As anotações @Id e @GeneratedValue são usadas no trabalho servem para definir a chave primária de uma entidade e permitir adicionar objetos na base de dados mais facilmente, sem ter de fazer mudanças a nível da política de negócios do sistema. No nosso caso foi escolhido o GenerationType.SEQUENCE, o que significa que um valor sequencial é gerado automaticamente pela base de dados assim que um objeto é adicionado.

Usamos @Inheritance que define a estratégia de herança usada na classe. No nosso caso, a estratégia usada foi a TABLE_PER_CLASS, que significa que cada subclasse da classe Cidadao, isto é, a classe Delegado, tem sua própria tabela na base de dados.

Usamos @MapKeyColumn, que indica que o campo assinalado da classe será, neste caso o id, será usado como a chave do mapa que se encontra na mesma classe.

Usamos @Temporal, que indica que o atributo é uma data e hora e que o tipo de persistência deve ser temporal. Neste caso, o tipo de persistência usado foi TemporalType.TIMESTAMP, o que significa que a data representada deve ter data e hora.

O @JoinColumn usado na classe Tema serve para especificar a coluna usada para a junção numa entidade associação. Neste caso em concreto é usada para especificar a coluna usada para a junção na associação many-to-one com o TemaPai. O atributo name é usado para especificar o nome da coluna que armazena a chave estrangeira que faz a ligação entre as duas tabelas, neste caso a tabela temas. Esta anotação é muito importante visto que permite à ORM mapear a associação entre as entidades na base dados, evitando assim ORM mismatch.

Usamos @ManyToMany, que indica que a relação entre a classe Cidadao e Delegado é muitos-para-muitos. Ou seja, um cidadão pode ter vários delegados (um por tema) e cada delegado pode ter vários cidadãos que o elegeram.

Usamos a anotação @ManyToOne, que indica que a relação entre a classe duas classes é muitos-para-um. Por exemplo na classe ProjetoLei, vários projetos de lei podem ter o mesmo tema, e um tema pode estar associado a vários projetos de lei.

Em contrapartida ao @ManyToOne, usamos o @OneToMany que indica uma relação um-para-muitos. Uma vez que foi usado @ManyToOne, por exemplo na classe ProjetoLei para referenciar o tema, implica que na classe Tema devemos usar a anotação @OneToMany para referenciar vários projetos de lei associados a um tema.

Usamos @ElementCollection que indica que a collection em questão deve ser tratada como uma entidade incorporada, ou seja, uma entidade que não tem uma identidade própria, mas é dependente da entidade proprietária. Por exemplo o atributo apoios depende do ProjetoLei (Isto no ficheiro ProjetoLei.java).

Juntamente com as últimas 4 anotações usamos o fetch, de tipo EAGER (fetch = FETCHTYPE.EAGER) que nos indica que as entidades relacionadas são carregadas

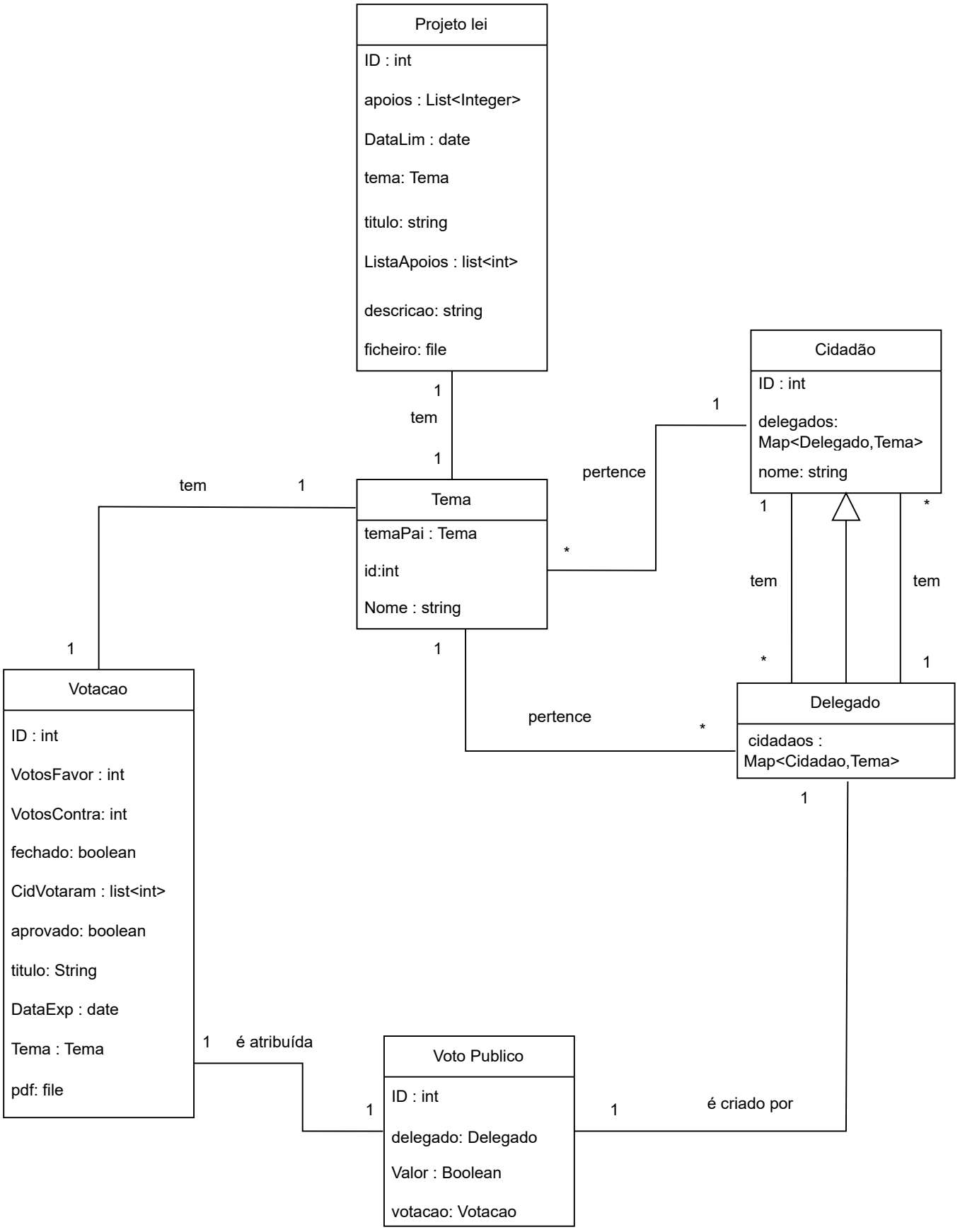
imediatamente junto com a entidade principal. Este tipo de fetch é muito útil em casos em que as entidades relacionadas são usadas com frequência, que acontece no nosso sistema visto que fazemos inúmeros SQL Statements que são impactados por estas relações.

Nos Handlers usamos anotações como o @Service que indica que a classe é um componente da camada serviço gerenciado pelo Spring. Esta anotação é útil para separar a lógica de negócios de outras partes da aplicação e para permitir a injeção de dependência em outras classes. Seguindo o tópico de injeção, foi usado a anotação @Autowired para injetar as dependências necessárias em outras classes assinaladas por esta anotação. Por fim num dos handlers foi usado a anotação @Transactional que garante que o método assinalado, será executado em uma única transação. Isso significa que se houver alguma exceção durante a execução do método, todas as alterações feitas à base de dados serão revertidas para garantir a consistência dos dados. Além disso, esta anotação permite que o método seja executado dentro do contexto da transação gerenciada pelo Spring.

Nos Catalogos e nos Listeners foi usado também a anotação Autowired, para além da anotação @Component assinalando que a classe é gerida pelo Spring, o que significa que o Spring cria uma instância dessa classe e a coloca no contexto de aplicação, permitindo que outras classes possam utilizá-la.

Em alguns listeners foi usada a anotação @Scheduled que serve para configurar um método para ser executado periodicamente em intervalos específicos de tempo. No caso, do método checkDate() da classe DataLimiteApoioListener, é executado a cada 3 segundos, conforme especificado na anotação. Isso permite que a classe monitore constantemente a data limite de apoio dos projetos de lei e execute ações como remover projetos de lei que a sua data limite já foi ultrapassada.

Por último foi usada a anotação @Repository que identifica que a classe é um repositório, isto quer dizer que esta classe fornece acesso aos dados armazenados na base de dados, com ajuda das consultas assinaladas pela anotação @Query.


Projeto lei
ID : int
apoios : List<Integer>
DataLim : date
tema: Tema
titulo: string
ListaApoios : list<int>
descricao: string
ficheiro: file
1
tem
1
Cidadão
ID : int
delegados: Map<Delegado,Tema>
nome: string
1
pertence
*
tem
1
Tema
temaPai : Tema
id:int
Nome : string
1
1
*
1
tem
*
tem
1
Votacao
ID : int
VotosFavor : int
VotosContra: int
fechado: boolean
CidVotaram : list<int>
aprovado: boolean
titulo: String
DataExp : date
Tema : Tema
pdf: file
pertence
*
Delegado
cidadaos : Map<Cidadao,Tema>
1
1
é atribuída
Voto Publico
ID : int
delegado: Delegado
Valor : Boolean
votacao: Votacao
1
1
é criado por

Entidades

Repositórios

Catálogos

Handlers

Listeners

FileHandler

+ fileToBinaryFile(String, String): File
+ deleteFile(File): void
+ BinaryFileToPdf(File): void

Democracia

- log: Logger

+ main(String[]): void

ProjetoLei

- tema: Tema
+ descricao: String
- titulo: String
- dataLim: Date
- id: Integer
- apoios: List<Integer>
+ ficheiro: File

+ getApoios(): List<Integer>
+ getTitulo(): String
+ equals(Object): boolean
+ getDescricao(): String
+ getFicheiro(): File
+ getDataLim(): Date
+ getTema(): Tema
+ addApoios(Integer): void
+ getId(): Integer

Votacao

- cidVotaram: List<Integer>
- tema: Tema
- fechado: Boolean
- titulo: String
- votosFavor: Integer
- votosContra: Integer
- aprovado: Boolean
- id: Integer
- dataExp: Date
- pdf: File

+ getId(): Integer
+ getTema(): Tema
+ getVotosFavor(): Integer
+ hashCode(): int
+ isClosed(): boolean
+ cidadaoJaVotou(int): boolean
+ equals(Object): boolean
+ addVoto(int, boolean): void
+ getPdf(): File
+ setDataExp(Date): void
+ getTitulo(): String
+ getDataExp(): Date
+ fechaVotacao(): void
+ aprovaVotacao(): void
+ getCidVotaram(): List<Integer>
+ getVotosContra(): Integer
+ isApproved(): boolean

Delegado

- cidadaos: Map<Cidadao, Tema>

+ addCidadao(Cidadao, Tema): void
+ getCidadaos(): Map<Cidadao, Tema>

Cidadao

- id: Integer
- delegados: Map<Delegado, Tema>
- nome: String

+ getDelegados(): Map<Delegado, Tema>
+ getId(): Integer
+ getNome(): String
+ addDelegado(Delegado, Tema): void

Tema

- nome: String
- id: Integer
- temaPai: Tema

+ getNome(): String
+ setTemaPai(Tema): void
+ hashCode(): int
+ getTemaPai(): Tema
+ getId(): Integer
+ equals(Object): boolean

VotoPublico

- votacao: Votacao
- id: Integer
- valor: boolean
- delegado: Delegado

+ getDelegado(): Delegado
+ getId(): Integer
+ getValor(): boolean
+ getVotacao(): Votacao

<<interface>>
VotacaoRepository

+ allVotacoes(): List<Votacao>
+ getVotacaoById(Integer): Votacao
+ getVotacaoByName(String): Votacao
+ existsByTitle(String): boolean
+ getVotacoesExpiradas(): List<Votacao>

<<interface>>
ProjetoLeiRepository

+ getByTema(Tema): List<ProjetoLei>
+ getProjetoLeiById(Integer): ProjetoLei
+ allProjetoLei(): List<ProjetoLei>
+ getByNomeTema(String): List<ProjetoLei>
+ getProjetosLeiComAprovacao(): List<ProjetoLei>
+ getProjetosLeiExpirados(): List<ProjetoLei>
+ getByTitle(String): ProjetoLei

<<interface>>
VotoPublicoRepository

+ getVotosPublicoByTitulo(String): List<VotoPublico>
+ getVotoPublicoByDelegado_Id(Integer): List<VotoPublico>
+ allVotosPublicos(): List<VotoPublico>
+ getVotoPublicoByDelegadoAndVotId(Integer, Integer): VotoPublico

<<interface>>
DelegadoRepository

+ findByName(String): List<Delegado>
+ allDelegados(): List<Delegado>
+ getDelegadoById(Integer): Delegado
+ getDelegadoByName(String): Delegado

<<interface>>
CidadaoRepository

+ getCidadaoById(Integer): Cidadao
+ allCidadaos(): List<Cidadao>
+ findByName(String): Cidadao

<<interface>>
TemaRepository

+ getByNome(String): Tema
+ existsByTitle(String): boolean
+ allTemas(): List<Tema>
+ getTemaById(Integer): Tema

VotacaoCatalog

- catalog: VotacaoCatalog
- votacaoRepository: VotacaoRepository

+ addVotacao(ProjetoLei): Integer
+ existsVotacao(String): boolean
+ setDateNow(Integer): void
+ getVotacoesExpiradas(): List<Votacao>
+ getVotacaoById(Integer): Votacao
+ getCatalogo(): VotacaoCatalog
+ updateVoto(Votacao): void
+ getVotacoes(): List<Votacao>
+ getVotacaoByTitle(String): Votacao
+ aprovaVotacao(Integer): void
+ deleteVotacaoById(Integer): void
+ fechaVotacao(Integer): void

ProjetoLeiCatalog

- projetoLeiRepository: ProjetoLeiRepository
- catalog: ProjetoLeiCatalog

+ getProjetosLei(): List<ProjetoLei>
+ getProjetoLeiById(int): ProjetoLei
+ getProjetoLeiByName(String): ProjetoLei
+ getProjetosLeiExpirados(): List<ProjetoLei>
+ getProjetosLeiByTema(String): List<ProjetoLei>
+ getCatalogo(): ProjetoLeiCatalog
+ existsProjetoLei(String): boolean
+ addProjetoLei(Integer, Tema, String, String, String, Date): Integer
+ getProjetosLeiQueObtiveramAprovacao(): List<ProjetoLei>
+ deleteProjetoLeiById(Integer): void

VotoPublicoCatalog

- votoPublicoRepository: VotoPublicoRepository
- catalog: VotoPublicoCatalog

+ getVotosPublicosDeDelegado(Integer): List<VotoPublico>
+ getVotosPublicosByTitle(String): List<VotoPublico>
+ getVotoPublicoByDelegadoAndVotId(Integer, Integer): VotoPublico
+ deleteVotoPublicoById(Integer): void
+ getCatalogo(): VotoPublicoCatalog
+ deleteVotoPublicoByDelegadoAndVotId(Integer, Integer): void
+ addVotoPublico(Delegado, Votacao, boolean): void

DelegadoCatalog

- delegadoRepository: DelegadoRepository
- catalog: DelegadoCatalog

+ getDelegadoByName(String): Delegado
+ addDelegado(String): int
+ updateDelegado(Delegado): void
+ resetDelegadoForDeletion(Integer): void
+ getDelegados(): List<Delegado>
+ getCatalogo(): DelegadoCatalog
+ getDelegadoById(Integer): Delegado
+ deleteDelegadoById(Integer): void
+ existsDelegado(Integer): boolean

CidadaoCatalog

- cidadaoRepository: CidadaoRepository
- catalog: CidadaoCatalog

+ addCidadao(String): Integer
+ resetCidadaoForDeletion(Integer): void
+ getCidadaos(): List<Cidadao>
+ deleteCidadaoById(Integer): void
+ getCidadaoById(Integer): Cidadao
+ existsCidadao(Integer): boolean
+ getCatalogo(): CidadaoCatalog
+ getCidadaoByName(String): Cidadao
+ updateCidadao(Cidadao): void

TemaCatalog

- temaRepository: TemaRepository
- catalog: TemaCatalog

+ getCatalogo(): TemaCatalog
+ getTemas(): List<Tema>
+ getTemaByName(String): Tema
+ existsTema(String): boolean
+ initTema(): void

FecharProjetoLeiHandler

- projetoLeiCatalog: ProjetoLeiCatalog
- votacaoCatalog: VotacaoCatalog

+ createVotacaoFromProjetoLei(Integer): void
+ allProjetosLei(): List<ProjetoLei>

VotarPropostaHandler

- votacaoCatalog: VotacaoCatalog
- delegadoCatalog: DelegadoCatalog
- votoPublicoCatalog: VotoPublicoCatalog
- cidadaoCatalog: CidadaoCatalog

+ votarProposta(Integer, Integer, boolean): void
- RealizarVoto(Cidadao, Integer, boolean): void
+ AddVoto(Integer, Integer, boolean): void

FecharVotacaoHandler

- votoPublicoCatalog: VotoPublicoCatalog
- votarPropostaHandler: VotarPropostaHandler
- votacaoCatalog: VotacaoCatalog
- cidadaoCatalog: CidadaoCatalog

+ fecharVotacao(Integer): void

ApresentarProjetoLeiHandler

- delegadoCatalog: DelegadoCatalog
- projetoLeiCatalog: ProjetoLeiCatalog
- temaCatalog: TemaCatalog

+ proporProjetoLei(Integer, String, String, String, String, Long): int

EscolherDelegadoHandler

- temaCatalog: TemaCatalog
- delegadoCatalog: DelegadoCatalog
- cidadaoCatalog: CidadaoCatalog

+ EscolherDelegado(Integer, Integer, String): void

LimiteDezMilListener

- fecharProjetoLeiHandler: FecharProjetoLeiHandler
- projetoLeiCatalog: ProjetoLeiCatalog

+ checkLimite(): void

DataLimiteApoioListener

- fecharProjetoLeiHandler: FecharProjetoLeiHandler
- projetoLeiCatalog: ProjetoLeiCatalog

+ checkDate(): void

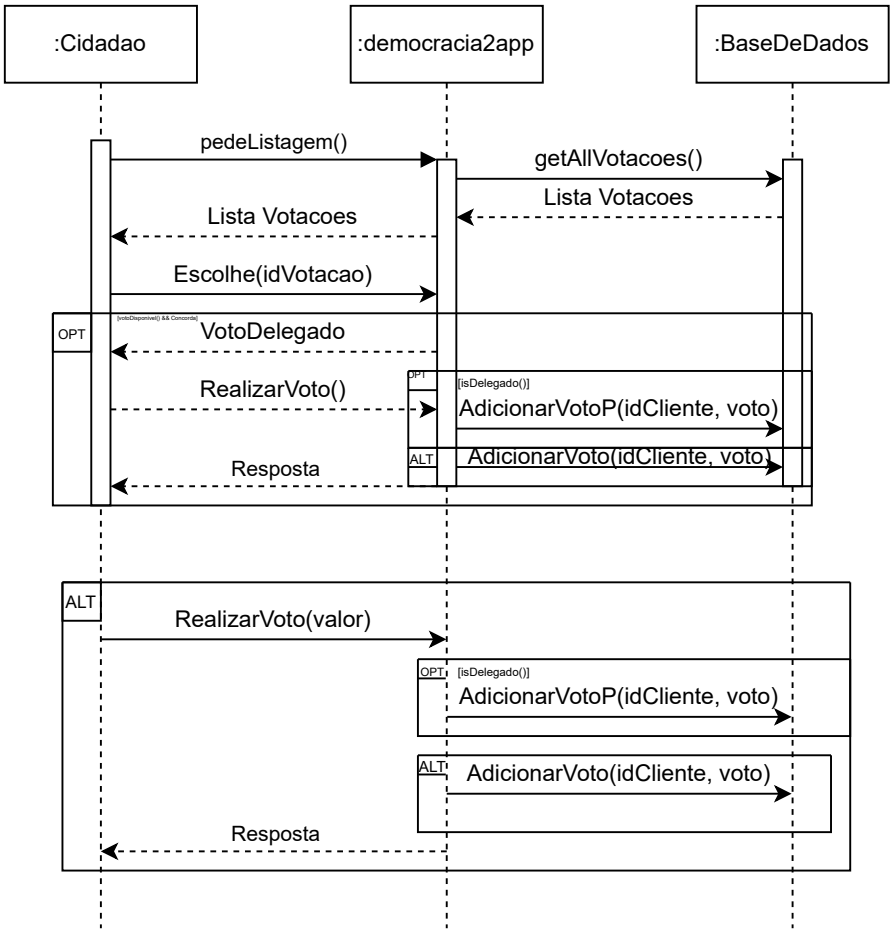
DataLimiteVotacaoListener

- fecharVotacaoHandler: FecharVotacaoHandler
- votacaoCatalog: VotacaoCatalog

+ checkDate(): void

Optámos por esta versão no SSD porque num cenário de front end aparecia o voto publico do delegado, onde o cidadão iria escolher uma opção de concordar ou não, e em termos de haver um voto disponível e o cidadão concordar com esse voto era apenas um “realizarVoto” sem input uma vez que o valor seria o mesmo do delegado.

Caso não concordasse iria realizar um voto com um valor onde iria entrar no mesmo cenário que se fosse um delegado seria um voto público, se não um privado. Estas verificações não são feitas em código, uma vez que isto seria num cenário de front-end. No entanto foi deixado como comentário no handler de votar numa proposta uma possível verificação para esse caso.


:Cidadao
:democracia2app
:BaseDeDados
pedeListagem()
getAllVotacoes()
Lista Votacoes
Lista Votacoes
Escolhe(idVotacao)
OPT
[votoDisponivel() && Concorda]
VotoDelegado
RealizarVoto()
OPT
[isDelegado()]
AdicionarVotoP(idCliente, voto)
ALT
AdicionarVoto(idCliente, voto)
Resposta
ALT
RealizarVoto(valor)
OPT
[isDelegado()]
AdicionarVotoP(idCliente, voto)
ALT
AdicionarVoto(idCliente, voto)
Resposta