Construção de Sistemas de Software

Relatório do 2º Trabalho

Tomás Barreto 56282
João Matos 56292
Diogo Pereira 56302

Workflow

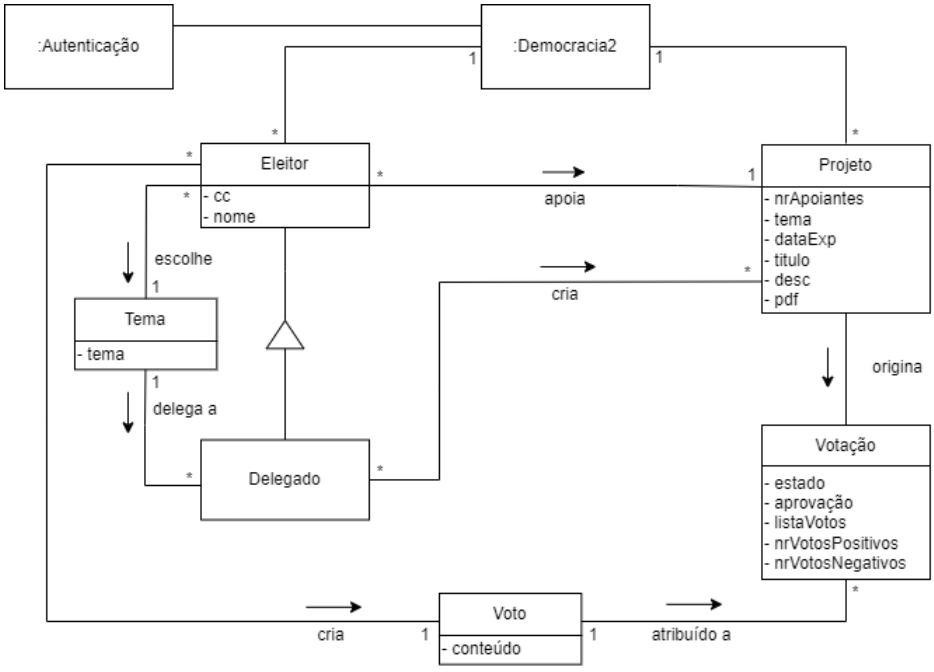
Ao longo do desenvolvimento do projeto não foram criados *branches*, dado que tomamos a decisão de desenvolver todo o código em conjunto e que os *commits* apenas seriam efetuados por um dos membros, servindo-se do campo $Co-authored-by$.

Popular base de dados

De forma a popular a base de dados, basta abrir a web app e a criação de uma conta deve ser efetuada na mesma aplicação.

Artefactos de Desenvolvimento

Modelo de Domínio



SSD do caso de uso J

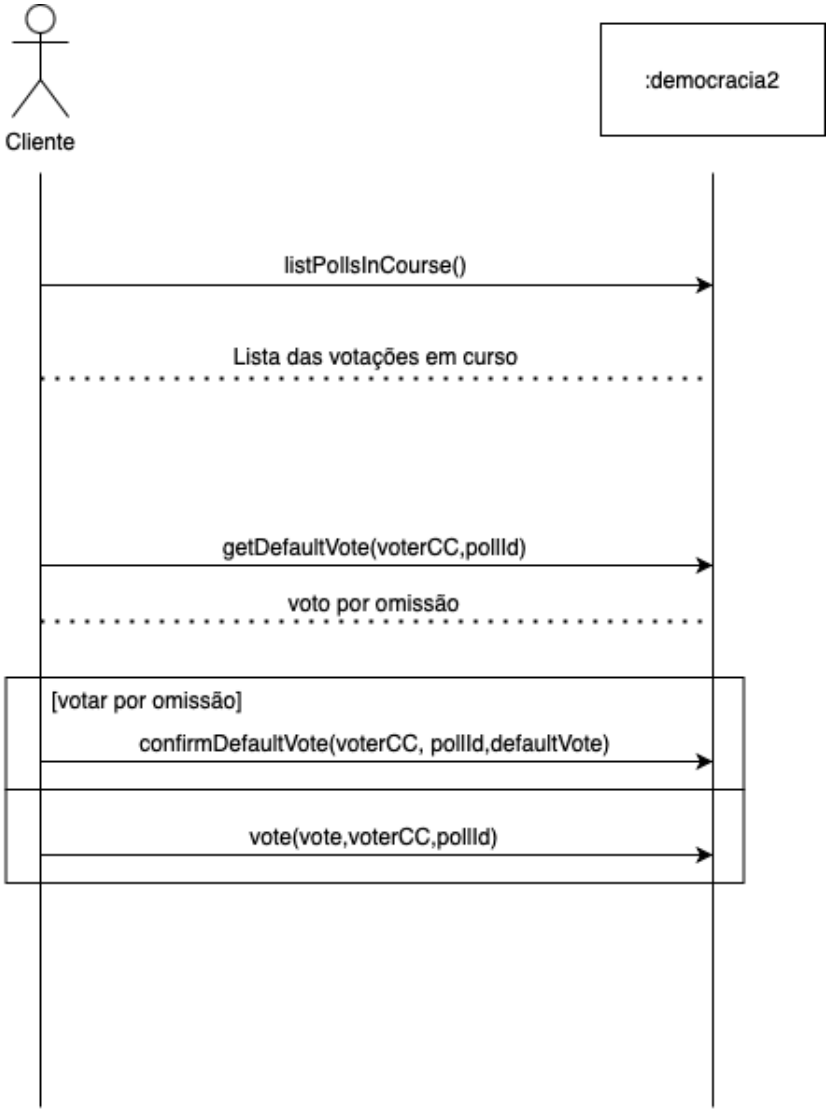
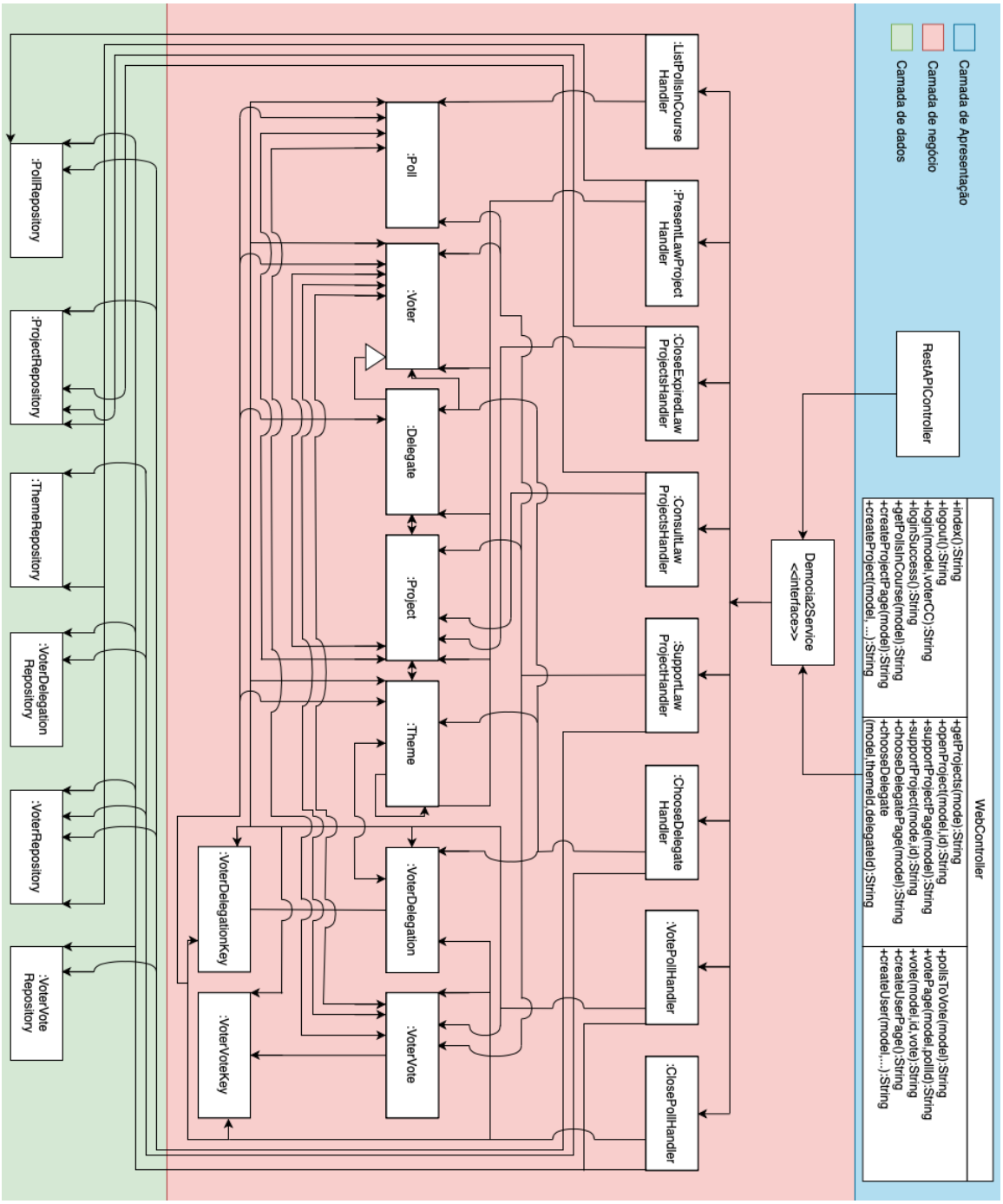


Diagrama de Classes



Atributos e Métodos do Diagrama de Classes

Entidades

Voter

```java
@Entity
@Inheritance(strategy = InheritanceType.SINGLE_TABLE)
public class Voter {

  @Id @GeneratedValue(strategy = GenerationType.SEQUENCE)
  private long id;

  5 usages
  @Column(unique = true) @Nonnull
  private String cc;

  5 usages
  private String name;

  3 usages
  @ManyToMany(fetch = FetchType.EAGER)
  @JoinTable(
    name = "voter_supports_project",
    joinColumns = @JoinColumn(name = "voter_id"),
    inverseJoinColumns = @JoinColumn(name = "project_id")
  )
  private List<Project> supportedProjects;

  2 usages
  @OneToMany(mappedBy = "voter")
  private Set<VoterVote> votes;

  2 usages
  @OneToMany(mappedBy = "voter")
  private Set<VoterDelegation> voterDelegation;
```

Na classe $Voter$ utilizamos as anotações @$Entity$ por se tratar de uma entidade da aplicação e @$Inheritance$ com a estratégia $SINGLE\_TABLE$, com vista a criar apenas uma tabela para a herança na base de dados, juntamente com classe $Delegate$.

Não se trata de uma tabela muito esparsa, uma vez que a classe não possui muitos atributos e é apenas estendida pela classe $Delegate$. O atributo Id possui a anotação @Id, uma vez que se trata da *primary key* da tabela e será gerada pela base de dados através de @$GeneratedValue$, servindo-se da estratégia $SEQUENCE$.

O atributo cc está anotado com @$Column(unique = true)$ para tornar este atributo único na base de dados e @$Nonnull$ para indicar a obrigatoriedade de preenchimento da coluna correspondente.

O atributo $supportedProjects$ possui @$ManyToMany(fetch = FetchType.EAGER)$,

uma vez que um $Voter$ pode suportar vários projetos e um projeto pode ser suportado por diversos $Voter$ e utilizado $FetchType.EAGER$ para, ao ser obtido um $Voter$, carregar instantaneamente todos os $Project$ suportados, e $JoinTable$, para criar uma tabela que mapeia a ligação entre $Voter$ e os projetos correspondentes.

O atributo *votes* é anotado com $@OneToMany(mappedBy = "voter")$, uma vez que um $Voter$ pode realizar vários votos para $Poll$ distintos, mapeando o atributo *voter* na classe $VoterVote$.

O atributo *voterDelegation* é também anotado com $@OneToMany(mappedBy = "voter")$, visto que um $Voter$ pode escolher vários $Delegate$ para diferentes temas, mapeando o atributo *voter* na classe $VoterDelegation$. Não foi usado $@ManyToMany$ pois a relação entre $Voter$ e $Delegate$ depende de um $Theme$, sendo assim, foi criada uma entidade para os associar ($VoterDelegation$).

Delegate

```java
@Entity
public class Delegate extends Voter {

    2 usages
    @OneToMany(mappedBy = "delegate")
    private Set<VoterChoosesDelegate> voterChoosesDelegate;

    3 usages
    @OneToMany(mappedBy = "delegate")
    private List<Project> ownedProjects;
```

Na classe $Delegate$ utilizamos a anotação $@Entity$ por se tratar de uma entidade da aplicação, que estende a classe $Voter$.

No atributo $voterDelegation$ usamos a anotação $@OneToMany(mappedBy = "delegate")$ visto que um $Delegate$ pode ser escolhido por vários $Voter$ para diferentes temas, mapeando o atributo *delegate* na classe $VoterDelegation$. Não foi usado $@ManyToMany$ pois a relação entre $Delegate$ e $Voter$ depende de um $Theme$, sendo assim, foi criada uma entidade para os associar ($VoterDelegation$).

O atributo $ownedProjects$ possui a anotação $@OneToMany(mappedBy = "delegate")$ uma vez que um $Delegate$ pode criar vários $Project$, sendo mapeado pelo atributo *delegate* da classe $Project$.

Project

```java
@Entity
public class Project {

  @Id
  @GeneratedValue(strategy = GenerationType.SEQUENCE)
  private long id;

  3 usages
  @ManyToOne
  @JoinColumn(name = "theme_id", nullable = false)
  private Theme theme;

  3 usages
  @Lob private byte[] pdf;

  3 usages
  @Column(name = "expiration_date", columnDefinition = "TIMESTAMP")
  private LocalDateTime expirationDate;

  5 usages
  @ManyToMany(mappedBy = "supportedProjects", fetch = FetchType.EAGER)
  private List<Voter> supporters;

  no usages
  @OneToOne(mappedBy = "project")
  private Poll poll;

  3 usages
  @ManyToOne
  @JoinColumn(name = "delegate_id", nullable = false)
  private Delegate delegate;
```

Na classe *Project* utilizamos a anotação @*Entity* por se tratar de uma entidade da aplicação.

O atributo *Id* possui a anotação @*Id*, uma vez que se trata da *primary key* da tabela e será gerada pela base de dados através de @*GeneratedValue*, servindo-se da estratégia *SEQUENCE*.

O atributo *theme* possui as anotações @*ManyToOne*, uma vez que vários *Project* podem servir-se do mesmo *Theme* e @$JoinColumn(name = "theme_id", nullable = false)$, como forma de criar uma coluna com o nome *theme_id* na tabela da entidade, sendo que este valor deve ser preenchido obrigatoriamente.

O atributo *pdf* tem a anotação @*Lob* para guardar dados binários na base de dados.

O atributo *expirationDate* é acompanhado da anotação @$Column(name = "expiration_date", columnDefinition = "TIMESTAMP")$, atribuindo o nome *expiration_date* e o tipo de dados correto à coluna correspondente.

O atributo *supporters* foi anotado com @$ManyToMany(mappedBy = "supportedProjects", fetch = FetchType.EAGER)$, uma vez que um *Project* é suportado por vários *Voter*, mapeando a atributo *supportedProjects* na classe *Voter* e utilizado *FetchType.EAGER*

para, ao ser obtido um *Project*, carregar instantaneamente todos os seus supporters.

O atributo *Poll* é anotado com @*OneToOne*(*mappedBy* = "*project*"), pois um *Project* corresponde a um *Poll*, em caso de aprovação, mapeando o atributo *project* na classe *Poll*.

O atributo *Delegate* tem as anotações @*ManyToOne*, porque um *Project* apenas pode ser criado por um *Delegate* e @*JoinColumn*(*name* = "*delegate*$_i$*d*", *nullable* = *false*), de forma a criar uma coluna com o nome *delegate_id*, de preenchimento obrigatório.

Poll

```java
@Entity
public class Poll {

  @Id
  @GeneratedValue(strategy = GenerationType.SEQUENCE)
  private long id;

  5 usages
  private boolean inCourse;

  4 usages
  private boolean isApproved;

  6 usages
  @OneToOne
  @JoinColumn(name = "project_id", referencedColumnName = "id")
  private Project project;

  2 usages
  @OneToMany(mappedBy = "poll", fetch = FetchType.EAGER)
  private Set<VoterVotesPoll> votes;

  5 usages
  private int positiveVotes;

  3 usages
  private int negativeVotes;

  4 usages
  @Column(name = "expiration_date", columnDefinition = "TIMESTAMP")
  private LocalDateTime expirationDate;
```

Na classe *Poll* utilizamos a anotação @*Entity* por se tratar de uma entidade da aplicação.

O atributo *id* possui a anotação @*Id*, uma vez que se trata da *primary key* da tabela e será gerada pela base de dados através de @*GeneratedValue*, servindo-se da estratégia *SEQUENCE*.

O atributo *project* possui as anotações @*OneToOne*, pois uma *Poll* apenas pode estar associada a um *Project* e @*JoinColumn*(*name* = "*project*$_i$*d*", *referencedColumnName* = "*id*"), com vista a criar uma coluna com o nome *project_id*, referenciando o id

do *Project*.

O atributo *votes* é anotado com @$OneToMany(mappedBy = "poll", fetch = FetchType.EAGER)$, uma vez que uma *Poll* tem vários $VoterVote$ associados, mapeando o atributo *poll* da classe $VoterVote$ com $FetchType.EAGER$, de forma a carregar todos os objetos instantaneamente.

O atributo *expirationDate* é acompanhado da anotação @$Column(name = "expiration_date", columnDefinition = "TIMESTAMP")$, atribuindo o nome *expiration_date* e o tipo de dados correto à coluna correspondente.

Theme

```java
@Entity
public class Theme {
  @Id
  @GeneratedValue(strategy = GenerationType.SEQUENCE)
  private long Id;

  3 usages
  private String theme;

  4 usages
  @ManyToOne(fetch = FetchType.LAZY)
  private Theme parent;

  2 usages
  @OneToMany(mappedBy = "parent")
  private List<Theme> children;

  2 usages
  @OneToMany(mappedBy = "theme")
  private List<VoterChoosesDelegate> voterChoosesDelegates;

  no usages
  @OneToMany(mappedBy = "theme")
  private List<Project> projects;
```

Na classe *Theme* utilizamos a anotação @*Entity* por se tratar de uma entidade da aplicação.

O atributo *id* possui a anotação @Id, uma vez que se trata da *primary key* da tabela e será gerada pela base de dados através de @$GeneratedValue$, servindo-se da estratégia $SEQUENCE$.

O atributo *parent* corresponde ao tema pai, sendo anotado com @$ManyToOne(fetch = FetchType.LAZY)$, uma vez que um *Theme* apenas tem um tema pai, utilizando $FetchType.LAZY$, de forma a carregar objetos apenas quando necessário, uma vez que apenas queremos subir na hierarquia quando o $Voter$ não tem um *Delegate* atribuído para este *Theme*.

O atributo *children* tem a anotação @$OneToMany(mappedBy = "parent")$, uma vez que um *Theme* pode ter vários temas filhos (subtemas), mapeando o atributo *parent* desta mesma classe.

O atributo $voterDelegations$ é anotado com $@OneToMany(mappedBy = "theme")$ pois pode ter vários $Delegate$ associados a $Voter$ por um mesmo $Theme$. Não foi usado $@ManyToMany$ pois a relação entre $Voter$ e $Delegate$ depende de um $Theme$, sendo assim, foi criada uma entidade para os associar ($VoterDelegation$).

O atributo $projects$ tem a anotação $@OneToMany(mappedBy = "theme")$ de forma a que um tema consiga ter vários $Project$ associados, mapeando o atributo $theme$ em $Project$.

VoterDelegation

```java
@Entity
public class VoterChoosesDelegate {

    3 usages
    @EmbeddedId private VoterChoosesDelegateKey id;

    3 usages
    @ManyToOne
    @MapsId("themeId")
    @JoinColumn(name = "theme_id", nullable = false)
    private Theme theme;

    3 usages
    @ManyToOne
    @MapsId("voterId")
    @JoinColumn(name = "voter_id", nullable = false)
    private Voter voter;

    3 usages
    @ManyToOne
    @JoinColumn(name = "delegate_id", nullable = false)
    private Delegate delegate;
```

Na classe $VoterDelegation$ utilizamos a anotação $@Entity$ por se tratar de uma entidade da aplicação.

O atributo id do tipo $VoterDelegationKey$ é anotado com $@EmbeddedId$, uma vez que corresponde a uma $primary\ key$ composta.

O atributo $theme$ tem as anotações $@ManyToOne$, uma vez que um $Theme$ pode associar diferentes $Delegate$ a diferentes $Voter$, $@MapsId("themeId")$, mapeando a $primary\ key$ com o id do $Theme$ e $@JoinColumn(name = "theme\_id", nullable = false)$, criando uma coluna $theme\_id$, de preenchimento obrigatório.

O atributo $voter$ tem as anotações $@ManyToOne$, uma vez que um $Voter$ pode estar associado a diferentes $Delegate$ e diferentes $Theme$, $@MapsId("voterId")$, mapeando a $primary\ key$ com o id do $Voter$ e $@JoinColumn(name = "voter\_id", nullable = false)$, criando uma coluna $voter\_id$, de preenchimento obrigatório.

O atributo $delegate$ tem as anotações $@ManyToOne$, uma vez que um $Delegate$ pode estar associado a diferentes $Voter$ e $@JoinColumn(name = "delegate\_id", nullable = false)$, criando uma coluna $delegate\_id$, de preenchimento obrigatório.

VoterDelegationKey

```java
@Embeddable
public class VoterChoosesDelegateKey implements Serializable

  1 usage
  private long voterId;

  1 usage
  private long themeId;
```

Na classe $VoterDelegationKey$ é usado $@Embeddable$, uma vez que corresponde a um atributo composto, mais concretamente, uma *primary key* composta, da classe $VoterDelegation$.

VoterVote

```java
@Entity
public class VoterVotesPoll {

  4 usages
  @EmbeddedId private VoterVotesPollKey id;

  4 usages
  @ManyToOne
  @MapsId("voterId")
  @JoinColumn(name = "voter_id", nullable = false)
  private Voter voter;

  4 usages
  @ManyToOne
  @MapsId("pollId")
  @JoinColumn(name = "poll_id", nullable = false)
  private Poll poll;

  3 usages
  private Boolean isApproved;
```

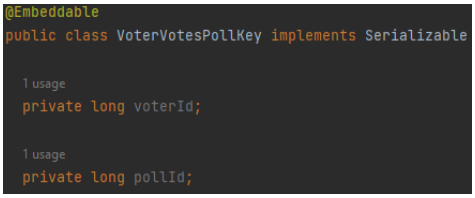
Na classe $VoterVote$ utilizamos a anotação $@Entity$ por se tratar de uma entidade da aplicação.

O atributo id do tipo $VoterVoteKey$ é anotado com $@EmbeddedId$, uma vez que corresponde a uma *primary key* composta.

O atributo $voter$ tem as anotações $@ManyToOne$, uma vez que um $Voter$ pode estar associado a diferentes $Poll$, através do seu voto, $@MapsId("voterId")$, mapeando a *primary key* com o id do $Voter$ e $@JoinColumn(name = "voter\_id", nullable = false)$, criando uma coluna $voter\_id$, de preenchimento obrigatório.

O atributo $poll$ tem as anotações $@ManyToOne$, uma vez que um $Poll$ pode estar associado a diferentes $Voter$, através dos seus votos, $@MapsId("pollId")$, mapeando a *primary key* com o id do $Poll$ e $@JoinColumn(name = "poll\_id", nullable = false)$, criando uma coluna $poll\_id$, de preenchimento obrigatório.

VoterVoteKey



Na classe $VoterVoteKey$ é usado @$Embeddable$, uma vez que corresponde a um atributo composto, mais concretamente, uma *primary key* composta, da classe $VoterVote$.

Testes

```java
@SpringBootTest
@DirtiesContext(classMode = DirtiesContext.ClassMode.BEFORE_EACH_TEST_METHOD)
public class ChooseDelegateHandlerTests {

  @Autowired private VoterChoosesDelegateRepository voterChoosesDelegateRepository;
  @Autowired private VoterRepository voterRepository;
  @Autowired private ThemeRepository themeRepository;

  fc56282
  @Test
  void chooseDelegate() throws VoterNotFoundException, ThemeNotFoundException {
```

```java
@BeforeEach
public void loadNecessaryObjects() {
```

Foi utilizada a anotação @*Autowired* como forma de obter os repositórios através de *dependency injection*.

A anotação @*BeforeEach* para executar a função que popula a base de dados antes de cada função de teste.

Foi ainda utilizada a anotação @$DirtiesContext(classMode = DirtiesContext.ClassMode.BEFORE\_EACH\_TEST\_METHOD)$ para obter uma nova instância da base de dados antes de cada teste efetuado.

Arquitetura de Componentes

Scheduled

```
@Configuration
@EnableScheduling
public class SpringConfig {
}
```

```
@Scheduled(fixedRate = 10000) //10 secs
```

Primeiramente foi criado um ficheiro *spring config* que possui @*Configuration* para ser considerado uma configuração. Após isto, anotamos @*EnableScheduling* para permitir que a aplicação suporte o *Scheduled*.

Depois, nos casos de uso desejados, mais precisamente nas funções que fecham *Project* e *Poll*, usamos @*Scheduled*(*fixedRate* = 10000) para que estas sejam executadas de 10 em 10 segundos.

Web App

Foi criado um controller *WebController* que responde a pedidos realizados na web, comunica com o serviço *Democracia2Service* para obter os dados necessários ao pedido e processa a página com esses dados, enviando-a ao cliente.

API REST

Foi criado um controller *RestApiController* que responde a pedidos http do tipo */api/*, devolvendo em objetos *JSON* os dados requisitados.

Testes API REST

Foi criada uma classe para testar a API REST, onde nos servimos do @$WebMvcTest$ para testar o controlador da API. Criamos também um mock MVC para simular e executar pedidos http e utilizamos @$MockBean$ no $Democracia2Service$. Após isto, testamos todos os pedidos com todas as exceções possíveis de forma a verificar se a resposta da API era correta. Para realizar isto utilizamos o mock service, forçando o $Democracia2Service$ a devolver as exceções e os resultados necessários.

Aplicação Desktop

Foi criada uma aplicação desktop que utiliza $JavaFX$, com uma classe $Main$ que inicia a interface e classes controller que atuam como controladores das diferentes cenas, definidas em ficheiros $fxml$. Existe ainda uma classe $RestApiClientService$ que efetua a comunicação entre a aplicação desktop e o servidor, através da sua REST API.