demo

facade

facade.controllers

:WebProjetoDeLeiController

+ listarProjetosDeLei(model) : String
+ apoiaProjetoDeLeiView(model) : String
+ apoiaProjetoDeLei(model, eapDTO) : String
+ projetoDeLeiDetails(model, projetoDeLeiId) : String
+ apresentaProjetoDeLeiView(model) : String
+ apresentaProjetoDeLei(model, proj, pdfFile) : String

:RestProjetoDeLeiController

+ getProjetosDeLeiNaoExpirados() : List<ProjetoDeLeiDTO>
+ getProjetosDeLeiDTO(projetoId) : ResponseEntity<?>
+ apoiaProjetoDeLei(eleitorCC, projetoDeLeiId) : ResponseEntity<?>
+ downloadPDF(id) : ResponseEntity<?>

:WebEleitorController

+ escolheDelegado(model) : String
+ escolheDelegadoAction(model, eadDTO) : String

:WebVotacaoController

+ getVotacoesEmCurso(model) : String
+ getFormularioVotoOmissao(model) : String
+ getVotoPorOmissa(model, voto) : String
+ getFormularioVoto(model) : String
+ createVoto(model, voto) : String

:RestVotacaoController

+ getVotacoesEmCurso() : List<VotacaoDTO>
+ votoPorOmissao(eleitorCC, votacaoId) : ResponseEntity<?>
+ createVoto(votoDto) : ResponseEntity<?>

:WebController

+ getIndex(model) : String

facade.services

:EleitorService

+ escolherDelegado(temaNome, eleitorCC, delegadoCC) : void

:ProjetoDeLeiService

+ getProjetosDeLeiNaoExpirados() : List<ProjetoDeLeiDTO>
+ apoiaProjetoDeLei(eleitorCC, projetoDeLeiId) : void
+ apresentaProjetoDeLei(titulo, temaNome, delegadoId, dataValidadeString, descricao, anexoPdf) : ProjetoDeLeiDTO
+ getProjetoDeLeiDTOById(id) : ProjetoDeLeiDTO

:VotacaoService

+ getVotacoesEmCurso() : List<VotacaoDTO>
+ getVotoPorOmissao(eleitorCC, votacaoId) : VotoDTO
+ createVoto(eleitorId, votacaoId, valorVoto) : VotoDTO
+ createVoto(eleitorCC, votacaoId, valorVoto) : VotoDTO

facade.handlers

:ApoiaProjetoDeLeiHandler

+ apoiaProjetoDeLei(projetoDeLeiId,eleitor) : void
- abrirVotacao(projetoDeLei) : Votacao
- votoDelegadoProponente(delegadoProponente, votacao) : void

:VotarNumaPropostaHandler

+ indicaVotacao(votacao, eleitor) : Voto
+ indicaVoto(voto) : void
- verificaVoto(voto) : boolean

:ApresentarProjetoDeLeiHandler

+ apresentarProjetoLei(projetoDeLei) : void

:FecharProjetosDeLeiHandler

+ fecharProjetosDeLei() : void

:ListarVotacoesEmCursoHandler

+ listarVotacoesEmCurso() : List<VotacaoDTO>

:EscolherDelegadoHandler

+ eleitorEscolheDelegado(eleitor, delegado, tema) : void

:ConsultarProjetosDeLeiHandler

+ listarProjetosDeLei() : List<ProjetoDeLeiDTO>
+ getProjetoDeLei(projetoDeLeiId) : ProjetoDeLeiDTO

:FecharVotacaoHandler

+ fecharVotacao(votacao) : void

repositories

:ProjetoDeLeiRepository <<interface>>

+ findByEstado(estado) : Optional<ProjetoDeLei>
+ fecharExpirados(curr_date, est) : void

:EleitorRepository <<interface>>

+ findDelegadoById(id) : Optional<Delegado>
+ findByCc(cc) : Optional<Eleitor>
+ findDelegadoByCC(cc) : Optional<Delegado>

:TemaRepository <<interface>>

+ findByNome(nome) : Optional<Tema>

:VotoRepository <<interface>>

+ findByEleitorVotacao(eid, vid) : Optional<Voto>
+ findDelegadosQueVotaram(vid) : List<Delegado>

:VotacaoRepository <<interface>>

+ findByEstado(estado) : List<Votacao>
+ updateVotosPositivos(vid) : void
+ updateVotosNegativos(vid) : void
+ updateVotacaoComoTerminada(vid, res, estado) : void

:EleitorDelegadoAssociacaoRepository <<interface>>

+ findByEleitorTema(eid, tid) : Optional<EleitorDelegadoAssociacao>
+ findEleitoresDelegadoParaTema(did, tid) : List<Eleitor>
+ updateDelegado(eid, d, tid) : void

associations

:VotoId

- votacaoId : long
- eleitorId : long

+ getters & setters

id

:Voto

- valorVoto: boolean

+ getters & setters

{Set} * * {Set}

EleitorDelegadoAssociacaoId

- eleitorId : long
- temaId : long

+ getters e setters

id

:EleitorDelegadoAssociacao

+ getters & setters

{Set} * * {Set}

entities

:Eleitor

- id : long
- nome : String
- cc : int
- token : String

+ getters & setters

1 1

:ProjetoDeLei

- id : long
- titulo : String
- descricao : String
- anexoPdf : byte[]
- dataValidade : LocalDateTime
- numeroApoios : int

+ getters & setters

{Set} projetosDeLeiApoiados * * apoiantes

delegadoProponente * 1

{Set} projetosDeLeiPropostos

:Delegado

+ getters & setters

1 * 1 1

1 1 projetoDeLei

de

:Votacao

- id : long
- estado : EstadoVotacao
- resultado : ResultadoVotacao
- votosPositivos : int
- dataValidade: LocalDateTime
- votosNegativos : int

+ getters & setters

:Tema

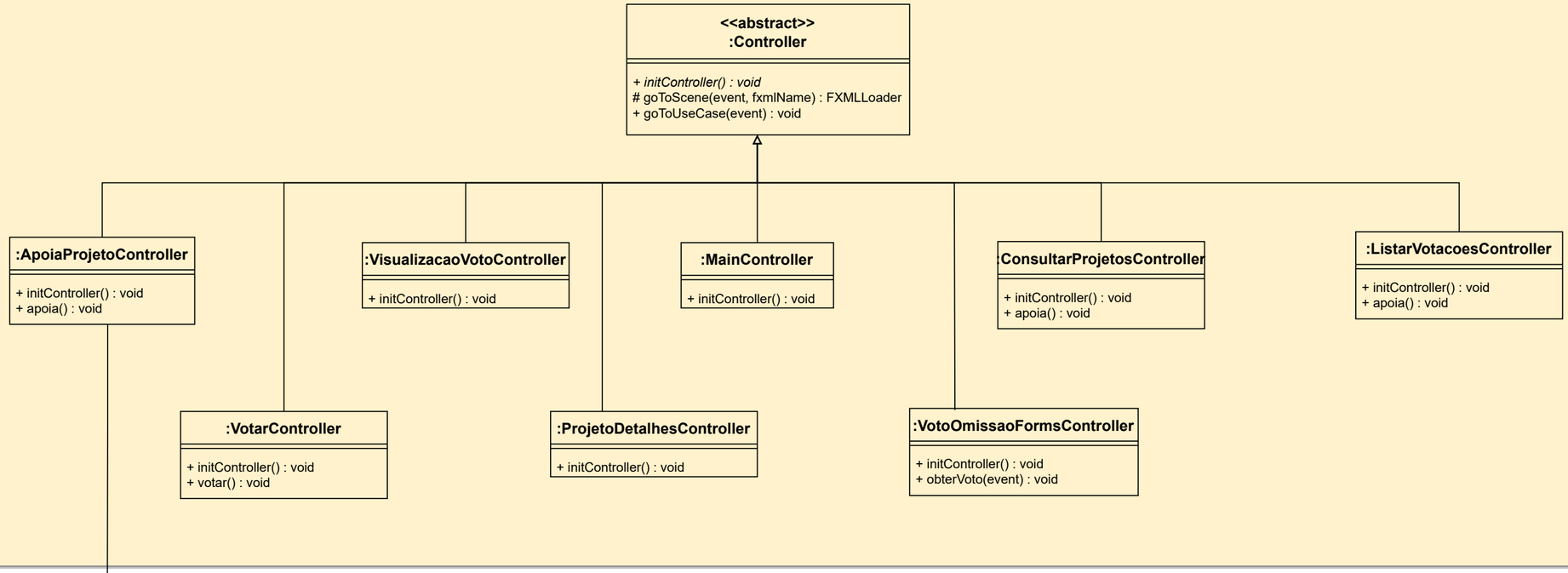
- id : long
- nome : String

+ getters & setters

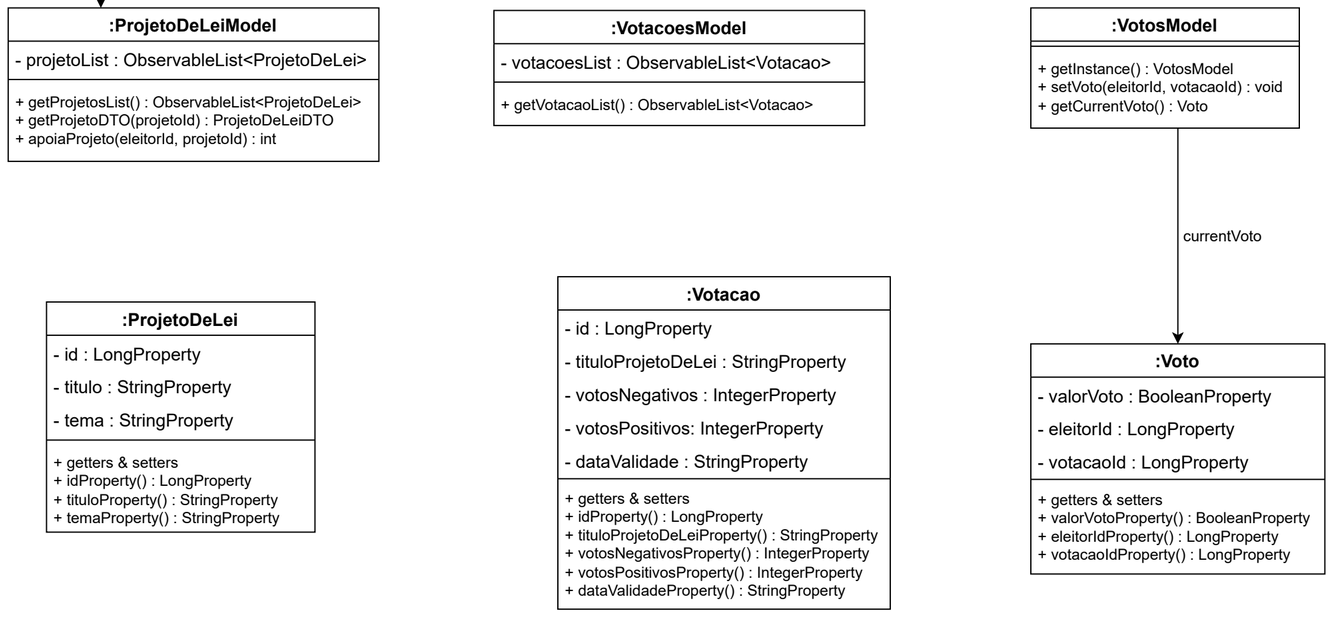
1 * temaPai

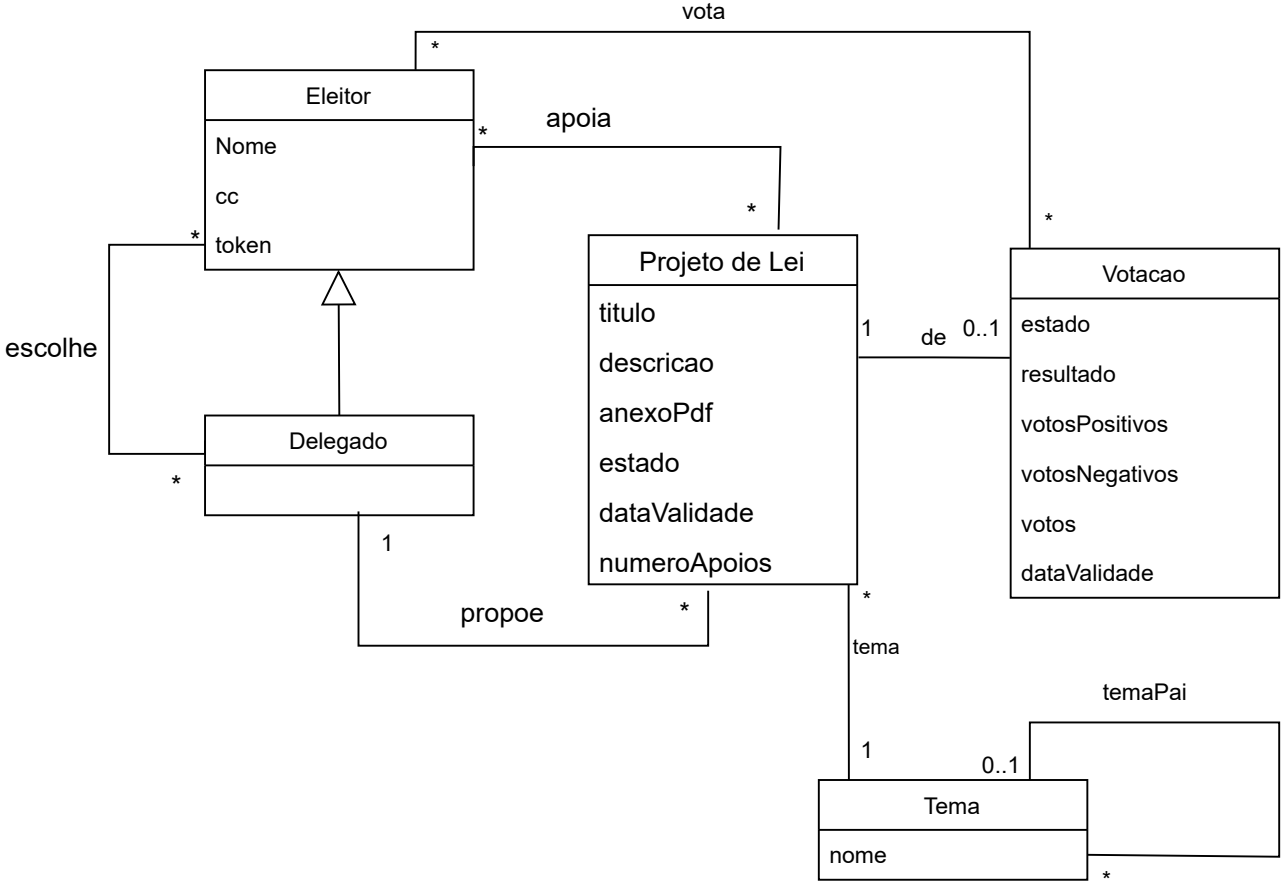
nativeapp

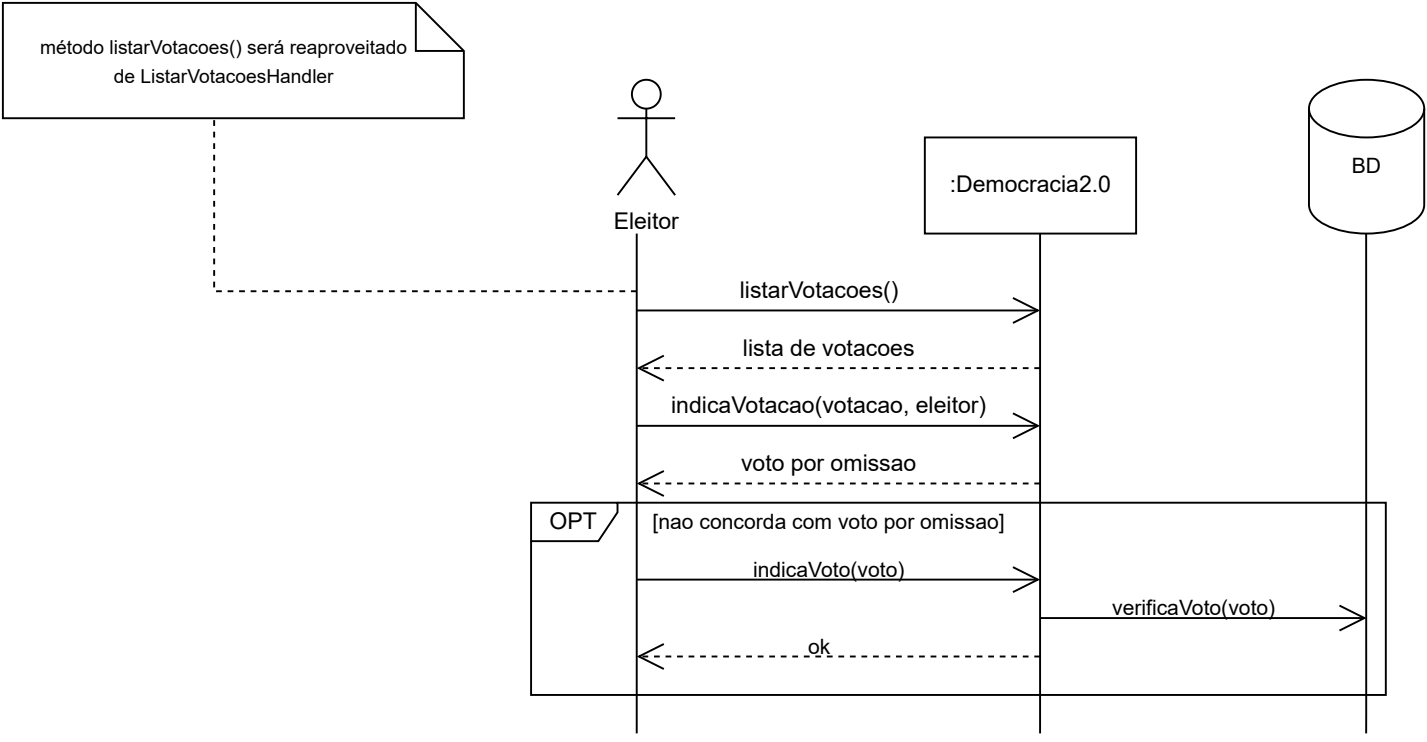
controllers



models




vota
*
Eleitor
Nome
cc
token
*
apoia
*
*
*
escolhe
*
Delegado
*
1
propoe
*
Projeto de Lei
titulo
descricao
anexoPdf
estado
dataValidade
numeroApoios
1
de
0..1
Votacao
estado
resultado
votosPositivos
votosNegativos
votos
dataValidade
*
tema
1
temaPai
0..1
Tema
nome
*


método listarVotacoes() será reaproveitado de ListarVotacoesHandler
Eleitor
:Democracia2.0
BD
listarVotacoes()
lista de votacoes
indicaVotacao(votacao, eleitor)
voto por omissao
OPT
[nao concorda com voto por omissao]
indicaVoto(voto)
verificaVoto(voto)
ok

Relatório

Escolha de herança: Foi escolhido o Single Table para o mapeamento entre as entidades Eleitor e Delegado, pois a tabela do Delegado não iria possuir novas colunas comparativamente com a tabela do Eleitor.

Criação de EleitorDelegadoAssociacao: A relação entre Eleitores e Delegados necessitava de um atributo adicional, o Tema para o qual o Eleitor escolhe um Delegado. Tendo isso em conta, decidimos criar uma classe auxiliar que representasse essa associação. Essa classe tem um EleitorDelegadoAssociacaoId (Embeddable) que agrupa o id do Eleitor e do Tema, visto que um Eleitor apenas pode escolher um Delegado para cada Tema.
A classe EleitorDelegadoAssociacao tem então um @ManyToOne para Eleitor, Delegado e Tema (cada Eleitor, Delegado e Tema poderão ter múltiplos EleitorDelegadoAssociacao, no entanto, cada EleitorDelegadoAssociacao só poderá ter um Eleitor, um Delegado e um Tema.), pela mesma razão, nas classes Eleitor, Delegado e Tema existe um Set<EleitorDelegadoAssociacao> anotado com @OneToMany, mapeados com mappedBy para o atributo correspondente da classe EleitorDelegadoAssociacao.

Escolha de associação entre Eleitores e Projetos de Lei: A classe Eleitor tem um Set<ProjetoDeLei> anotado com @ManyToMany, isto pois um Eleitor poderá apoiar múltiplos ProjetoDeLei, enquanto que um ProjetoDeLei poderá ser apoiado por múltiplos Eleitor. Assim, o ProjetoDeLei também possui um Set<Eleitor> anotado com @ManyToMany. Estas anotações visam a criação de uma tabela auxiliar que contém FKs para as PKs do Eleitor e ProjetoDeLei. Foi usado mappedBy=”apoiantes” na classe Eleitor para tornar esta relação bidirecional.
De forma a que os projetoDeLeiApoiados do Eleitor sejam imediatamente carregados assim que se carrega o objeto, foi colocado a anotação “fetch = FetchType.EAGER” no atributo.

Criação de Voto: A relação entre Eleitores e as Votacoes onde este votou precisava de ter em conta também qual o valor do voto. Desta forma, foi criada uma classe auxiliar Voto. Esta classe tem um VotoId (Embeddable) que agrupa o id do Eleitor e o id da Votacao, uma vez que, o Eleitor só pode votar uma vez em cada Votacao.
A classe Voto, para além de ter o valor do voto, tem as anotações @ManyToOne para os atributos Eleitor e Votacao (cada Eleitor e cada Votacao poderão ter vários votos, no entanto, cada Voto só poderá ter um Eleitor e uma Votacao), pela mesma razão, nas classes Eleitor e Votacao, existe um Set<Voto> com a anotação @OneToMany e mapeado com mappedBy para o atributo correspondente da classe Voto.

Escolha da associação entre Delegados e Projetos de Lei: A classe Delegado tem um Set<ProjetoDeLei> anotado com @OneToMany, isto porque um Delegado poderá propor vários ProjetosDeLei, enquanto que um ProjetoDeLei pode ser proposto por apenas um Delegado. Assim, o ProjetoDeLei tem sempre um Delegado proponente, anotado com @ManyToOne. Aqui pretendeu-se na tabela dos ProjetoDeLei, ter em cada registo uma FK para o Delegado que propôs o ProjetoDeLei. Para isto acrescentou-se a anotação @JoinColumn no Delegado proponente, na classe ProjetoDeLei. Como também se pretendia tornar a relação bidirecional, foi usado mappedBy=”delegadoProponente”, na classe Delegado.

Escolha da Associação entre ProjetoDeLei e Tema: Visto que cada ProjetoDeLei precisa de estar associado a um só Tema, é necessário que a classe ProjetoDeLei possua um Tema como atributo com a anotação @ManyToOne, sendo que cada Tema pode estar associado a vários ProjetosDeLei. Como só será preciso saber a que Tema está associado o ProjetoDeLei, anotamos o atributo Tema da classe do ProjetoDeLei com um @JoinColumn(name=”tema_id”, nullable=false), para que não seja criada uma tabela auxiliar para esta associação. Assim, a tabela ProjetoDeLei terá mais uma coluna com uma FK que será a PK do Tema (esta mesma coluna não poderá ter valor null, visto que é sempre preciso um Tema para cada ProjetoDeLei).

Escolha da Associação entre ProjetoDeLei e Votacao: Quando atinge 10000 apoios, o ProjetoDeLei poderá abrir uma Votacao. Desta forma, haverá para cada Votacao um atributo ProjetoDeLei anotado com @OneToOne e a mesma notação será utilizada no atributo Votacao na classe ProjetoDeLei. Para que a relação seja bidirecional foi anotado o atributo Votacao da classe ProjetoDeLei com um mappedBy=”projetoDeLei”. E para que não seja criada uma tabela auxiliar para representar a associação, foi anotado o atributo ProjetoDeLei da classe Votacao com um @JoinColumn(name=”projeto_de_lei_id”, referencedColumn=”id”), assim a tabela Votacao possuirá uma coluna que conterá como FK a PK do ProjetoDeLei.

EstadoValidade e ResultadoVotacao: Tanto um ProjetoDeLei como uma Votacao têm um EstadoValidade que representa se o mesmo está atualmente aberto ou fechado, sendo EstadoValidade um enumerado, usou-se @Enumerated(EnumType.ORDINAL) pois, apesar de guardar como EnumType.STRING ser mais estável, não se espera que sejam adicionados novos valores a EstadoValidade, assim essa estabilidade pode ser trocada pela maior eficiência de EnumType.ORDINAL. O mesmo se passa para o ResultadoVotacao de uma Votacao, em que os valores são apenas rejeitada e aprovada, não se esperando a adição de novos.

Escolha de Associacao entre Tema e o seu Tema pai: Um tema poderá ter um tema pai, de forma a que possam ser organizados numa estrutura em árvore. Muitos Temas poderão ter o mesmo pai, assim, o atributo temaPai na classe Tema foi anotado com @ManyToOne.

Campos id em Eleitor, Delegado, ProjetoDeLei, Tema e Votacao: Foi decidido que os Ids das Entidades seriam gerados automaticamente, sendo assim os campos anotados com @Id, marcando-os como chave primária e @GeneratedValue(strategy = GenerationType.AUTO) indicando que deverá ser escolhida uma estratégia apropriada para a base de dados em questão.

Escolha de Entidades: Foi decidido que as classes Eleitor, Delegado, ProjetoDeLei, Tema e Votacao seriam entidades, uma vez que, ou possuem atributos suficientes que as caracterizem ou são um conceito do domínio, sendo assim marcadas com @Entity. As classes EleitorDelegadoAssociacao e Voto foram igualmente marcadas com @Entity para que sejam criadas tabelas que representem a associação em causa e os seus atributos.

Relatório CSS

2ª FASE

ALEXANDRE FIGUEIREDO FC57099

AFONSO SANTOS FC56368

RAQUEL DOMINGOS FC56378

Índice

1. Desenvolvimento da aplicação Web com rendering Server-Side

Primeiramente, para a realização de uma aplicação web com rendering Server-Side, foram criados controllers para cada tópico principal dos casos de uso a implementar, i.e, foram criados o WebVotacaoController, o WebProjetoDeLeiController e o WebEleitorController, os quais lidam com os casos de uso relativos a votações, projetos de lei e eleitores/delegados, respetivamente.

Será agora explicado como foi desenvolvida a aplicação WEB com rendering Server-Side para os casos de uso identificados com F2W.

1.1. Listar as Votações em Curso

O caso de uso **Listar as Votações em Curso**, foi implementado como um pedido GET /votacoes-em-curso, uma vez que se trata apenas de apresentar a listagem de todas as votações em curso. É então mostrada uma página HTML ao utilizador com uma tabela onde o mesmo pode consultar os diversos atributos das votações. Nesta página poderá também ser redirecionado para outras páginas da aplicação onde pode obter o seu voto por omissão para uma das votações ou votar diretamente numa votação. Casos de uso que serão analisados na parte seguinte deste relatório. Se não houver qualquer votação em curso, será apresentada uma mensagem a indicá-lo, não sendo mostrados botões ou links que permitam o redireccionamento para os casos de uso mencionados previamente.

1.2. Consultar Projetos de Lei

O caso de uso **Consultar Projetos de Lei** inclui a apresentação da lista de projetos de lei disponíveis, assim como a apresentação dos detalhes de um dado projeto de lei. Como as duas partes se tratam apenas de visualização de conteúdo, foram ambas implementadas como pedidos GET. Assim, para obter a lista com todos os projetos de lei, deve ser feito um pedido GET /projetos-de-lei. Contudo, para visualizar os detalhes de um projeto específico, é necessária a sua identificação no URL do pedido GET, portanto o id do Projeto de Lei é adicionado ao URL, ficando: /projetos-de-lei/{projeto-id}. Na página devolvida pelo GET /projetos-de-lei, o utilizador consegue unicamente ver o id, título e tema dos projetos de lei, mas é possível carregar na opção detalhes

para ser redirecionado para a página devolvida pelo pedido GET /projetos-de-lei/{projetoId}, onde pode consultar todas as informações associadas ao projeto de lei em questão. Os detalhes de um projeto de lei também poderão ser acedidos a partir da lista de votações (caso de uso anterior), uma vez que, para cada votação da lista é apresentado o nome do projeto de lei associado, o qual é um link para os detalhes do projeto de lei em questão. No caso em que o projeto de lei com o id dado não se encontre disponível, uma página de erro 404 (NOT FOUND) será apresentada.

1.3. Apresentar um Projeto de Lei

Para a concretização do caso de uso **Apresentar um Projeto** de Lei, decidiu-se que seria implementado um pedido POST /projetos-de-lei, pois trata-se da adição de um novo projeto de lei ao conjunto de projetos de lei já existentes, enviando um objeto com a totalidade do conteúdo do projeto de lei a ser criado. É também enviado MultipartFile como @RequestParam para que o controller possa ter acesso ao ficheiro pdf submetido pelo utilizador. Nesta função do controller, podem ocorrer diversos erros devido a maus pedidos, (O tema e/ou o delegado proponente do projeto de lei apresentado podem não existir), e nesse caso o utilizador é redirecionado para uma página com o erro 404 (NOT FOUND), sendo indicado o sucedido. Caso a data apresentada para o projeto de lei seja superior a um ano, é indicado o erro 406 (NOT ACCEPTABLE), pois a mesma não é aceite pelo servidor. Por último, se a transferência do ficheiro pdf para o servidor falhar, é apresentada uma página com o erro 500 (INTERNAL SERVER ERROR).

Vale a pena mencionar que, para o utilizador poder fornecer as informações do novo projeto de lei, é necessária a apresentação de um formulário, e que o mesmo pode ser obtido a partir de um pedido GET /projetos-de-lei/apresenta.

1.4. Votar numa Proposta

No caso de uso **Votar numa Proposta**, o utilizador pode escolher tanto obter um voto por omissão, como votar. Para a obtenção do voto por omissão, realiza-se um pedido GET /voto-omissao ao servidor, pedido este que irá renderizar a página com o valor do voto (ou um aviso que não existe voto). As situações de erro neste caso são a não existência do eleitor ou da votação, sendo tratadas com uma apresentação de uma página com o erro 404 (NOT FOUND). Já para ser atribuído um novo voto, será realizado um POST /votos, uma vez que, se trata de acrescentar um novo voto à estrutura com os votos já existentes, enviando então um VotoDTO com os campos

necessários. O tratamento do caso onde o eleitor ou a votação não existem é igual ao anterior. No caso do eleitor já ter votado nesta votação, é apresentada uma página com o erro 409 (CONFLICT). Para que ambas as ações anteriores possam ser realizadas, é necessária a aquisição de informação (por exemplo, votação relativa ao voto por omissão, ou, valor do voto adicionado), que será efetuada através de formulários, um para as informações do voto por omissão e outro para as informações do novo voto, sendo que estes serão obtidos por pedidos GET /voto-omissao-forms e GET /votos, respetivamente. Tanto o formulário para a obtenção do voto por omissão como o formulário para um eleitor poder votar numa votação podem ser acedidos a partir da página que lista as votações.

1.5. Escolher Delegado

Para o caso de uso que permite a um eleitor escolher um delegado (**Escolher Delegado**), é realizado um pedido POST /escolhe-delegado, visto que, será criada uma nova EleitorDelegadoAssociacao com os atributos tema, eleitorCC e delegadoCC. Estes atributos foram recebidos via um formulário, obtido com a realização de um pedido GET /escolhe-delegado. Caso algum dos campos introduzidos no formulário não exista, o tratamento do erro é igual aos casos anteriores.

1.6. Apoiar Projetos de Lei

Para o caso de uso **Apoiar Projetos de Lei**, é realizado um POST /projetos-de-lei/apoia. Apesar da adição de um apoio se tratar da mudança de atributos no projeto de lei e no eleitor, foi decidido que seria realizado um POST, pois as informações relativas ao apoio devem ser obtidas por um formulário HTML, o qual não suporta um pedido PATCH (que no caso seria o mais correto). Para a realização deste endpoint foi criada a classe EleitorApoiaProjetoDTO que tem os campos necessários para poder ser realizada a operação. Havendo esta classe, pode ser passado ao endpoint um objeto EleitorApoiaProjetoDTO como @ModelAttribute. A página com o formulário necessário para um eleitor poder apoiar um projeto de lei pode ser acedida com GET /projetos-de-lei/apoia. Para além das situações de erro diversas vezes já descritas, pode ocorrer que o eleitor já tenha apoiado este projeto de lei anteriormente ou que o projeto de lei já tenha expirado, sendo nesses casos mostrada uma página indicativa do erro com o código 409 (CONFLICT).

1.7. Observações - Thymeleaf

Para todas as páginas HTML apresentadas, os controllers respectivos passam à página HTML os objetos a apresentar e/ou modificar, por meio do objeto Model. A partir daí, com recurso ao Thymeleaf, a página HTML consegue renderizar corretamente apresentando as informações necessárias, ou modificando os objetos que mais tarde serão passados aos controllers por meio de um POST.

2. Desenvolvimento da API REST

Para o desenvolvimento de uma API REST, foram criados controllers para cada tópico principal dos casos de uso a implementar, i.e, foram criados o RestVotacaoController e o RestProjetoDeLeiController, os quais lidam com os casos de uso relativos a votações e projetos de lei, respetivamente.

De forma a distinguir os pedidos da aplicação Web dos pedidos da REST API, foi colocado /api/ antes dos endereços que definem os pedidos REST.

Será agora explicado como é se desenvolveu a REST API para os casos de uso identificados com F2R.

2.1. Listar as Votações em Curso

O caso de uso **Listar as Votações em Curso**, foi implementado como um pedido GET api/votacoes/votacoes-em-curso, pois trata-se apenas da obtenção de uma lista de VotacaoDTO. Não foram tratadas situações de erro visto que caso não existam votações em curso, é enviada uma lista vazia.

2.2. Consultar Projetos de Lei

No caso de uso **Consultar Projetos de Lei**, são implementados três pedidos GET:

- GET api/projetos-de-lei/projetos-de-lei-nao-expirados, que devolve uma lista de ProjetoDeLeiDTO.
- GET que retorna um ProjetoDeLeiDTO específico. É, então, necessária a identificação do projeto de lei a visualizar no pedido GET, portanto o id do Projeto de Lei é adicionado ao URL, ficando: api/projetos-de-lei/{projeto-id}. Ao usar este endpoint, pode ser fornecido um id que não corresponde a nenhum projeto na base de dados, nessa situação é devolvido um código de erro 404.

- GET api/projetos-de-lei/download-pdf/{id}, que permite descarregar o byte[] representativo do pdf associado a um dado projeto de lei, correspondendo o id do endereço ao id do projeto de lei. Bem como no endpoint anterior, caso seja fornecido um id que não represente um projeto na base de dados, é devolvido um código de erro 404.

2.3. Apoiar Projetos de Lei

Para o caso de uso **Apoiar Projetos de Lei**, é realizado um pedido PATCH api/projetos-de-lei/apoia/{eleitorCC}/{projetoDeLeiId}, pois para a adição de um apoio, efetua-se modificações de atributos tanto no projeto de lei (identificado com projetoDeLeiId no endereço) como no eleitor (identificado com eleitorCC no endereço). Ao usar este endpoint, caso seja passado um projetoDeLeiId e/ou um eleitorCC que não corresponda a nenhum dos guardados na base de dados, é enviado um código de erro 404. No caso de um eleitor tentar apoiar um projeto de lei que já tenha apoiado previamente, é enviado o código de erro 409 (CONFLICT) visto que o pedido entra em conflito com o estado da aplicação, onde é apenas permitido um apoio por cidadão.

2.4. Votar numa Proposta

Para efetuar o caso de uso **Votar numa Proposta**, foram implementados um pedido GET e um pedido POST.

- O pedido GET api/votacoes/voto-omissao/{eleitorCC}/{votacaoId} é responsável por devolver um VotoDTO, que corresponde a um voto da votação identificada no endereço por votacaoId. Este voto é o voto atribuído por um delegado, sendo que este delegado é aquele escolhido, para o tema da votação, pelo eleitor identificado pelo cartão de cidadão no endereço em eleitorCC. Desta forma, o voto obtido, será o voto por omissão do eleitor identificado para a votação em questão. Caso o eleitorCC e/ou votacaoId não existam, o código da resposta obtida será 404 (NOT FOUND). Este código também será obtido caso o delegado não tenha votado, não havendo assim voto por omissão para ser retornado.
- O pedido POST api/votacoes/votos, permite adicionar um novo voto aos já existentes, e, para tal, recebe no corpo do pedido um objeto com os campos do voto, nomeadamente, cartão de cidadão do eleitor, id da votação e o valor do voto. Caso a atribuição do voto seja bem sucedida, irá devolver o código de resposta 200 (OK) e no corpo da mesma, o voto adicionado. No entanto, caso o

eleitor correspondente ao cartão de cidadão no pedido e/ou a votação correspondente ao id no pedido não existam, é devolvido na resposta o código 404 (NOT FOUND). Na situação em que um voto já tinha sido atribuído pelo eleitor àquela votação, será devolvido um código de resposta 409 (CONFLICT).

2.5. Observações – Service Layer

Tanto os controllers usados pela aplicação web, bem como os controllers usados pela REST API, recorrem às classes EleitorService, ProjetoDeLeiService e VotacaoService. Estas classes foram criadas durante esta etapa do projeto, de forma a servirem como uma abstração para a camada de apresentação, para que a mesma não tenha de estar ligada aos detalhes mais íntimos da camada de negócio, tal como todos os handlers existentes e os seus parâmetros. Para isto anotaram-se os handlers com @Component de forma a poderem ser injetados no Services, e também anotou-se @Component nos Services de forma a estes poderem ser injetados nos controllers. Desta forma, a camada de apresentação pode usar uma camada intermédia (Service Layer) e usufruir da sua abstração.

3. Desenvolvimento da Aplicação Nativa com JavaFX

Como os casos de uso a implementar na Aplicação Nativa estavam relacionados a votações, votos e projetos de lei, foram criadas as classes Votacao, Voto e ProjetoDeLei no módulo nativeapp, as quais servem o propósito de representar os respectivos conceitos na Aplicação Nativa, com os campos do tipo Property necessários para a sua apresentação nas páginas. A partir daqui, qualquer menção às classes Votacao, Voto e ProjetoDeLei estar-se-á a referir às classes do módulo nativeapp e não às classes pertencentes ao backend (criadas durante a 1ªfase do projeto).

Para cada ficheiro .fxml foi criado um Controller correspondente. Desta forma, o controller ficará responsável pela inicialização dos diversos campos da página, carregando as informações necessárias.

Foram também criadas as classes ProjetosDeLeiModel, VotacoesModel e VotosModel, que ficaram responsáveis pela obtenção dos dados a serem apresentados na aplicação, usando para isso a REST API descrita anteriormente. De forma semelhante, quando é necessário modificar algum objeto, o controller chama a função apropriada do model,

passando os atributos necessários. Por sua vez o model irá então usar a REST API para comunicar com o servidor e realizar as modificações necessárias.

A aplicação começa por apresentar um menu principal, onde se encontram os botões que redirecionam para as páginas de diferentes casos de uso (Listar Votações, Consultar Projetos de Lei, Apoiar Projeto de Lei e Votar numa Proposta) e correm a função initController() do Controller correspondente a cada caso de uso. Todas as páginas da aplicação, com exceção do menu principal, também apresentam um header com estes botões para que o utilizador consiga navegar facilmente pelos vários casos de uso. Contudo, caso o utilizador se encontre numa página que corresponda a um desses botões, o mesmo não será apresentado no header.

Será agora explicado como foi desenvolvido a aplicação desktop usando JavaFX que tem implementados os casos de uso identificados com F2D.

3.1. Listar as Votações em Curso

Para implementar o caso de uso **Listar as Votações em Curso**, recorreu-se à classe VotacoesModel, a qual possui uma lista de votacoes (ObservableList<Votacao>). Ao ser chamado o método getVotacaoList(), esta lista será preenchida com todas as votações em curso existentes na base de dados, sendo que para tal, será realizado um pedido GET /api/votacoes/votacoes-em-curso à API REST descrita anteriormente. Quando se executar a funcao initController() do controller relativo ao caso de uso (ListarVotacoesController), este irá obter, a partir do VotacoesModel, a lista de votações, e irá apresentá-las em formato de tabela numa página fxml.

3.2. Consultar Projetos de Lei

Para **Consultar Projetos de Lei**, recorreu-se à classe ProjetoDeLeiModel para a obtenção dos projetos de lei da base de dados. Sendo que para o caso em que é necessário obter todos os projetos de lei não expirados, será realizado um pedido GET /api/projetos-de-lei/projetos-de-lei-nao-expirados à API REST, preenchendo, assim, a lista de projetos de lei da classe (ObservableList<ProjetodDeLei>), a qual será obtida pelo ConsultarProjetosDeLeiController (quando chamada a função initController) e utilizada para popular a tabela apresentada na interface da aplicação. Já no caso em que se pretende obter os detalhes de um projeto de lei, o mesmo pode ser concretizado ao clicar no link de detalhes apresentado para cada projeto da tabela de projetos de lei. Esse link executará a função initModel do ProjetoDetalhesController, definindo assim, num atributo desta classe, o id do projeto de lei que será consultado. O link chama também a função initController, a qual utiliza a função getProjetoDTO de

VotacoesModel enviando como parâmetro o id do projeto de lei, definido com o initModel(). A função getProjetoDTO realiza um pedido GET /api/projetos-de-lei/{projetoId} à API REST, no qual projetoId é o parâmetro recebido. As informações relativas ao ProjetoDeLeiDTO serão apresentadas na página fxml.

3.3. Apoiar Projetos de Lei

No caso de uso **Apoiar Projetos de Lei**, a página fxml apresentada é um formulário no qual devem ser colocados o id do projeto de lei e o cartão de cidadão do eleitor. Quando o formulário é submetido, é chamada a função apoia() do ApoiaProjetoDeController, a qual invocará a função apoiaProjeto() do VotacoesModel, enviando como parâmetros, as informações obtidas do formulário. Esta última função enviará um pedido POST /api/projetos-de-lei/apoia/{eleitorId}/{projetoId} para a API REST, sendo eleitorId e projetoId os parâmetros recebidos, e devolve o código da resposta que obteve do pedido. Após a submissão do formulário, é apresentada na página fxml, uma mensagem de acordo com o código da resposta. Assim, caso o pedido seja bem sucedido e o código de resposta for 200 (OK), será apresentada uma mensagem de sucesso, e caso o pedido seja mal sucedido e o código de resposta for 404 (NOT FOUND) ou 409 (CONFLICT), serão apresentadas mensagens de acordo com erro em causa. É também apresentada uma mensagem de erro caso os campos do formulário não estejam preenchidos, ou caso o campo do id do projeto de lei esteja preenchido com algo que não um inteiro.

3.4. Votar numa Proposta

Para obter o voto omissão no caso de uso **Votar numa Proposta**, foi também apresentado um formulário com os campos respetivos ao cartão de cidadão do eleitor e ao id da votação. Quando o formulário é submetido, é invocada a função setVoto() do VotosModel, a qual recebe como parâmetros o conteúdo dos campos do formulário. Esta função faz um pedido GET /api/votacoesi/voto-omissao/{eleitorCC}/{votacaoId}, sendo eleitorCC e votacaoId os parâmetros recebidos. O VotosModel possui um atributo currentVoto, em que caso o código da resposta ao pedido GET seja 200 (OK), este atributo será definido com o voto também recebido na resposta. Já em caso do código da resposta ser 404, o currentVoto será colocado a null. A submissão do formulário também redireciona para a página de visualização do voto e corre o initController() do VisualizarVotoController. Ao executar a função, o controller irá obter o currentVoto do VotosModel que foi definido aquando a submissão do formulário, o que é possível de realizar pelo facto de VotosModel ter

sido implementado como um Singleton, havendo assim, apenas uma instância do mesmo. O VisualizarVotoController quando obtém o voto, verifica se o mesmo é null, e se for, é apresentada uma mensagem referindo que o voto não existe, caso contrário, é apresentado o valor do voto.

Já para implementar na nativeapp a funcionalidade de votar do mesmo caso de uso, é novamente apresentado um formulário também com os campos cartão de cidadão do eleitor e id de votação, e, para além destes, com uma ChoiceBox que tem como opções rejeitar ou aprovar o voto. Ao submeter o formulário, é chamada a função votar() do VotarController, a qual cria um VotoDTO com o conteúdo dos campos do formulário, sendo que este voto é depois incluído no corpo de um pedido POST /api/votacoes/votos à API REST. Após ser recebida a resposta do pedido, e caso a mesma tenha código 200 (OK), é apresentada uma mensagem de sucesso na página fxml. Contudo, caso o código seja 404 (NOT FOUND) ou 409 (CONFLICT), será apresentada uma mensagem do respetivo erro. Na situação em que é efetuada a submissão do formulário com os seus campos vazios, ou em que é colocado um valor que não é um inteiro no campo do id da votação, é apresentada também uma mensagem de erro.

3.5. Observações - DTOs

Os pedidos realizados pelas classes da aplicação JavaFX à REST API terão como resposta objetos DTO definidos na primeira fase do projeto. Como se pretende que a aplicação JavaFx seja independente da aplicação servidor, foi criada um diretório “dtos” dentro da pasta nativeapp, onde foram criadas as classes ProjetoDeLeiDTO, VotacaoDTO e VotoDTO. Estas classes nada mais são que cópias das classes DTO do projeto servidor. Desta forma, a aplicação JavaFX poderá depender destas classes e não ter qualquer dependência de classes do servidor.

4. Casos de Uso com Scheduled Tasks

Existiam dois casos de uso os quais teriam de ser apresentados com recurso às Scheduled Tasks do Spring. Esses casos de uso são o “Fechar projectos de lei expirados” e o “Fechar uma Votação”. Para uma função poder ser anotada com @Scheduled, a mesma necessita de ser void e não esperar argumentos.

Será agora explicado como foram implementados os casos de uso identificados com C à custa de Scheduled Tasks no Spring.

4.1. Fechar projetos de lei expirado

Para o **Fechar projectos de lei expirados**, como a função fecharProjetosDeLei() já era void e não esperava argumentos foi simplesmente adicionada a anotação @Scheduled(fixedRate = 15000). O uso de fixedRate teve o intuito de não permitir que duas execuções da função ocorressem em simultâneo. Assim, após a execução da função fecharProjetosDeLei(), a mesma voltará a ser executada 15s depois.

4.2. Fechar Votação

Para o caso de uso **Fechar Votação**, a função fecharVotacao esperava uma Votacao como argumento, assim, este handler teve de ser reestruturado para o caso de uso poder ser implementado com recurso a Scheduled Tasks. Assim, foi criada a função fecharVotacoesForaValidade, anotada com @Scheduled(fixedRate = 15000). Esta função, obtém todas as votações fora de validade e usa a função fecharVotacao para as fechar.
O uso de fixedRate teve o mesmo intuito que no caso de uso anterior.
O uso de 15s de diferença entre cada execução destas funções deve-se ao facto de o grupo ter considerado 15s um intervalo de tempo sensato, que permitisse uma constante atualização do estado da aplicação, ao mesmo tempo que não sobrecarregasse o servidor com execuções de funções que em muitas vezes podiam não realizar qualquer trabalho (Por não haver projetos de lei ou votações para fechar).

5. Observações Gerais

Como ao inicializar o servidor, o mesmo não vai ter nenhuns dados na base de dados, foi criado um controller com o nome RestPopulateController. Este controller não tem qualquer uso na aplicação, sendo apenas usado para popular a base de dados.
Assim, se for feito um pedido GET ao endpoint api/populate/test será colocado na BD um Eleitor, um Delegado, um Tema, um ProjetoDeLei, uma Votacao (Votacao do ProjetoDeLei mencionado anteriormente) e um voto, voto este do Delegado na Votacao.