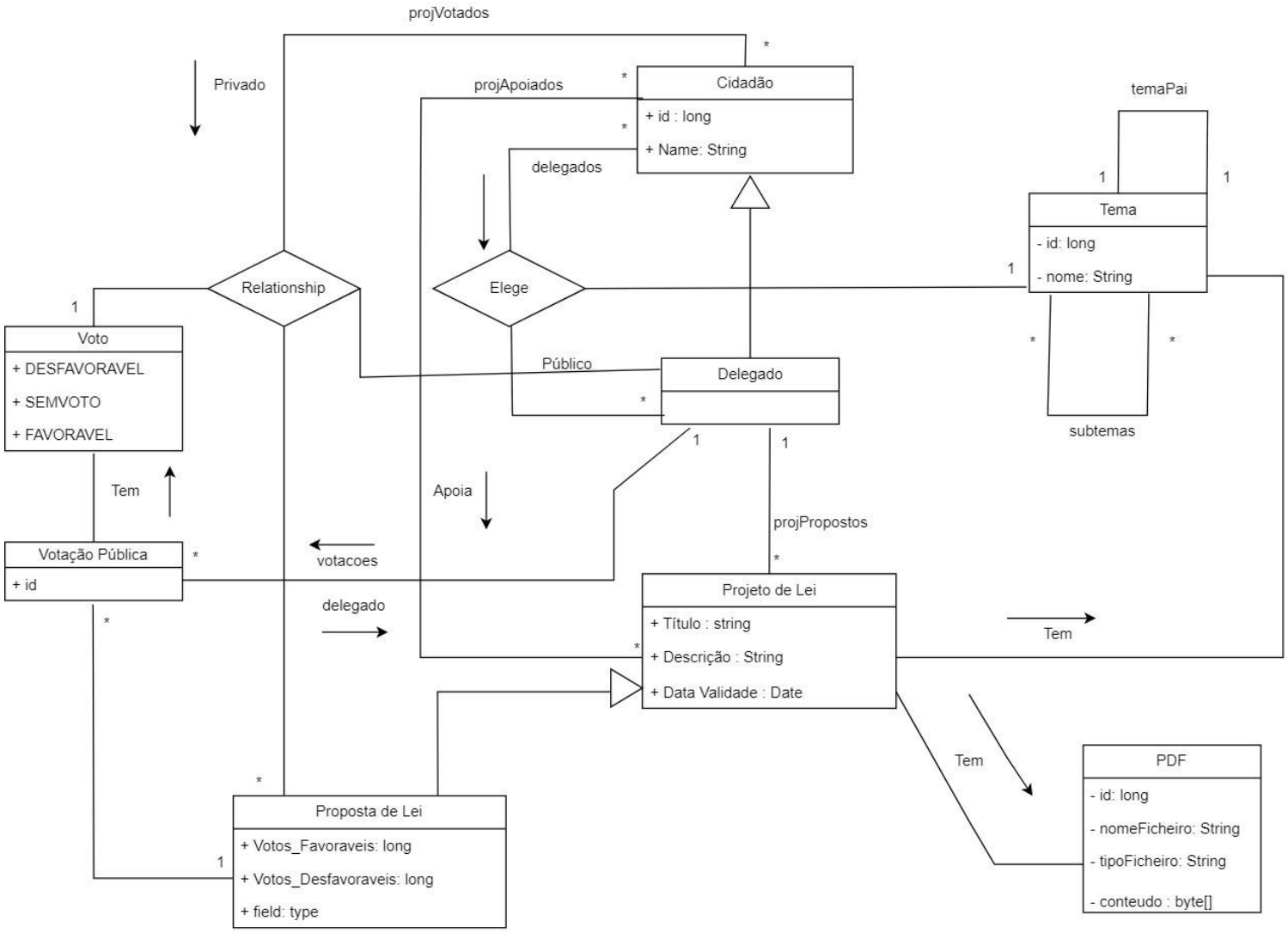
SSD's caso de uso D, E e F



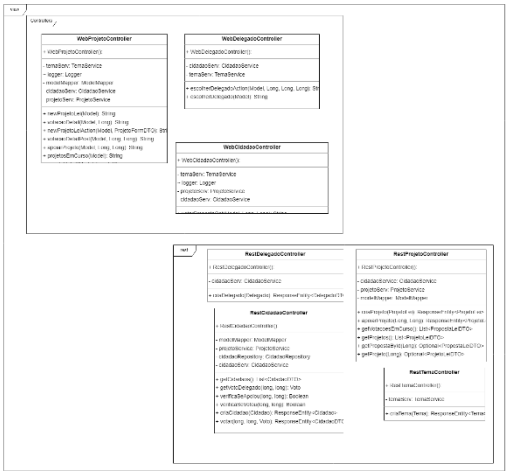
SSD caso de uso J

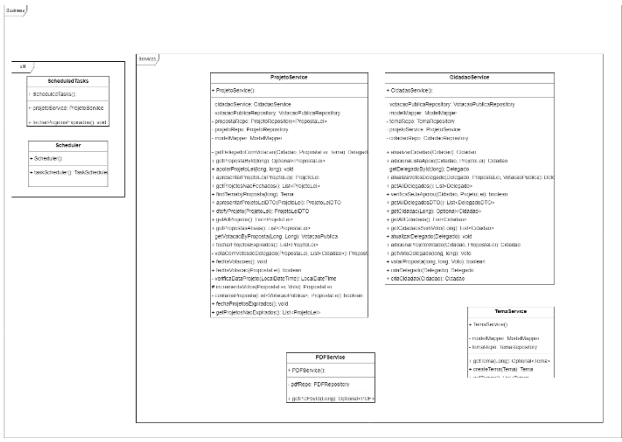


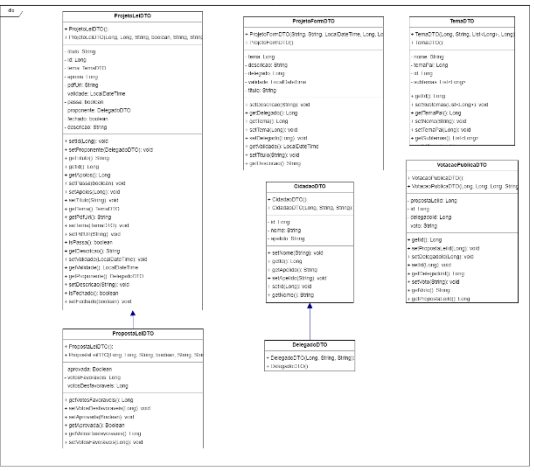
Modelo de Domínio

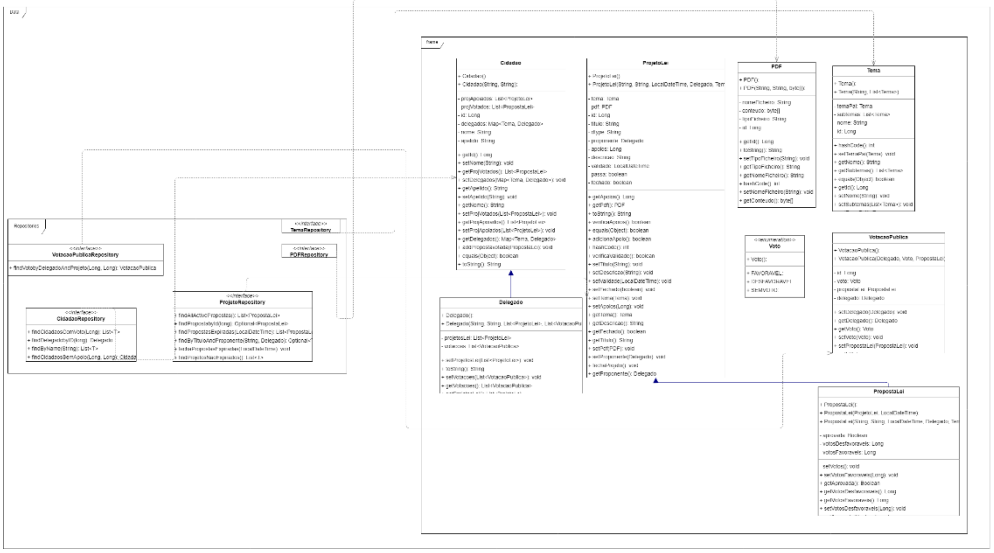


Esboço do Diagrama de Classes









Mapeamento

Os objetos que queriamos persistir foram anotadas com @Entity.

Na classe Cidadao utilizámos @Inheritance com a strategy SINGLE_TABLE para persistirmos o Cidadaoe a sua classe filha, Delegado, na mesma tabela. Escolhemos fazer assim pois o Delegado também atuacomo cidadao, sendo a informação toda guardada na mesma tabela para um acesso mais fácil.

Todas as entidades têm o atributo id anotado com @Id e @GeneratedValue(strategy = GenerationType.SEQUENCE), para distinguir as entidades pela chave primária e para o id ser autogerado pela Base de dados.

Atribuimos @NonNull a atributos que considerámos necessários para a identificação das entidades.

Foi utilizado o *FetchType.EAGER* em grande parte dos atributos mapeados com @ManyToMany ou com @OneToMany para evitar o lançamento de exceções relacionados com *LazyInitializationException*.

Foi utilizado o CascadeType para propagar as operações realizadas nas entidades.

Ligação criada entre o Delegado e as Votações Públicas correspondentes através do @OneToMany e @ManyToOne, sendo os atributos do Delegado correspondentes mapeados pela classe Dona.

Utilização do @Enumerated na classe VotacaoPublica para persistir o tipo de voto como um enumerado, tendo sido escolhido o tipo String para uma maior facilidade de interpretação da informação.

Utilização do @Lob para guardar o conteúdo do PDF do ProjetoLei na Base de Dados.

É utilizada a anotação @Transaction para realizar uma transação.

Estrutura Interna - Componentes

- Web Server-Side

Server-side web: trata, basicamente, da camada de aprensetação (view) dos dados, sendo por isso responsável pela interface gráfica que é enviada ao cliente, aquando de um pedido. Trata da interação do utilizador com a aplicação, tratando também de exibir os dados.

Usamos Controllers para fornecer aos clientes a possibilidade de conseguirem fazer várias funcionalidades na página web, como por exemplo, adicionar um projeto de lei, votar numa proposta, etc.

Estas operações são feitas usando os Services criados na 1º fase, que permite aos clientes realizarem as várias funcionalidades possíveis.

Usamos ficheiros html para que, quando um cliente faça um pedido (por exemplo, para ver a lista de projetos ou apoiar um projeto), seja disponibilizada a informação e os dados de uma certa maneira especifica, que seja intuitiva e agradável ao cliente.

Nos controladores web, usamos essencialmente 4 anotações: @Controller, que define a classe como sendo um Controller, @Autowired para injetar as dependências, e @GetMapping e @PostMapping para podermos fazer os pedidos, sendo que esses pedidos são utilizados para popular o modelo e mudar para a vista (MVC), na página web.

- REST API

São utilizados Controllers para disponibilizar aos Clientes endpoints onde podem comunicar com o Servidor para fazer diversos pedidos. Estes Controladores chamam por si os *Services* criados na primeira fase, para realizar operações sobre os objetos criados.

As classes responsáveis pela REST API utilizam a anotação @RestController() e @RequestMapping("api"), sendo os pedidos realizados para http://localhost:8080/api. Os pedidos são identificados por @GetMapping e por @PostMapping, tendo cada um um endereço específico. Foi usada a classe ModelMapper para mapear os objetos para os DTO respetivos. A interface REST é usada pela aplicação desktop para interagir com o servidor.

- JavaFX

Para a aplicação desktop, utilizamos o javaFX para realizar a aplicação. Utilizámos como base o projeto fornecido pelos docentes. Criámos uma interface comum (JavaFXController) para implementar os controladores das diversas cenas da nossa interface. A anotação @FXML é usada. A informação é conseguida a partir dos pedidos REST implementados. Criamos modelos dos objetos (correspondem a DTOs) para representar a informação recebida. Os ficheiros .fxml foram criados utilizando o SceneBuilder.

Algumas notas

- A nossa intenção era colocar a aplicação Spring numa pasta e o JavaFx noutra, para separar os módulos, mas tivemos alguns problemas com o Docker e por isso pusemos o diretório do cliente-desktop (aplicação JavaFx) dentro do projeto Spring.
- É necessária ter o maven instalado para correr o client.sh.
- Para lidarmos com uma espécie de simulação de login, visto não ser pedido no trabalho, o que fizemos foi, cada vez que for para votar/apoiar, seleciona-se o cidadão que o vai fazer.
- Em ubuntu, é preciso dar permissões aos scripts através do comando chmod +x.
- Para correr o cliente e ser possível realizar pedidos REST, é preciso primeiro correr o Servidor (run.sh).
- Para correr o cliente.sh, o JAVA_HOME precisa de ser pelo menos o Java 17 (assumimos que a aplicação vai ser corrida nesse ambiente de execução).