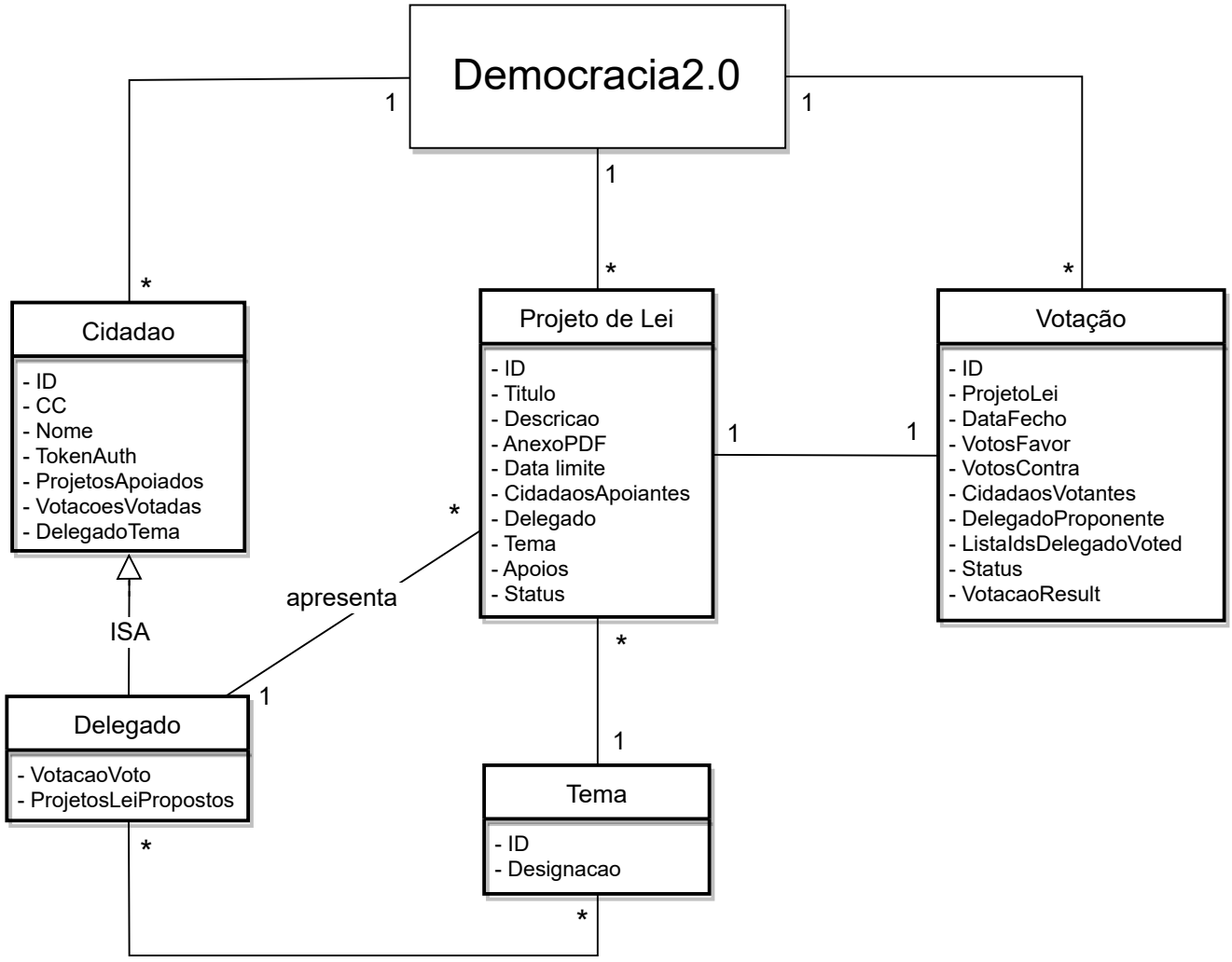
Modelo Domínio



SSD Caso de Uso J

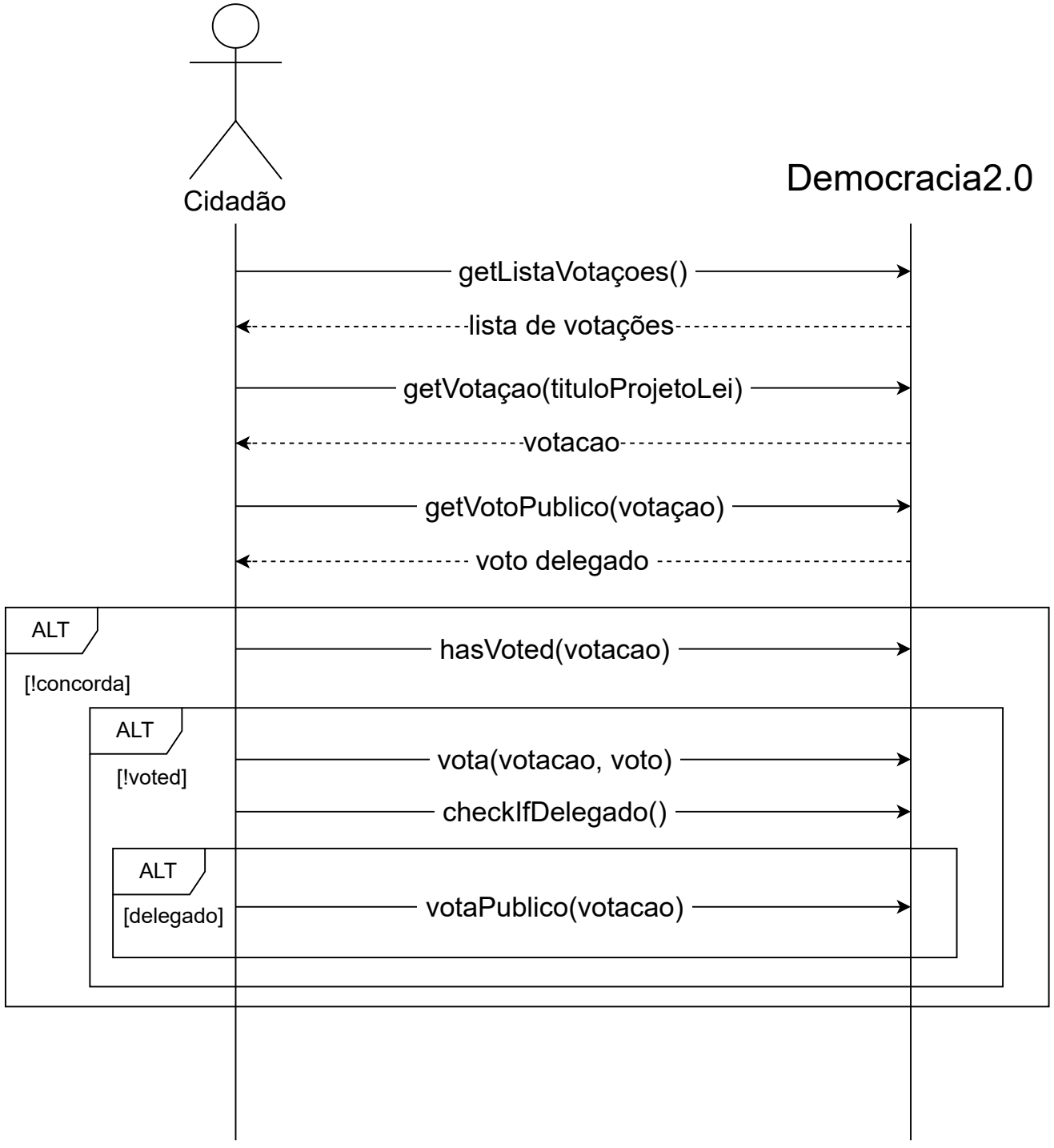


Diagrama de Classes

facade

services

ApoiarProjetoLeiService
- repositorioCidadao: CidadaoRepository
- repositorioProjetoLei: ProjetoLeiRepository
- reporisotioVotacao: VotacaoRepository

+ apoia(String, int): void
+ upToVotacao(String): boolean
+ createVotacao(ProjetoLei, LocalDateTime): void
+ existsVotacao(String): boolean

Delegado, Cidadao, ProjetoLei, Votacao, Status, ProjetoLeiRepository, VotacaoRepository, CidadaoRepository

CidadaoService
- repositorioCidadao: CidadaoRepository
- reporisotioDelegado: DelegadoRepository

+ adicionaCidadao(int,String,Long): void
+ listarCidadaos(): List<Cidadao>
+ findByCC(int): Cidadao
+ apresentaComoDelegado(Cidadao): void
+ findDelegadoByCC(int): Delegado

Cidadao, Delegado, CidadaoRepository, DelegadoRepository

VotarVotacaoService
- repositorioCidadao: CidadaoRepository
- repositorioDelegado: DelegadoRepository
- reporisotioVotacao: VotacaoRepository

+ votaVotacao(String,Long, Voto): void

Cidadao, Delegado, CidadaoRepository, DelegadoRepository, Votacao, Voto, VotacaoRepository

FecharVotacaoService
- repositorioCidadao: CidadaoRepository
- reporisotioVotacao: VotacaoRepository
- repoditorioDlegado: DelegadoRepository

+ fecharVotacao(String): void
+ closeVotacao(String): void
+ resultadoVotacao(String,VotacaoResult): void
+ updateDataLimite(Long,LocalDateTime): void

Cidadao, Delegado, CidadaoRepository, VotacaoRepository, Status, Votacao, VotacaoResult, Voto

ProjetoLeiService
- repositorioProjetoLei: ProjetoLeiRepository
- reporisotioTema: TemaRepository

+ findByTitulo(String): ProjetoLei
+ listarProjetosLei(): List<ProjetoLei>
+ consultarProjetoLei(String): String
+ createTema(String): void
+ findTemaByDesignacao(String): Tema
+ updateApoios(Long,int): void
+ updateDataLimite(Long,data):void

ProjetoLei, Status, ProjetoLeiRepository, TemaRepository, Tema

ApresentarProjetoLeiService
- repositorioProjetoLei: ProjetoLeiRepository

+ apresentarProjetoLei(String,String,byte[], LocalDateTime,Delegado,Tema): void

Status, ProjetoLei, Tema, ProjetoLeiRepository, Delegado

VotacaoService
- reporisotioVotacao: VotacaoRepository

+ findByProjetoLeiTitulo(String): Votacao
+ getListaVotacoesAbertas(): List<Votacao>

Status, Votacao, VotacaoRepository

EscolherDelegadoService
- repositorioCidadao: CidadaoRepository

+ escolheDelegado(int, int, String): void
+ findDelegadoCCByTema(String): int

Cidadao, Delegado, CidadaoRepository, DelegadoRepository

FecharProjetosLeiExpiradosService
- repositorioProjetoLei: ProjetoLeiRepository

+ fecharProjetosLeiExpirados(): void

Status, ProjetoLei, ProjetoLeiRepository

DemoApplication

exceptions

- VotacaoNotFoundException
- TemaNotFoundException
- InvalidDateException
- ApplicationException
- CidadaoNotFoundException
- ProjetoLeiNotFoundException

business

entities

Cidadao
- id: Long
- nrCC: int
- nome: String
- tokenAuth: long
- projetosApoiados: List<ProjetoLei>
- votacoesVotadas: List<Votacao>
- delegadoCCTemaDesignacao: List<String>

+ addApoioProjeto(ProjetoLei): void
+ isApoiante(ProjetoLei): boolean
+ addDelegadoTema(int,String): void
+ hasDelegado(int): boolean
+ getDelegadoByTema(String): int
+ addVotoVotacao(Votacao): void
+ hasVoted(Votacao): boolean

ProjetoLei
- id: Long
- titulo: String
- descricao: String
- anexoPDF: byte[]
- dataLimite: LocalDateTime
- cidadaosApoiantes: List<Cidadao>
- delegado: Delegado
- tema: Tema
- apoios: int
- status: Status

+ isValidDate(LocalDateTime): boolean
+ hasExpired(LocalDateTime): boolean
+ addApoio(): void
+ addApoioDelegado(): void

Votacao
- id: Long
- projetoLei: ProjetoLei
- dataFecho: LocalDateTime
- votosAFavor: int
- votosContra: int
- cidadaosVotantes: List<Cidadao>
- delegadoProp: Delegado
- listaIdsDelegadoVoted: List<int>
- status: Status
- votacaoResult: VotacaoResult

+ addVoto(Cidadao,Voto): void
+ getResultadoVotacao(): int
+ addDelegadoVoted(int): void
+ hasDateExpired(): boolean

Delegado
- votacaoVoto: List<String>
- projetosLeiPropostos: List<ProjetoLei>

+ addVotoPublico(Votacao,Voto): void
+ getVotoByVotacao(Long): Voto

Tema
-id: Long
-designacao: String

+ equals(): boolean

ISA

data types

<<enumeration>> Voto
FAVOR
CONTRA

<<enumeration>> VotacaoResult
APROVADA
REPROVADA

<<enumeration>> Status
ABERTO
FECHADO

repositories

ProjetoLeiRepository
+ findById(int): ProjetoLei
+ findByTitulo(int): ProjetoLei
+ findByStatus(Status): List<ProjetoLei>
+ deleteByTitulo(String): void

VotacaoRepository
+ findById(int): Votacao
+ findByProjetoLeiTitulo(String): Votacao
+ findByStatus(Status): List<Votacao>
+ deleteByProjetoLeiTitulo(String): void
+ existsVotacaoByTitulo(String): boolean

DelegadoRepository
+ findById(int): Delegado
+ findByCC(int): Delegado
+ deleteByCC(int): void
+ existsDelegadoByCC(int): boolean

TemaRepository
+ findById(int): Tema
+ findByDesignacao(String): Tema
+ deleteByDesignacao(String): void

CidadaoRepository
+ findCidadaoById(int): Cidadao
+ findByCC(int): Cidadao
+ deleteByCC(int): void
+ findDelegadoCCTemaDescription(String): String

Decisões tomadas no mapeamento referente às anotações JPA:

Cidadao @Entity @Inheritance (strategy = heritanceType.TABLE_PER_CLASS) para especificar que queremos criar uma tabela para a entidade filha desta entidade (Delegado):

- **id** *@id @GeneratedValue* para indicar que é a primary key e que é gerado automaticamente de forma a garantir que não existem ids iguais entre cidadaos.
- **nrCC** *@Column(nullable=false, unique=true)* para especificar que a coluna não pode ser nula e não existem valores de CCs iguais na tabela.
- **tokenAuth** não foi implementado nesta fase do projeto.
- **List <ProjetoLei> projetosApoiados** *@ManyToMany @JoinTable(name = "cidadao_projetoLei", joinColumns = @JoinColumn(name = "cidadao_id"), inverseJoinColumns = @JoinColumn(name = "projetoLei_id")) @JsonIgnore* serve para guardar todos os projetos apoiados pelo cidadao. Cada cidadão pode apoiar vários projetos de lei e cada projeto de lei pode ser apoiado por vários cidadãos. Criamos uma tabela com o cidadão_id e o projetoLei_id que associa cada apoio de um cidadão a um projeto de lei. Vai ser ignorado este atributo quando um cidadão é acedido/devolvido na API REST.
- **List<Votacao> votacoesVotadas** *@ManyToMany @JoinTable(name = "cidadao_votacao", joinColumns = @JoinColumn(name = "cidadao_id"), inverseJoinColumns = @JoinColumn(name = "votacao_id")) @JsonIgnore* serve para guardar todas as votações votadas pelo cidadao. Cada cidadão pode votar em várias votações e cada votação pode ser votada por vários cidadãos. Criamos uma tabela com o cidadão_id e o votacao_id que associa cada voto de um cidadão a uma votacao. Vai ser ignorado este atributo quando um cidadão é acedido/devolvido na API REST.
- **List<String> delegadoCCTemaDesignação** *@ElementCollection* serve para associar um delegado a um tema quando um cidadão escolhe um delegado para um tema. Cria uma tabela que associa o delegado e um tema através de uma string com o formato "[delegadoId];[temaDesignacao]".

Delegado @Entity extends Cidadao:

- **List<String> votacaoVoto** *@ElementCollection @Column(name = "votacaoVoto") @JsonIgnore* serve para guardar o voto público do delegado para cada votação. Cria uma coluna que associa votação_id e um voto através de uma string com o formato "[votacaoId];[voto]". Vai ser ignorado este atributo quando um delegado é acedido/devolvido na API REST.
- **List<ProjetoLei> projetosLeiPropostos** *@OneToMany(mappedBy="delegado") @JsonIgnore* serve para guardar os projetos de lei propostos pelo delegado. Vai ser ignorado este atributo quando um delegado é acedido/devolvido na API REST.

ProjetoLei @Entity:

- **id** *@id @GeneratedValue* para indicar que é a primary key e que é gerado automaticamente de forma a garantir que não existem ids iguais entre projetos de lei.
- **titulo** *@Column(nullable=false, unique=true)* para especificar que a coluna não pode ser nula e não existem titulos iguais na tabela.
- **descricao** *@Column(nullable = false)* é a descrição do projeto de lei.
- anexoPDF *@Lob @Column(name = "pdf_file", nullable = false) @JsonIgnore* especifica que é um objeto com muitos bytes. Vai ser ignorado este atributo quando um projeto de lei é acedido/devolvido na API REST.
- **dataLimite** *@Column(name = "local_date_time", columnDefinition = "TIMESTAMP") @JsonIgnore* é a data no qual o projeto de lei termina. Vai ser ignorado este atributo quando um projeto de lei é acedido/devolvido na API REST.
- **List<Cidadao> cidadaosApoiantes** *@Transient @JsonIgnore* pois não é sincronizado com a base de dados. É usada para guardar os cidadãos apoiantes do ProjetoLei, e não é guardado na base de dados pois basta guardar o número de apoios na tabela, não quem apoia. Vai ser ignorado este atributo quando um projeto de lei é acedido/devolvido na API REST.
- **delegado** *@ManyToOne @JoinColumn(name = "delegado_id", nullable = false)* pois cada delegado pode apresentar vários projetos de Lei. É o delegado que apresentou o projeto de Lei.
- **tema** *@ManyToOne @JoinColumn(name = "tema_id", nullable = false)* pois podem haver vários projetos de lei com o mesmo tema.
- **status** *@Enumerated(EnumType.STRING)* pois é um Enumerated e queremos guardar a string do Enumerated. Representa o status da votação (ABERTO/FECHADO)

Tema @Entity:

- **id** *@id @GeneratedValue* para indicar que é a primary key e que é gerado automaticamente de forma a garantir que não existem ids iguais entre temas.
- **designacao** *@Column(nullable=false, unique=true)* para especificar que a coluna não pode ser nula e não existem designações iguais na tabela.

Votacao @Entity:

- **id** *@id @GeneratedValue* para indicar que é a primary key e que é gerada automaticamente de forma a garantir que não existem ids iguais entre votações.
- **projetoLei** *@OneToOne(cascade = CascadeType.PERSIST)* pois cada projeto de lei só pode ter uma votação e vice-versa e a opção CascadeType.PERSIST especifica que quando uma entidade é persistida, a outra também.
- **dataFecho** *@Column(name = "local_date_time", columnDefinition = "TIMESTAMP")* é a data no qual a votação fecha automaticamente.
- **votosAFavor** é o número de votos a favor da votação.

- **votosContra** é o número de votos contra da votação.
- **List<Cidadao> cidadaosVotantes** *@ManyToMany(mappedBy="votacoesVotadas") @JsonIgnore* serve para guardar os cidadaos que já votaram na votação. Vai ser ignorado este atributo quando uma votação é acedida/devolvida na API REST.
- **delegadoProp** *@ManyToOne @JoinColumn(name = "delegado_id", nullable = false)* é o delegado que propôs o projeto de lei que se tornou na votação. Cada delegado pode ser proponente de várias votações mas cada votação só tem um delegado proponente.
- **List<Integer> listaIdsDelegadoVoted** *@ElementCollection* guarda os ids dos delegados que já votaram na votação.
- **status** *@Enumerated(EnumType.STRING)* pois é um Enumerated e queremos guardar a string do Enumerated. Representa o status da votação (ABERTO/FECHADO)
- **votacaoResult** *@Enumerated(EnumType.STRING)* pois é um Enumerated e queremos guardar a string do Enumerated. Representa o resultado da votação (APROVADA/REJEITADA/NAO_TERMINADA).

Arquitetura interna do projeto:

Aplicação Web (Casos de Uso F_{2W}):

Foi implementada a camada de apresentação (*templates HTML e Web Controllers*). Os *Controllers* fazem pedidos HTTP usando o padrão MVC no Spring, e usufruem dos Serviços que integram *Queries* de acesso à base de dados implementadas nos repositórios, com as funções presentes nos *Handlers*.

API REST (Casos de Uso F_{2R}):

Foram implementados métodos para satisfazer os casos de uso identificados com a anotação F_{2R}, que posteriormente iriam ser acedidos pela Aplicação Desktop, através de pedidos REST, retornando a informação pedida através dos Serviços.

Aplicação Desktop c/ JavaFx (Casos de Uso F_{2D}):

Para o funcionamento da aplicação desktop, os ficheiros .fxml usam os *Presentation Controllers* para navegar entre as várias cenas da aplicação, navegando entre diferentes ficheiros .fxml. Estes *Controllers* fazem uso do *DataModel*, que define toda a estrutura de dados que é disposta na aplicação. Esta classe lida com as conversões dos objetos que são devolvidos da base de dados para o tipo de objetos a mostrar na app (*ProjetoLeiPresentation* & *VotacaoPresentation*). O modo de acesso à base de dados é através de pedidos REST, feitos nos *Rest Controllers*, que também fazem uso dos Serviços para enviar aos *Presentation Controllers* e ao *DataModel* a informação pedida.

Spring Scheduled Tasks (Casos de Uso *C*):

Através da anotação *@Schedule()* os métodos de fechar votações e projetos de lei expirados passaram a ser executados automaticamente a cada 24h. Foi preciso adicionar a anotação @EnableScheduling no ficheiro de configuração *ThymeleafWebConfig*.

Testes:

No ficheiro de testes *DemoApplicationTest* realizamos testes para cada caso de uso e para cada método dos REST *Controllers*. Os métodos dos WEB *Controllers* não foram testados neste ficheiro, mas é possível serem testados através da aplicação WEB desenvolvida.

NOTA: Para testar os métodos de criar votação criámos a votação diretamente através do *endpoint* do método do *controller*. Para podermos visualizar a criação da votação após o projeto de lei ter apoios suficientes, alterámos a linha de código que avalia esta condição:

```
40          Optional<ProjetoLeiDTO> pl = projetoLeiService.apoia(projetoLeiId, cidadaoId);
41          if (pl.isPresent()) {
42              if (votacaoService.upToVotacao(pl.get().getId()) && !votacaoService.existsVotacao(pl.get().getId())) {
43                  createVotacao(pl.get().getId());
44              }
```

1. *RestProjetoLeiController.apoiaProjetoLei()*

```
82
83          if (votacaoService.upToVotacao(projeto.get().getId()) && !votacaoService.existsVotacao(projeto.get().getId())) {
84              createVotacao(projeto.get().getId());
85          }
```

2. *WebProjetoLeiController.apoiarProjetoLei()*

Ambos os *controllers* alterámos a primeira condição do *if (.upToVotacao())* para avaliar apenas se o Projeto de Lei tinha 2 apoios *(projeto.get().getApoios() == 2).* Desta forma, ao apoiar um Projeto de Lei, desde que não seja o mesmo delegado que a lançou, será criada uma votação.

DataBase:

Populámos a base de dados através da classe DemoApplication, criando diversos cidadãos, delegados, temas, alguns projetos de lei e votações:

Estes cidadãos criados têm números de cartão de cidadão associados, os quais foram usados para efetuar todos os testes e funcionalidades tanto na aplicação web como aplicação desktop. Estes números de cartão de cidadão deve ser inserido nos campos de login adequados para entrar nas aplicações e conseguir usar as funcionalidades.

Para efetuar login, podem ser usados os números de 1 a 12, inclusive, sendo os cidadãos com número 1,6 e 11 delegados. Apenas estes podem apresentar projetos de lei e ser escolhidos para temas. Os temas criados são com as seguintes descrições: Educação, Economia, Desporto, Turismo. O delegado 1 tem dois projetos de lei apresentados, e os outros dois delegados apresentaram um projeto de lei cada. As votações criadas são dos projetos de lei apresentados pelos delegados com número de CC 1 e 6.