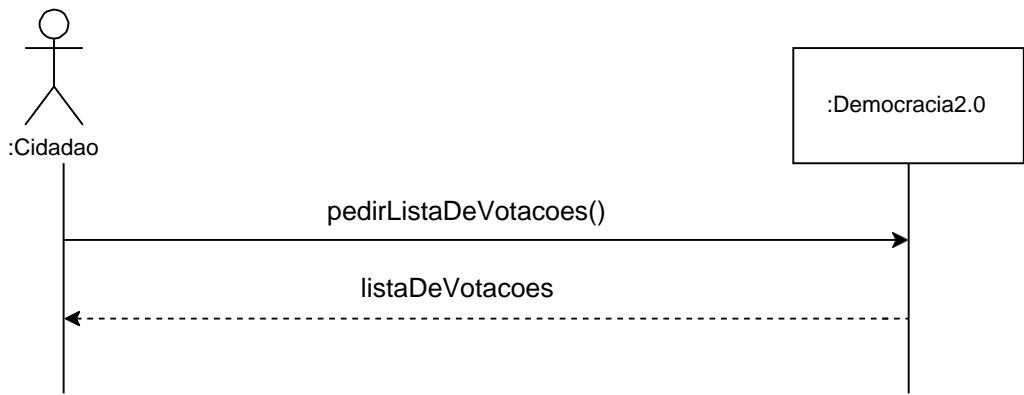
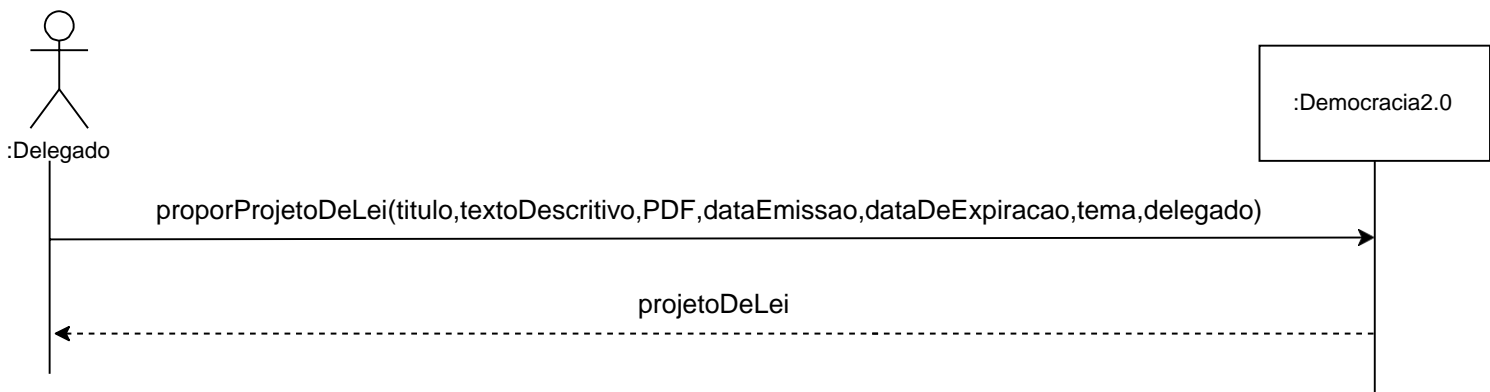


SSD do caso de uso D



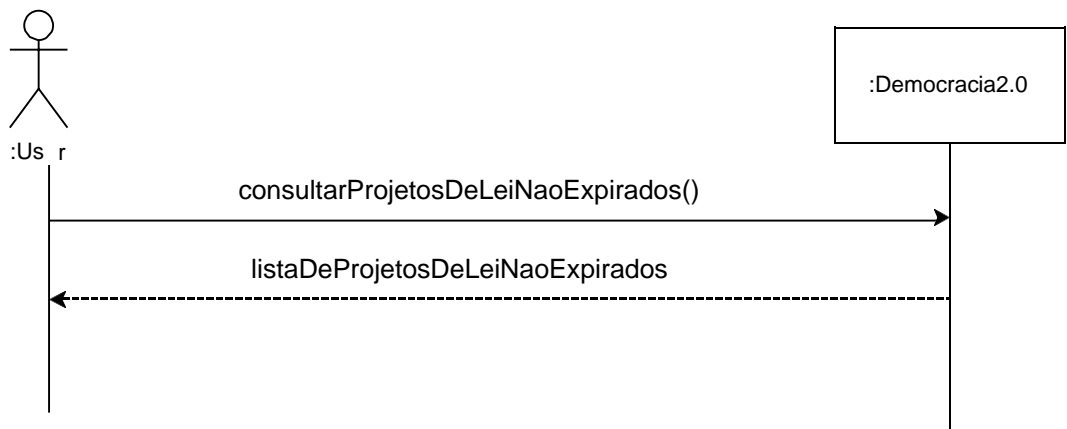
SSD do caso de uso E



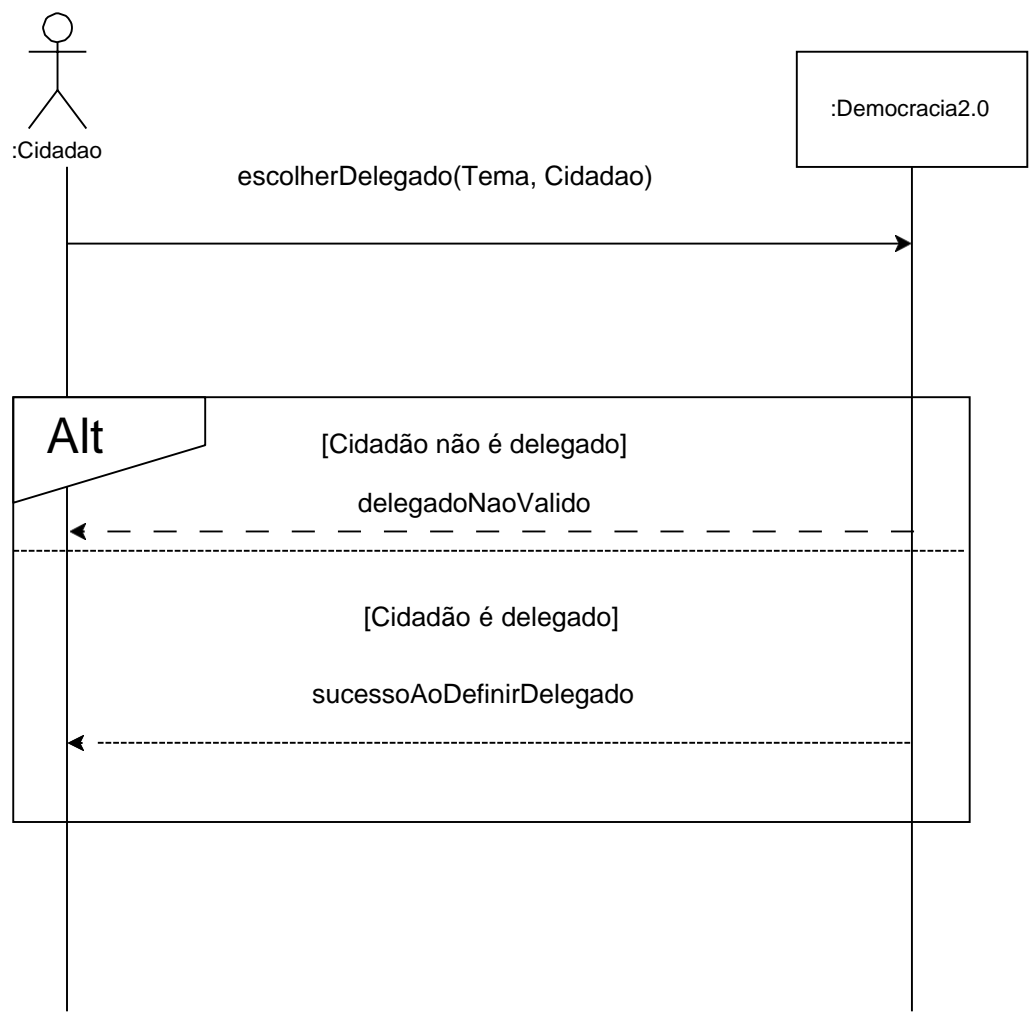
SSD do caso de uso F



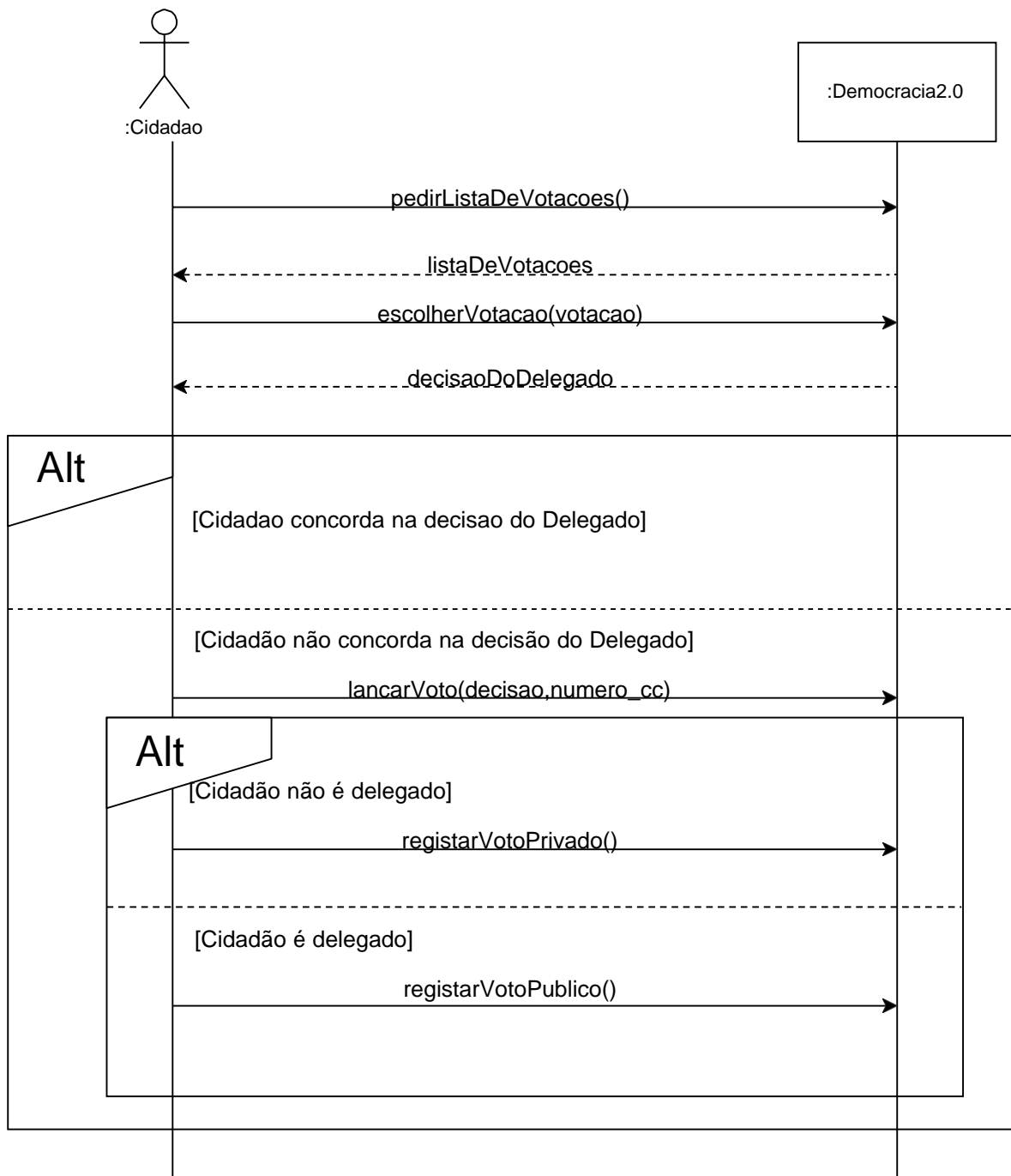
SSD do caso de uso G



SSD do caso de uso I

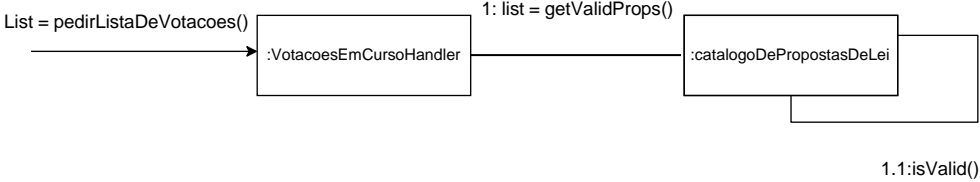


SSD do caso de uso J

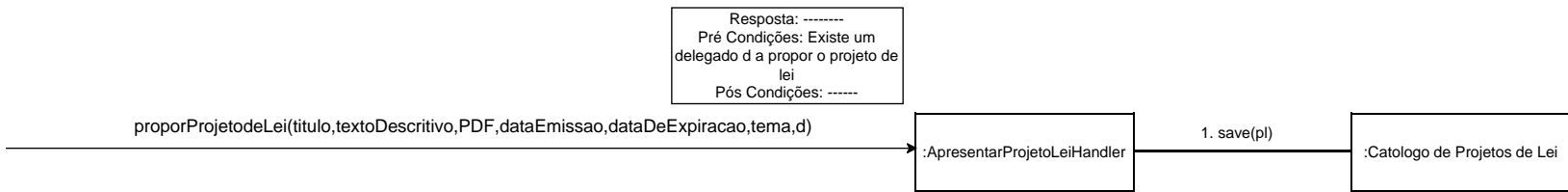


Resposta: List
Pós Condições: -----

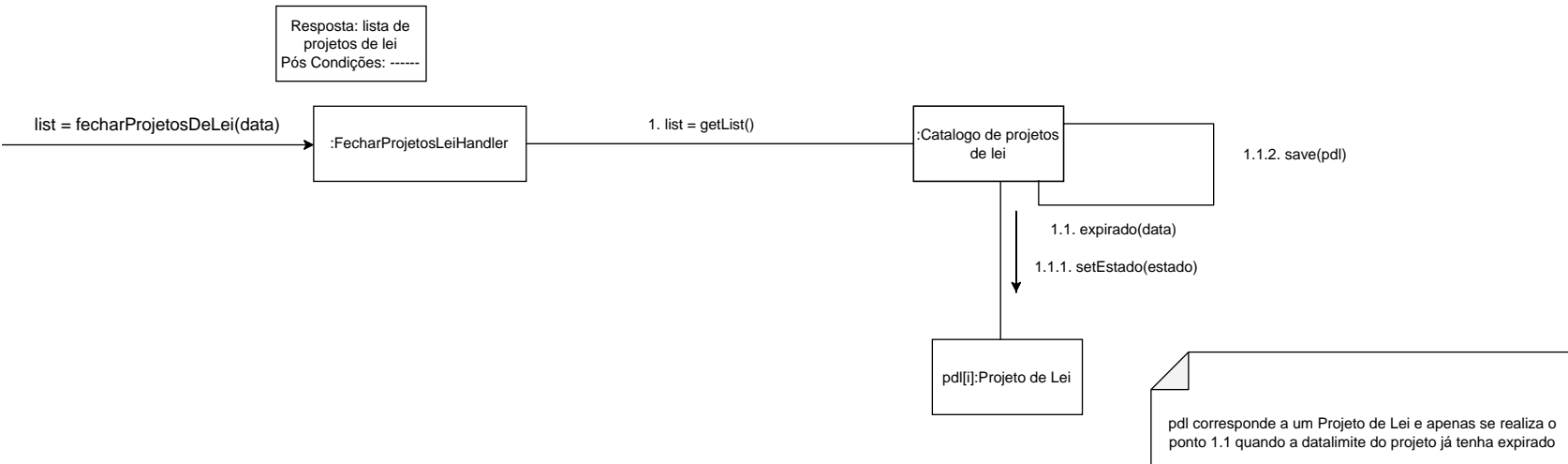
ID do caso de uso D



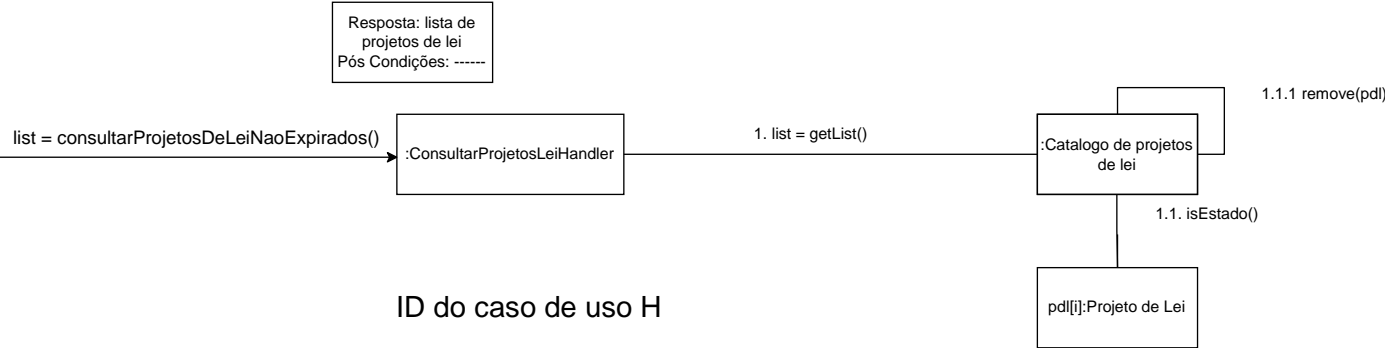
ID do caso de uso E



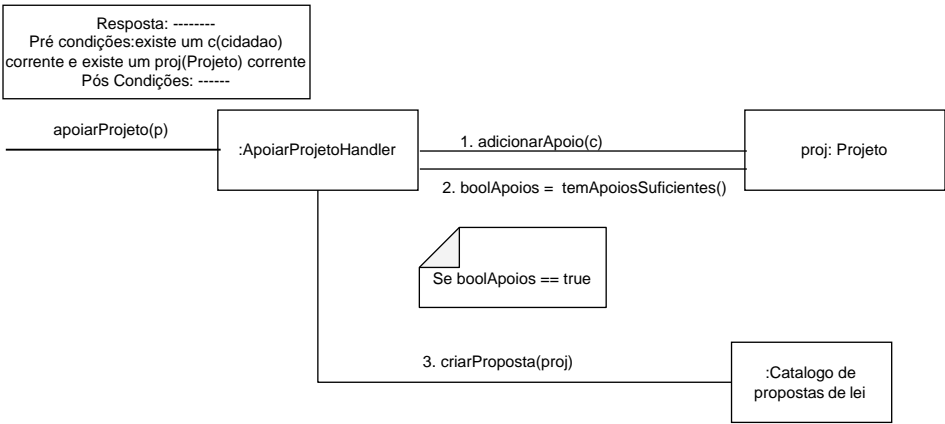
ID do caso de uso F



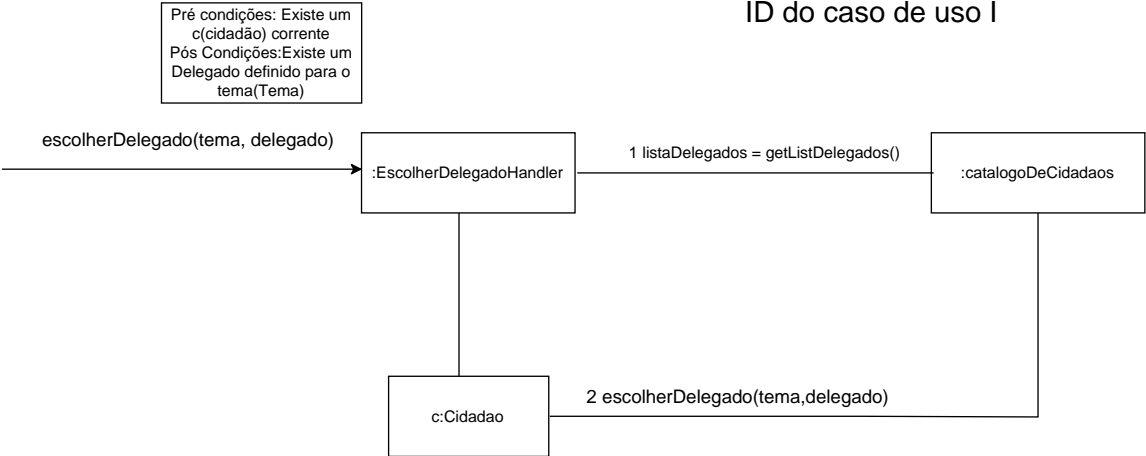
ID do caso de uso G



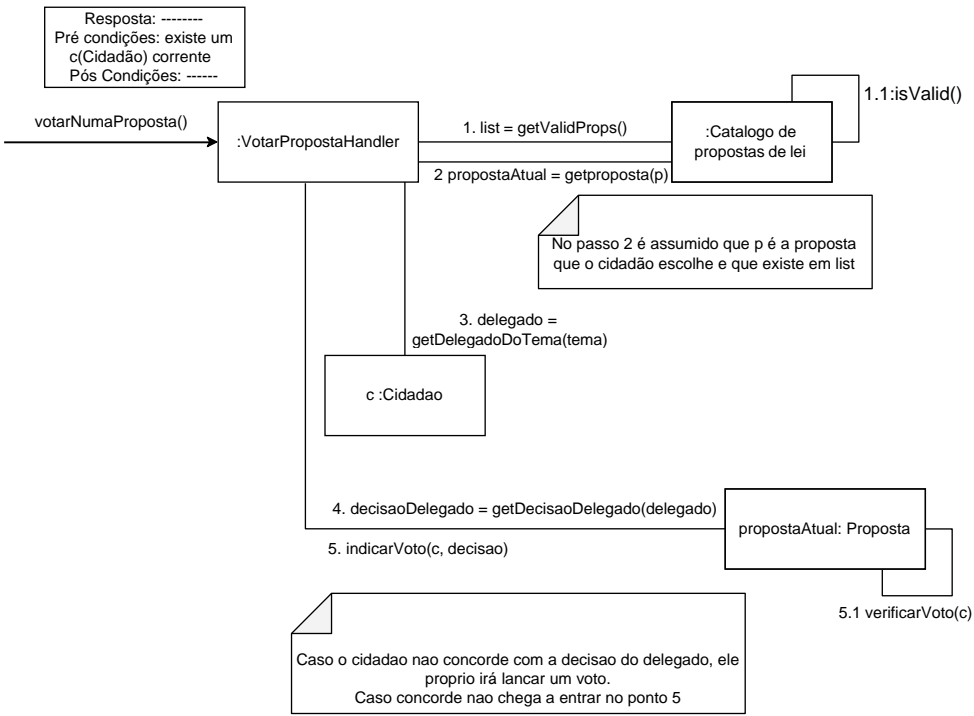
ID do caso de uso H



ID do caso de uso I



ID do caso de uso J



ID do caso de uso K

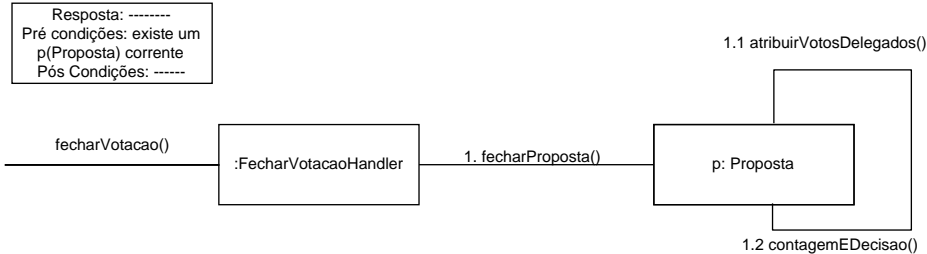
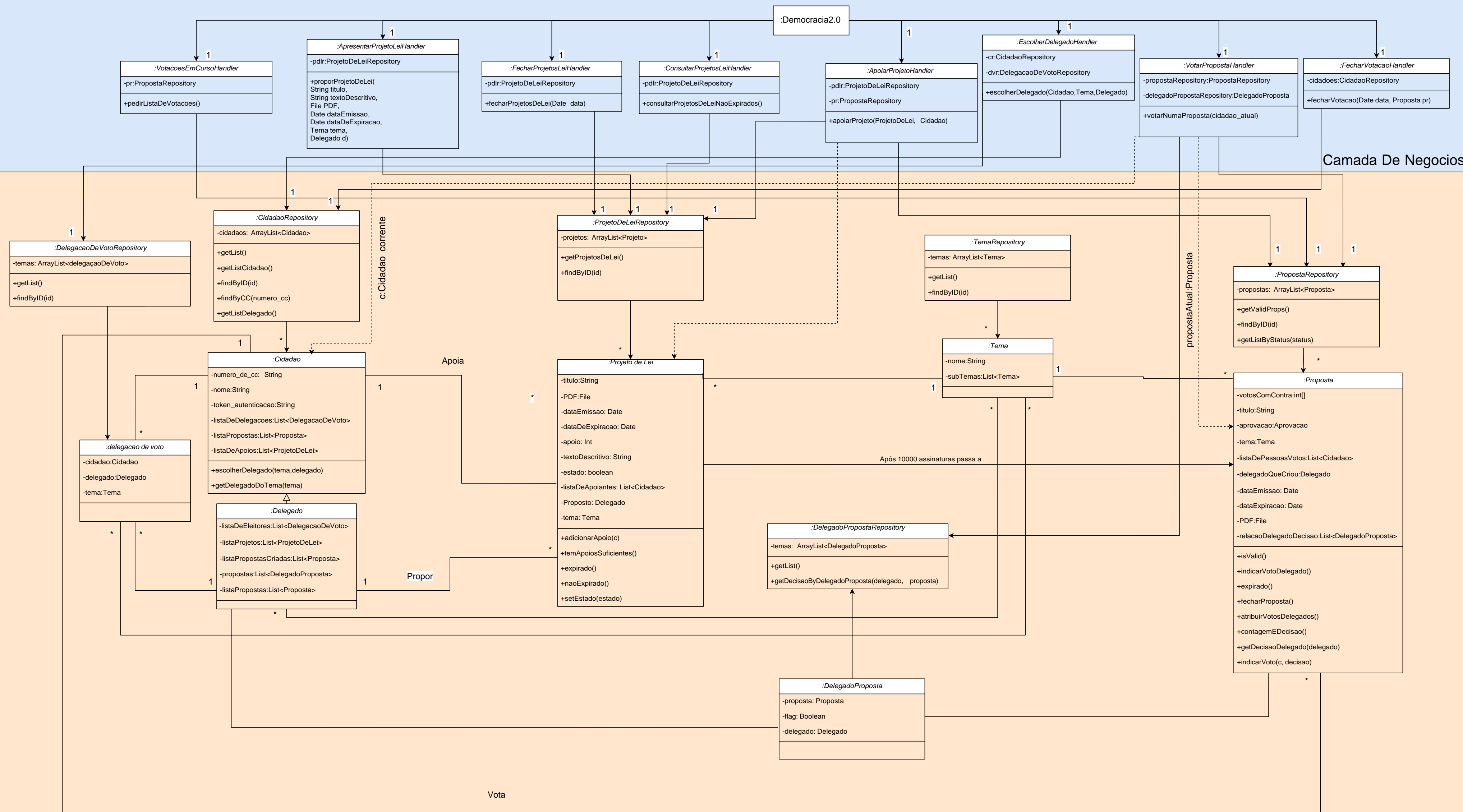


Diagrama de Classes



Camada De Dados

Anotações JPA usadas no Projeto – Fase #1

Classe *Cidadao*:

```
@Entity  
public class Cidadao {
```

- **@Entity**: informa que classe *Cidadao* representa uma entidade que é mapeada para uma tabela.

```
@Column(insertable = false, updatable = false)  
private String dtype;
```

- **@Column**: permite realizar o mapeamento do campo *dtype* com uma coluna na base de dados.
- **(insertable = false, updatable = false)**: indica que não é permitido inserir ou atualizar os dados da coluna correspondente a *dtype*

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

- **@Id**: marca atributo *id* como a chave primária da entidade *Cidadao*
- **GeneratedValue**: define a estratégia
- **(strategy = GenerationType.SEQUENCE)**: informa que os valores de *id* serão gerados a partir de uma sequência

```
@NotNull private String numero_de_cc;
```

- **@NotNull**: informa que o atributo *numero_de_cc* não pode ser nulo

```
@NotNull private String nome;
```

- **@NotNull**: informa que o atributo *nome* não pode ser nulo

```
@OneToMany(
    mappedBy = "cidadao",
    cascade = {CascadeType.ALL}
)
private List<DelegacaoDeVoto> listaDeDelegacoes;
```

- **@OneToMany**: define uma relação 1-N entre *Cidadao* e *listaDeDelegacoes*
- (**mappedBy = "cidadao"**): define que a relação entre *Cidadao* e *listaDeDelegacoes* é mapeada por "cidadao"
- (**{cascade = CascadeType.ALL}**): persiste *listaDeDelegacoes* para todas as operações (MERGE, PERSIST, REFRESH e REMOVE)

```
@ManyToMany(
    mappedBy = "listaDePessoasVotos",
    cascade = {CascadeType.ALL}
)
private List<Proposta> listaPropostas;
```

- **@ManyToMany**: define uma relação N-N entre *Cidadao* e *listaPropostas*
- (**mappedBy = "listaDePessoasVotos"**): define que a relação entre *Cidadao* e *listaPropostas* é mapeada por "listaDePessoasVotos"
- (**{cascade = CascadeType.ALL}**): persiste *listaPropostas* para todas as operações (MERGE, PERSIST, REFRESH e REMOVE)

```
@ManyToMany(
    mappedBy = "listaDeApoiantes",
    cascade = {CascadeType.ALL}
)
private List<ProjetoDeLei> listaDeApoios;
```

- **@ManyToMany**: define uma relação N-N entre *Cidadao* e *listaDeApoios*
- (**mappedBy = "listaDeApoiantes"**): define que a relação entre *Cidadao* e *listaDeApoios* é mapeada por "listaDeApoiantes"
- (**{cascade = CascadeType.ALL}**): persiste *listaDeApoios* para todas as operações (MERGE, PERSIST, REFRESH e REMOVE)

```
public Cidadao(@NotNull String numero_de_cc, @NotNull String nome, String token_autenticacao) {
    this.numero_de_cc = numero_de_cc;
    this.nome = nome;
    this.token_autenticacao = token_autenticacao;
    listaDeDelegacoes = new ArrayList<DelegacaoDeVoto>();
}
```

- **@NotNull**: informa que os atributos *numero_de_cc* e *nome* não podem ser nulos


```
@NonNull
public String getNome() {
    return nome;
}
```

- **@NonNull:** informa que o atributo *nome* não pode ser nulo

```
public void setNome(@NonNull String nome) {
    this.nome = nome;
}
```

- **@NonNull:** informa que o atributo *nome* não pode ser nulo

```
public void setNumero_de_cc(@NonNull String numero_de_cc) {
    this.numero_de_cc = numero_de_cc;
}
```

- **@NonNull:** informa que o atributo *numero_de_cc* não pode ser nulo

```
public void escolherDelegado(Tema t, @NonNull Delegado d) {
    listaDeDelegacoes.add(new DelegacaoDeVoto(this, d, t));
}
```

- **@NonNull:** informa que o atributo *d* do type *Delegado* não pode ser nulo

```
@Override
public boolean equals(Object obj) {
    if (obj == this) return true;
    if (obj == null || obj.getClass() != this.getClass()) return false;
    var that = (Cidadao) obj;
    return Objects.equals(this.id, that.id)
        && Objects.equals(this.nome, that.nome)
        && Objects.equals(this.numero_de_cc, that.numero_de_cc)
        && Objects.equals(this.token_autenticacao, that.token_autenticacao);
}
```

- **@Override:** informa que se está a sobrescrever o método *equals* e não a criar um novo

```
@Override
public int hashCode() {
    return Objects.hash(numero_de_cc, nome, token_autenticacao);
}
```

- **@Override:** informa que se está a sobrescrever o método *hashCode* e não a criar um novo

```
@Override
public String toString() {
    return "Author["
        + "numero_de_cc="
        + numero_de_cc
        + ", "
        + "nome="
        + nome
        + ", "
        + "token_autenticacao="
        + token_autenticacao
        + ']';
}
```

- **@Override:** informa que se está a sobrescrever o método *toString* e não a criar um novo

Classe *DelegacaoDeVoto*:

```
@Entity  
public class DelegacaoDeVoto {
```

- **@Entity**: informa que classe *DelegacaoDeVoto* representa uma entidade que é mapeada para uma tabela.

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

- **@Id**: marca atributo *id* como a chave primária da entidade *DelegacaoDeVoto*
- **GeneratedValue**: define a estratégia
- **(strategy = GenerationType.SEQUENCE)**: informa que os valores de *id* serão gerados a partir de uma sequência

```
@NotNull @ManyToOne private Cidadao cidadao;
```

- **@NotNull**: informa que o atributo *cidadao* não pode ser nulo
- **@ManyToOne**: define uma relação N-1 entre *DelegacaoDeVoto* e *cidadao*

```
@NotNull @ManyToOne private Delegado delegado;
```

- **@NotNull**: informa que o atributo *delegado* não pode ser nulo
- **@ManyToOne**: define uma relação N-1 entre *DelegacaoDeVoto* e *delegado*

```
@ManyToOne @NotNull private Tema tema;
```

- **@ManyToOne**: define uma relação N-1 entre *DelegacaoDeVoto* e *tema*
- **@NotNull**: informa que o atributo *tema* não pode ser nulo

```
public DelegacaoDeVoto(  
    @NotNull Cidadao cidadao, @NotNull Delegado delegado, @NotNull Tema temaDelegado) {  
    this.cidadao = cidadao;  
    this.delegado = delegado;  
    this.tema = temaDelegado;  
}
```

- **@NotNull**: informa que os atributos *cidadao*, *delegado*, e *temaDelegado* não podem ser nulos

```
@NotNull  
public Cidadao getCidadao() {  
    return cidadao;  
}
```

- **@NotNull**: informa que o atributo *cidadao* não pode ser nulo

```
public void setCidadao(@NonNull Cidadao cidadao) {
    this.cidadao = cidadao;
}
```

- **@NonNull:** informa que o atributo *cidadao* não pode ser nulo

```
@NonNull
public Delegado getDelegado() {
    return delegado;
}
```

- **@NonNull:** informa que o atributo *delegado* não pode ser nulo

```
public void setDelegado(@NonNull Delegado delegado) {
    this.delegado = delegado;
}
```

- **@NonNull:** informa que o atributo *delegado* não pode ser nulo

```
@NonNull
public Tema getTemaDelegado() {
    return tema;
}
```

- **@NonNull:** informa que o atributo *tema* não pode ser nulo

```
public void setTemaDelegado(@NonNull Tema temaDelegado) {
    this.tema = temaDelegado;
}
```

- **@NonNull:** informa que o atributo *temaDelegado* não pode ser nulo

```
@Override
public boolean equals(Object obj) {
    if (obj == this) return true;
    if (obj == null || obj.getClass() != this.getClass()) return false;
    var that = (DelegacaoDeVoto) obj;
    return Objects.equals(this.id, that.id)
        && Objects.equals(this.cidadao, that.cidadao)
        && Objects.equals(this.delegado, that.delegado)
        && Objects.equals(this.tema, that.tema);
}
```

- **@Override:** informa que se está a sobrescrever o método *equals* e não a criar um novo

Classe *Delegado*:

```
@Entity  
public class Delegado extends Cidadao {
```

- **@Entity**: informa que classe *Delegado* representa uma entidade que é mapeada para uma tabela.

```
@OneToMany(mappedBy = "delegado", cascade = CascadeType.ALL)  
private List<DelegacaoDeVoto> lista_eleitores;
```

- **@OneToMany**: define uma relação 1-N entre *Delegado* e *lista_eleitores*
- **(mappedBy = "delegado")**: define que a relação entre *Delegado* e *lista_eleitores* é mapeada por "delegado"
- **(cascade = CascadeType.ALL)**: persiste *lista_eleitores* para todas as operações (MERGE, PERSIST, REFRESH e REMOVE)

```
@ManyToMany(mappedBy = "relacaoDelegadoDecisao", cascade = CascadeType.ALL)  
private List<Proposta> listaPropostas;
```

- **@ManyToMany**: define uma relação N-N entre *Delegado* e *listaPropostas*
- **(mappedBy = "relacaoDelegadoDecisao")**: define que a relação entre *Delegado* e *listaPropostas* é mapeada por "relacaoDelegadoDecisao"
- **(cascade = CascadeType.ALL)**: persiste *listaPropostas* para todas as operações (MERGE, PERSIST, REFRESH e REMOVE)

```
@OneToMany(mappedBy = "proposto", cascade = CascadeType.ALL)  
private List<ProjetoDeLei> listaProjetos;
```

- **@OneToMany**: define uma relação 1-N entre *Delegado* e *listaProjetos*
- **(mappedBy = "proposto")**: define que a relação entre *Delegado* e *listaProjetos* é mapeada por "proposto"
- **(cascade = CascadeType.ALL)**: persiste *listaProjetos* para todas as operações (MERGE, PERSIST, REFRESH e REMOVE)

```
@OneToMany(mappedBy = "delegadoQueCriou", cascade = CascadeType.ALL)  
private List<Proposta> listaPropostasCriadas;
```

- **@OneToMany**: define uma relação 1-N entre *Delegado* e *listaPropostasCriadas*
- **(mappedBy = "delegadoQueCriou")**: define que a relação entre *Delegado* e *listaPropostasCriadas* é mapeada por "delegadoQueCriou"
- **(cascade = CascadeType.ALL)**: persiste *listaPropostasCriadas* para todas as operações (MERGE, PERSIST, REFRESH e REMOVE)

```
@OneToMany(mappedBy = "delegado", cascade = CascadeType.ALL, orphanRemoval = true)
private List<DelegadoProposta> propostas = new ArrayList<>();
```

- **@OneToMany**: define uma relação 1-N entre *Delegado* e *propostas*
- **(mappedBy = "delegado")**: define que a relação entre *Delegado* e *propostas* é mapeada por "delegado"
- **(cascade = CascadeType.ALL)**: persiste *propostas* para todas as operações (MERGE, PERSIST, REFRESH e REMOVE)
- **(orphanRemoval = true)**: seleciona a entidade "filha" (*Delegado*) a ser removida quando já não é referenciada pela entidade "parente" (*Cidadao*)

```
public Delegado(@NonNull String numero_de_cc, @NonNull String nome, String token_autenticacao) {
    super(numero_de_cc, nome, token_autenticacao);
}
```

- **@NonNull**: informa que os atributos *numero_de_cc* e *nome* não podem ser nulos

Classe *DelegadoProposta*:

```
@Entity  
public class DelegadoProposta {
```

- **@Entity**: informa que classe *DelegadoProposta* representa uma entidade que é mapeada para uma tabela.

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

- **@Id**: marca atributo *id* como a chave primária da entidade *DelegadoProposta*
- **GeneratedValue**: define a estratégia
- **(strategy = GenerationType.SEQUENCE)**: informa que os valores de *id* serão gerados a partir de uma sequência

```
@ManyToOne private Proposta proposta;
```

- **@ManyToOne**: define uma relação N-1 entre *DelegadoProposta* e *proposta*

```
@ManyToOne private Delegado delegado;
```

- **@ManyToOne**: define uma relação N-1 entre *DelegadoProposta* e *delegado*

```
@Column(name = "flag")  
private boolean flag;
```

- **@Column(name = "flag")**: permite realizar o mapeamento do campo *flag* com uma coluna na base de dados com o nome "flag"

Classe *ProjetoDeLei*:

```
@Entity  
public class ProjetoDeLei {
```

- **@Entity**: informa que classe *ProjetoDeLei* representa uma entidade que é mapeada para uma tabela.

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

- **@Id**: marca atributo *id* como a chave primária da entidade *ProjetoDeLei*
- **GeneratedValue**: define a estratégia
- **(strategy = GenerationType.SEQUENCE)**: informa que os valores de *id* serão gerados a partir de uma sequência

```
@NonNull private String titulo;
```

- **@NonNull**: informa que o atributo *titulo* não pode ser nulo

```
@Lob private File PDF;
```

- **@Lob**: permite persistir dados binários (neste caso um ficheiro: File *PDF*)

```
@Temporal(TemporalType.DATE)  
private Date dataEmissao;
```

- **@Temporal(TemporalType.DATE)**: permite persistir dados do tipo Date: *dataEmissao*

```
@Temporal(TemporalType.DATE)  
private Date dataDeExpiracao;
```

- **@Temporal(TemporalType.DATE)**: permite persistir dados do tipo Date: *dataDeExpiracao*

```
@NonNull private String textoDescritivo;
```

- **@NonNull**: informa que o atributo *textoDescritivo* não pode ser nulo

```
@NonNull private boolean estado;
```

- **@NonNull**: informa que o atributo *estado* não pode ser nulo


```
@ManyToMany private List<Cidadao> listaDeApoiantes;
```

- **@ManyToMany:** define uma relação N-N entre *ProjetoDeLei* e *listaDeApoiantes*

```
@ManyToOne @NotNull private Delegado proposto;
```

- **@ManyToOne:** define uma relação N-1 entre *ProjetoDeLei* e *proposto*
- **@NotNull:** informa que o atributo *proposto* não pode ser nulo

```
@OneToOne @NotNull private Tema tema;
```

- **@OneToOne:** define uma relação 1-1 entre *ProjetoDeLei* e *tema*
- **@NotNull:** informa que o atributo *tema* não pode ser nulo

```
@NotNull  
public Tema getTema() {  
    return tema;  
}
```

- **@NotNull:** informa que o atributo *tema* não pode ser nulo

```
public void setTema(@NotNull Tema tema) {  
    this.tema = tema;  
}
```

- **@NotNull:** informa que o atributo *tema* não pode ser nulo

```
@NotNull  
public Delegado getProposto() {  
    return proposto;  
}
```

- **@NotNull:** informa que o atributo *proposto* não pode ser nulo

```
public void setProposto(@NotNull Delegado proposto) {  
    this.proposto = proposto;  
}
```

- **@NotNull:** informa que o atributo *proposto* não pode ser nulo

Classe *Proposta*:

```
@Entity  
public class Proposta {
```

- **@Entity**: informa que classe *Proposta* representa uma entidade que é mapeada para uma tabela.

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

- **@Id**: marca atributo *id* como a chave primária da entidade *Proposta*
- **GeneratedValue**: define a estratégia
- **(strategy = GenerationType.SEQUENCE)**: informa que os valores de *id* serão gerados a partir de uma sequência

```
@NotNull private String titulo;
```

- **@NotNull**: informa que o atributo *titulo* não pode ser nulo

```
@ManyToMany(fetch = FetchType.EAGER)  
private List<Cidadao> listaDePessoasVotos = new ArrayList<>();
```

- **@ManyToMany**: define uma relação N-N entre *Proposta* e *listaDePessoasVotos*
- **(fetch = FetchType.EAGER)**: informa que ocorre Eager Loading com o atributo *listaDePessoasVotos*, ou seja, os dados são carregados mesmo que não venham a ser utilizados

```
@NotNull private Aprovacao aprovacao;
```

- **@NotNull**: informa que o atributo *aprovacao* não pode ser nulo

```
@Temporal(TemporalType.DATE)  
private Date dataEmissao;
```

- **@Temporal(TemporalType.DATE)**: permite persistir dados do tipo Date: *dataEmissao*

```
@Temporal(TemporalType.DATE)  
private Date dataExpiracao;
```

- **@Temporal(TemporalType.DATE)**: permite persistir dados do tipo Date: *dataDeExpiracao*

```
@Lob private File PDF;
```

- **@Lob:** permite persistir dados binários (neste caso um ficheiro: File *PDF*)

```
@ManyToOne @NotNull private Delegado delegadoQueCriou;
```

- **@ManyToOne:** define uma relação N-1 entre *Proposta* e *delegadoQueCriou*
- **@NotNull:** informa que o atributo *delegadoQueCriou* não pode ser nulo

```
@OneToMany(  
    fetch = FetchType.EAGER,  
    mappedBy = "proposta",  
    cascade = CascadeType.ALL,  
    orphanRemoval = true  
)  
private List<DelegadoProposta> relacaoDelegadoDecisao = new ArrayList<>();
```

- **@OneToMany:** define uma relação 1-N entre *Proposta* e *relacaoDelegadoDecisao*
- **(fetch = FetchType.EAGER):** informa que ocorre Eager Loading com o atributo *relacaoDelegadoDecisao*, ou seja, os dados são carregados mesmo que não venham a ser utilizados
- **(mappedBy = "proposta"):** que a relação entre *Proposta* e *relacaoDelegadoDecisao* é mapeada por "proposta"
- **(cascade = CascadeType.ALL):** persiste *relacaoDelegadoDecisao* para todas as operações (MERGE, PERSIST, REFRESH e REMOVE)
- **(orphanRemoval = true):** seleciona a entidade "filha" a ser removida quando já não é referenciada pela entidade "parente"

```
@ManyToOne private Tema tema;
```

- **@ManyToOne:** define uma relação N-1 entre *Proposta* e *tema*

Classe *Tema*:

```
@Entity  
public class Tema {
```

- **@Entity**: informa que classe *Tema* representa uma entidade que é mapeada para uma tabela.

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

- **@Id**: marca atributo *id* como a chave primária da entidade *Tema*
- **GeneratedValue**: define a estratégia
- **(strategy = GenerationType.SEQUENCE)**: informa que os valores de *id* serão gerados a partir de uma sequência

```
@NonNull private String nome;
```

- **@NonNull**: informa que o atributo *nome* não pode ser nulo

```
@ManyToOne private Tema parent;
```

- **@ManyToOne**: define uma relação N-1 entre *Tema* e *parente*

```
@OneToMany(mappedBy = "parent", cascade = CascadeType.ALL)  
private List<Tema> subTemas;
```

- **@OneToMany**: define uma relação 1-N entre *Tema* e *subTemas*
- **(mappedBy = "parent")**: define que a relação entre *Tema* e *subTemas* é mapeada por "*parent*"
- **(cascade = CascadeType.ALL)**: persiste *subTemas* para todas as operações (MERGE, PERSIST, REFRESH e REMOVE)

```
@OneToMany(mappedBy = "tema", cascade = CascadeType.ALL)  
private List<DelegacaoDeVoto> listaDeDelegacoes;
```

- **@OneToMany**: define uma relação 1-N entre *Tema* e *listaDeDelegacoes*
- **(mappedBy = "tema")**: define que a relação entre *Tema* e *listaDeDelegacoes* é mapeada por "*tema*"
- **(cascade = CascadeType.ALL)**: persiste *listaDeDelegacoes* para todas as operações (MERGE, PERSIST, REFRESH e REMOVE)

```
@OneToMany(mappedBy = "tema", cascade = CascadeType.ALL)
private List<Proposta> listaDePropostas;
```

- **@OneToMany**: define uma relação 1-N entre *Tema* e *listaDePropostas*
- **(mappedBy = "tema")**: define que a relação entre *Tema* e *listaDePropostas* é mapeada por "tema"
- **(cascade = CascadeType.ALL)**: persiste *listaDePropostas* para todas as operações (MERGE, PERSIST, REFRESH e REMOVE)

```
@NotNull
@Column(unique = true)
public String getNome() {
    return nome;
}
```

- **@NotNull**: informa que o atributo *nome* não pode ser nulo
- **@Column(unique = true)**: permite a coluna mapeada para um atributo *nome* com um valor único(unique = true)

```
public void setNome(@NotNull String nome) {
    this.nome = nome;
}
```

- **@NotNull**: informa que o atributo *nome* não pode ser nulo

```
public Tema(@NotNull String nome) {
    this.nome = nome;
}
```

- **@NotNull**: informa que o atributo *nome* não pode ser nulo

Classe *VotarPropostaHandler*:

```
@Transactional
public Proposta votarNumaProposta(Cidadao cidadao_atual) throws Exception {
```

- **@Transactional**: especifica as semânticas das transações no método *votarNumaProposta*

Classe *CidadaoRepository*:

```
@Query("SELECT a FROM Cidadao a")  
List<Cidadao> getList();
```

- **@Query("SELECT a FROM Cidadao a")**: permite fazer consulta de tabelas de entidades em SQL. Neste caso pretende-se consultar a lista de todos os cidadãos.

```
@Query("SELECT c FROM Cidadao c WHERE c.dtype = Delegado ")  
List<Delegado> getListDelegado();
```

- **@Query("SELECT c FROM Cidadao c WHERE c.dtype = Delegado")**: permite fazer consulta de tabelas de entidades em SQL. Neste caso pretende-se consultar a lista de cidadãos que sejam também eles delegados.

```
@Query("SELECT c FROM Cidadao c WHERE c.dtype = Cidadao ")  
List<Cidadao> getListCidadao();
```

- **@Query("SELECT c FROM Cidadao c WHERE c.dtype = Cidadao")**: permite fazer consulta de tabelas de entidades em SQL. Neste caso pretende-se consultar a lista de cidadãos que sejam apenas cidadãos e não sejam delegados.

```
@Query("SELECT c FROM Cidadao c WHERE c.id = :id")  
Cidadao findById(@Param("id") Long id);
```

- **@Query("SELECT c FROM Cidadao c WHERE c.id = :id")**: permite fazer consulta de tabelas de entidades em SQL. Neste caso pretende-se procurar um determinado cidadão usando o *id* como parâmetro de pesquisa (@Param).

```
@Query("SELECT c FROM Cidadao c WHERE c.numero_de_cc LIKE %:q% ")  
Cidadao findByCC(@Param("q") String q);
```

- **@Query("SELECT c FROM Cidadao c WHERE c.numero_de_cc LIKE %:q% ")**: permite fazer consulta de tabelas de entidades em SQL. Neste caso pretende-se procurar um determinado cidadão usando o número de cartão de cidadão (*numero_de_cc*) como parâmetro de pesquisa (@Param).

Classe *DelegacaoDeVotoRepository*:

```
@Query("SELECT a FROM DelegacaoDeVoto a")
List<DelegacaoDeVoto> getList();
```

- **@Query("SELECT a FROM DelegacaoDeVoto a")**: permite fazer consulta de tabelas de entidades em SQL. Neste caso pretende-se consultar a lista de todas as delegações de voto.

```
@Query("SELECT dv FROM DelegacaoDeVoto dv WHERE dv.id = :id")
DelegacaoDeVoto findById(@Param("id") Long id);
```

- **@Query("SELECT dv FROM DelegacaoDeVoto dv WHERE dv.id = :id")**: permite fazer consulta de tabelas de entidades em SQL. Neste caso pretende-se procurar uma determinada delegação de voto usando o *id* como parâmetro de pesquisa (@Param).

Classe *DelegadoPropostaRepository*:

```
@Query("SELECT l FROM ProjetoDeLei l")
List<DelegadoProposta> getList();
```

- **@Query("SELECT l FROM ProjetoDeLei l")**: permite fazer consulta de tabelas de entidades em SQL. Neste caso pretende-se consultar a lista de todos os elementos do tipo *DelegadoProposta*

```
@Query("SELECT dl FROM DelegadoProposta dl WHERE dl.id = :id")
DelegadoProposta findById(@Param("id") Long id);
```

- **@Query("SELECT dl FROM DelegadoProposta dl WHERE dl.id = :id")**: permite fazer consulta de tabelas de entidades em SQL. Neste caso pretende-se procurar um determinado elemento do tipo *DelegadoProposta* usando o *id* como parâmetro de pesquisa (@Param).

```
@Query(
    "SELECT dl FROM DelegadoProposta dl WHERE dl.proposta = :proposta AND dl.delegado = :delegado")
DelegadoProposta getDecisaoByDelegadoProposta(
    @Param("proposta") Proposta propostaCorrente, @Param("delegado") Delegado dd);
```

- **@Query("SELECT dl FROM DelegadoProposta dl WHERE dl.proposta = :proposta AND dl.delegado = :delegado")**: permite fazer consulta de tabelas de entidades em SQL. Neste caso pretende-se procurar um determinado elemento do tipo *DelegadoProposta* usando a *propostaCorrente* e o *dd* do tipo *Delegado* como parâmetros de pesquisa (@Param).

Classe *ProjetoDeLeiRepository*:

```
@Query("SELECT l FROM ProjetoDeLei l")  
List<ProjetoDeLei> getList();
```

- **@Query("SELECT l FROM ProjetoDeLei l")**: permite fazer consulta de tabelas de entidades em SQL. Neste caso pretende-se consultar a lista de todos os projetos de lei.

```
@Query("SELECT pl FROM ProjetoDeLei pl WHERE pl.id = :id")  
ProjetoDeLei findById(@Param("id") Long id);
```

- **@Query("SELECT pl FROM ProjetoDeLei pl WHERE pl.id = :id")**: permite fazer consulta de tabelas de entidades em SQL. Neste caso pretende-se procurar um determinado projeto de lei usando o *id* como parâmetro de pesquisa (@Param).

Classe *PropostaRepository*:

```
@Query("SELECT p FROM Proposta p")  
List<Proposta> getList();
```

- **@Query("SELECT p FROM Proposta p")**: permite fazer consulta de tabelas de entidades em SQL. Neste caso pretende-se consultar a lista de todas as propostas.

```
@Query("SELECT p FROM Proposta p WHERE p.id = :id")  
Proposta findById(@Param("id") Long id);
```

- **@Query("SELECT p FROM Proposta p WHERE p.id = :id")**: permite fazer consulta de tabelas de entidades em SQL. Neste caso pretende-se procurar uma determinada proposta usando o *id* como parâmetro de pesquisa (@Param).

```
@Query("SELECT p FROM Proposta p WHERE p.aprovacao = :status")  
List<Proposta> getListByStatus(@Param("status") Aprovacao status);
```

- **@Query("SELECT p FROM Proposta p WHERE p.aprovacao = :status")**: permite fazer consulta de tabelas de entidades em SQL. Neste caso pretende-se consultar a lista de todas as propostas usando o *status* como parâmetro de pesquisa (@Param).

Classe *TemaRepository*:

```
@Query("SELECT a FROM Tema a")  
List<Tema> getList();
```

- **@Query("SELECT a FROM Tema a")**: permite fazer consulta de tabelas de entidades em SQL. Neste caso pretende-se consultar a lista de todos os temas.

```
@Query("SELECT t FROM Tema t WHERE t.id = :id")  
Tema findById(@Param("id") Long id);
```

- **@Query("SELECT t FROM Tema t WHERE t.id = :id")**: permite fazer consulta de tabelas de entidades em SQL. Neste caso pretende-se procurar um determinado tema usando o *id* como parâmetro de pesquisa (@Param).

Classe *DemoApplication*:

```
@SpringBootApplication  
public class DemoApplication {
```

- **@SpringBootTest**: permite ter o contexto do Spring inicializado em cada teste pertencente à classe *DemoApplication*

```
@Autowired EntityManagerFactory emf;
```

- **@Autowired**: fornece controlo sobre onde e como a ligação entre os beans deve ser realizada sendo usado neste caso para o atributo correspondente ao criador de entity managers: *emf*

```
@Bean  
public CommandLineRunner demo()
```

- **@Bean**: oferece ao método em questão dependências do XML *<bean/>*

Classe *DemoApplicationTests*:

```
@SpringBootTest  
class DemoApplicationTests {
```

- **@SpringBootTest**: permite ter o contexto do Spring inicializado em cada teste pertencente à classe *DemoApplicationTests*

```
@Autowired private ProjetoDeLeiRepository projetoDeLeiRepository;  
@Autowired private PropostaRepository propostaRepository;  
@Autowired private CidadaoRepository cidadaoRepository;  
@Autowired private DelegacaoDeVotoRepository delegacaoDeVotoRepository;  
@Autowired private TemaRepository temaRepository;  
@Autowired private DelegadoPropostaRepository delegadoPropostaRepository;
```

- **@Autowired**: fornece controlo sobre onde e como a ligação entre os beans deve ser realizada sendo usado neste caso para cada repositório das entidades: *projetoDeLeiRepository*, *propostaRepository*, *cidadaoRepository*, *delegacaoDeVotoRepository*, *temaRepository* e *delegadoPropostaRepository*

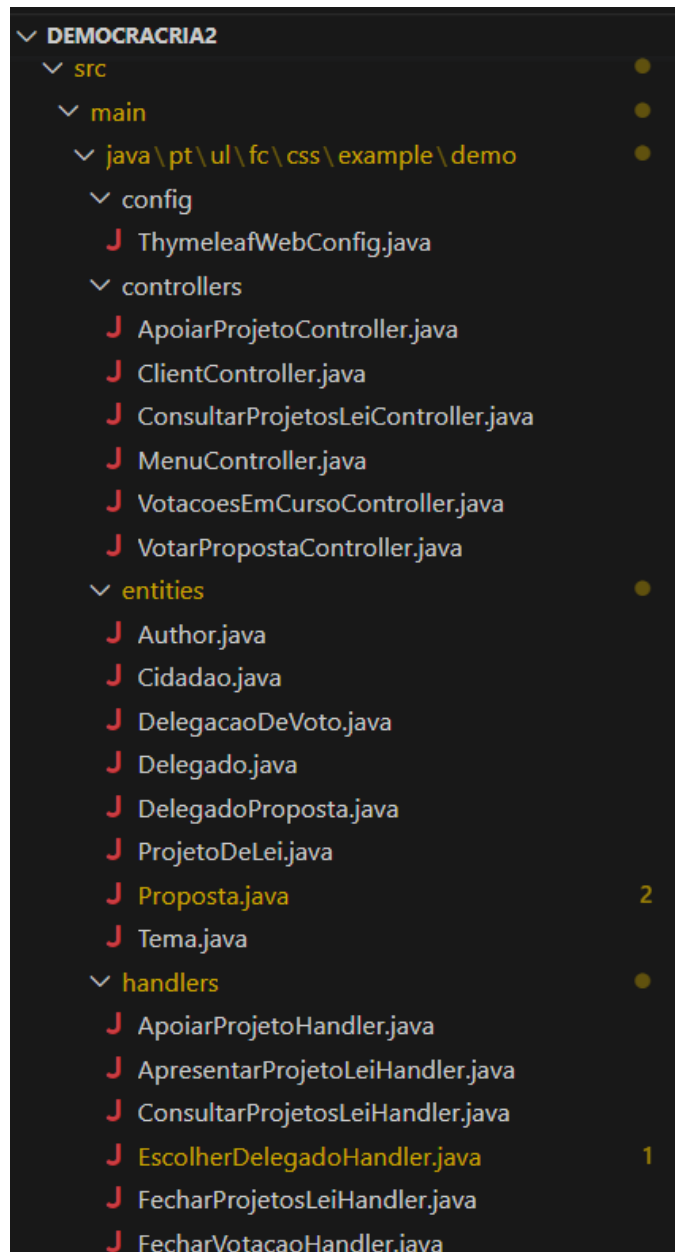
```
@Test
```

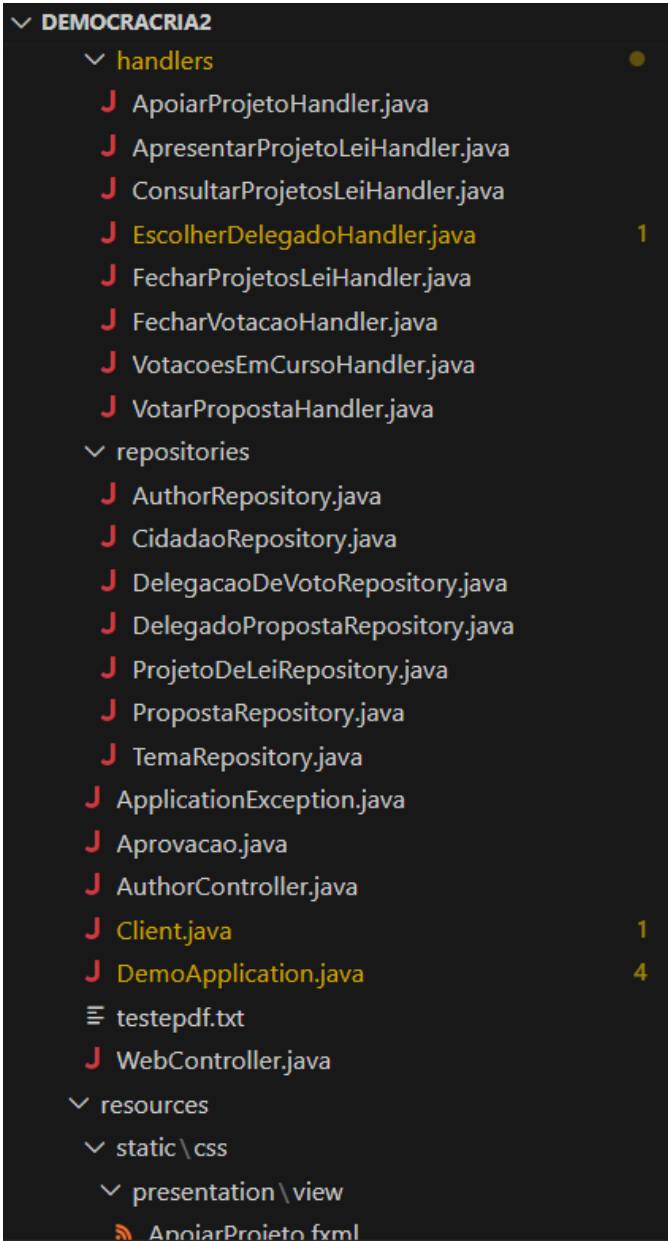
- **@Test**: identifica que um método com esta anotação corresponde a um teste JUnit

Arquitetura interna dos componentes – Fase #2

Divisão de componentes:

A fase 2 do projeto está composta por ficheiros correspondentes à aplicação WEB, à API REST, à aplicação desktop usando JavaFX e Scheduled Tasks em métodos dos handlers.





▼ DEMOCRACRIA2

▼ resources

▼ static\css

▼ presentation\view

🔗 ApoiarProjeto.fxml

🔗 ConsultarProjetosLei.fxml

🔗 MenuPrincipal.fxml

🔗 VotacoesEmCurso.fxml

🔗 VotarProposta.fxml

style.css

▼ templates

🔗 apoiarProjeto.html

🔗 apresentarProjetoLei.html

🔗 author.html

🔗 consultarProjetosLei.html

🔗 escolherDelegado.html

🔗 index.html

🔗 menu.html

🔗 votacoesEmCurso.html

🔗 votarProposta.html

☰ application.properties

▼ test

▼ java\pt\ul\fc\css\example\demo

🔗 DemoApplicationTests.java

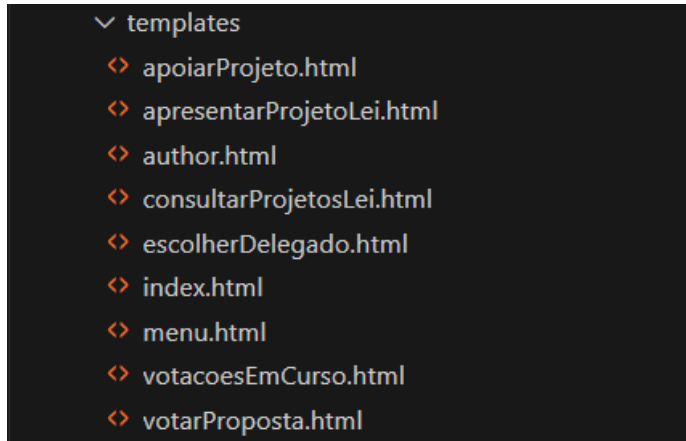
5

▼ resources

☰ application.properties

Aplicação WEB (F_{2w}):

Os ficheiros correspondentes à aplicação WEB são os ficheiros html para cada caso de uso. Estes ficheiros estão presentes no diretório *templates* como é possível verificar na imagem abaixo:



O caso de uso D permite obter uma listagem das propostas de lei em votação atualmente e está representado no ficheiro *votacoesEmCurso.html*

No caso de uso E, um delegado pode propor um projeto de lei e está representado no ficheiro *apresentarProjetoLei.html*

O caso de uso G permite listar e consultar os projetos de lei não expirados estando representado no ficheiro *consultarProjetosLei.html*

No caso de uso H, um projeto de lei pode ser apoiado por cidadãos e está representado no ficheiro *apoiarProjeto.html*

No caso de uso I, um cidadão pode escolher vários delegados e está representado no ficheiro *escolherDelegado.html*

No caso de uso J, um cidadão pede a listagem das votações e escolhe a que lhe interessa estando representado no ficheiro *votarProposta.html*

Existe um ficheiro que corresponde a um menu onde o utilizador pode escolher a página HTML de cada caso de uso que pretende aceder estando representado em *menu.html*

Caso de uso D (*votacoesEmCurso.html*):

```
<tr data-th-each="proposta : ${propostas}">
  <td data-th-text="${proposta.id}"></td>
  <td data-th-text="${proposta.titulo}"></td>
  <td data-th-text="${proposta.dataEmissao}"></td>
  <td data-th-text="${proposta.dataExpiracao}"></td>
  <td data-th-text="${proposta.delegadoQueCriou.getNome()}"></td>
  <td data-th-text="${proposta.PDF}"></td>
  <td data-th-text="${proposta.votosComContra[0]}"></td>
  <td data-th-text="${proposta.votosComContra[1]}"></td>
</tr>
```

- **data-th-each:** permite iterar sobre um datatype, neste caso uma determinada proposta num conjunto de propostas
- **data-th-text:** atribui um determinado valor a um atributo de uma entidade proposta

Caso de uso E (*apresentarProjetoLei.html*):

```
<div class="mb-3">
  <label for="exampleInputTitulo" class="form-label">Introduza o titulo do projeto de lei:</label>
  <input type="text" name="titulo" class="form-control" id="exampleInputTitulo" aria-describedby="TituloInsert" placeholder="Titulo do projeto">
</div>
<div class="mb-3">
  <label for="exampleInputTextoDescritivo" class="form-label">Introduza o texto descritivo do projeto de lei:</label>
  <input type="text" name="textoDescritivo" class="form-control" id="exampleInputTextoDescritivo" aria-describedby="TextoDescritivoInsert" placeholder="Texto descritivo do projeto">
</div>
<div class="mb-3">
  <label for="exampleInputDataDeExpiracao" class="form-label">Introduza a data de expiração do projeto de lei:</label>
  <input type="text" name="dataDeExpiracao" class="form-control" id="exampleInputDataDeExpiracao" aria-describedby="DataDeExpiracaoInsert" placeholder="Data de expiração do projeto">
</div>
<div class="mb-3">
  <label for="exampleInputTema" class="form-label">Introduza o tema do projeto de lei:</label>
  <input type="text" name="tema" class="form-control" id="exampleInputTema" aria-describedby="TemaInsert" placeholder="Tema do projeto">
</div>
<div class="mb-3">
  <label for="exampleInputDelegado" class="form-label">Introduza o delegado proponente do projeto:</label>
  <input type="text" name="proposto" class="form-control" id="exampleInputDelegado" aria-describedby="DeleagdoInsert" placeholder="Delegado proponente do projeto">
</div>
```

- **name:** permite usar o valor de um determinado campo da entidade ProjetoDeLei num servidor WEB

Caso de uso G (*consultarProjetosLei.html*):

```
<tr th:each="projetoDeLei : ${projetosDeLei}">
  <td th:text="${projetoDeLei.id}"></td>
  <td th:text="${projetoDeLei.titulo}"></td>
  <td th:text="${projetoDeLei.PDF}"></td>
  <td th:text="${projetoDeLei.dataEmissao}"></td>
  <td th:text="${projetoDeLei.dataDeExpiracao}"></td>
  <td th:text="${projetoDeLei.apoio}"></td>
  <td th:text="${projetoDeLei.textoDescritivo}"></td>
  <td th:text="${projetoDeLei.estado}"></td>
  <td>
    <span th:each="apoio : ${projetoDeLei.listaDeApoiantes}" th:text="${apoio.getNome()}"></span>
  </td>
  <td th:text="${projetoDeLei.proposto.getNome()}"></td>
  <td th:text="${projetoDeLei.tema.getNome()}"></td>
</tr>
```

- **th:each:** permite iterar sobre um datatype, neste caso um determinado projeto de lei num conjunto de projetos de lei
- **th:text:** atribui um determinado valor a um atributo de uma entidade projetoDeLei

Caso de uso H (*apoiarProjeto.html*):

```
<form method="POST" action="/apoiarProjeto">
  <div class="mb-3">
    <label for="exampleInputID1" class="form-label">Escolha o Projeto que deseja apoiar</label>
    <input type="text" name="id" class="form-control" id="exampleInputID1" aria-describedby="IdInsert" placeholder="ID do projeto">
  </div>
  <button type="submit" class="btn btn-primary">Submeter</button>
</form>
```

- **method="POST":** especifica o modo de envio de dados do formulário, sendo neste caso um método HTTP do tipo POST, adicionando o conteúdo introduzido como um novo elemento
- **action="/apoiarProjeto":** especifica o caminho para onde os dados do formulário são enviados, neste caso para /apoiarProjeto
- **name:** permite usar o valor de um determinado campo da entidade *ProjetoDeLei* num servidor WEB

Caso de uso I (*escolherDelegado.html*):

```
<form>
  <div class="mb-3">
    <label for="exampleInputEmail1" class="form-label">Escolha o Delegado que quer eleger</label>
    <input type="text" name="id" class="form-control" id="exampleInputEmail1" aria-describedby="emailHelp" placeholder="ID do delegado">
  </div>

  <div class="mb-3">
    <label for="exampleInputPassword1" class="form-label">Tema(s) do delegado</label>
    <input type="text" class="form-control" id="exampleInputPassword1"
      placeholder="Introduza o(s) tema(s) relacionados com o seu delegado">
    <div id="emailHelp" name="temas" class="form-text">Caso introduza mais que um tema separe os temas por virgulas,
      por exemplo, "geral, saude, educacao"</div>
  </div>

  <button type="submit" class="btn btn-primary">Submeter</button>
</form>
```

- **name:** permite usar o valor de um determinado campo da entidade Delegado num servidor WEB

Caso de uso J (*votarProposta.html*):

```
<form method="POST" action="/votarProposta">

  <div class="mb-3">
    <label for="exampleInputID1" class="form-label">Qual o ID da proposta que deseja votar?</label>
    <input type="text" name="id_prop" class="form-control" id="exampleInputID1" aria-describedby="IdInsert" placeholder="ID da proposta">
  </div>

  <div class="mb-3">
    <label for="exampleInputID1" class="form-label">Qual a sua Votação? T caso esteja a favor e F caso esteja contra.</label>
    <input type="text" name="decision" class="form-control" id="exampleInputID1" aria-describedby="IdInsert" placeholder="ID da proposta">
  </div>

  <button type="submit" class="btn btn-primary">Submeter Voto</button>
</form>
```

- **method="POST":** especifica o modo de envio de dados do formulário, sendo neste caso um método HTTP do tipo POST, adicionando o conteúdo introduzido como um novo elemento
- **action="/votarProposta":** especifica o caminho para onde os dados do formulário são enviados, neste caso para /votarProposta
- **name:** permite usar o valor de um determinado campo da entidade Proposta num servidor WEB

Menu para navegar pelos casos de uso (*menu.html*):

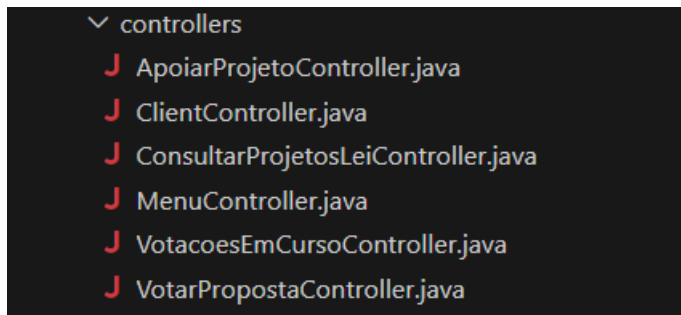
```
<div class="container">

  <div class="d-flex flex-column align-items-center">
    <h1>Democracia 2.0</h1>
    <div class="mb-3">
      <button class="btn btn-outline-danger fixed-width-button" onclick="redirectTo('/votacoesEmCurso')">Lista de votacoes em curso</button>
    </div>
    <div class="mb-3">
      <button class="btn btn-outline-danger fixed-width-button" onclick="redirectTo('/apresentarProjetoLei')">Apresentar um projeto de lei</button>
    </div>
    <div class="mb-3">
      <button class="btn btn-outline-danger fixed-width-button" onclick="redirectTo('/apoiarProjeto')">Apoiar projeto de lei</button>
    </div>
    <div class="mb-3">
      <button class="btn btn-outline-danger fixed-width-button" onclick="redirectTo('/consultarProjetosLei')">Consultar projetos de lei</button>
    </div>
    <div class="mb-3">
      <button class="btn btn-outline-danger fixed-width-button" onclick="redirectTo('/escolherDelegado')">Escolher delegado</button>
    </div>
    <div class="mb-3">
      <button class="btn btn-outline-danger fixed-width-button" onclick="redirectTo('/votarProposta')">Votar numa proposta</button>
    </div>
  </div>
</div>
<script>
  function redirectTo(url) {
    window.location.href = url;
  }
</script>
```

- **onclick="redirectTo('/<path>')"**: quando o botão correspondente a cada caso de uso é pressionado, o utilizador acede à página html desse caso de uso graças ao método redirectTo com o URL correspondente ao caso de uso

API REST (F_{2R}):

Os ficheiros correspondentes à API REST são os ficheiros *Controller* para cada caso de uso. Estes ficheiros estão presentes no diretório *controllers* como é possível verificar na imagem abaixo:



O caso de uso D permite obter uma listagem das propostas de lei em votação atualmente e está representado no ficheiro *VotacoesEmCursoController.java*

O caso de uso G permite listar e consultar os projetos de lei não expirados estando representado no ficheiro *ConsultarProjetosLeiController.java*

No caso de uso H, um projeto de lei pode ser apoiado por cidadãos e está representado no ficheiro *ApoiarProjetoController.java*

No caso de uso J, um cidadão pede a listagem das votações e escolhe a que lhe interessa estando representado no ficheiro *VotarPropostaController.java*

Existe um ficheiro que corresponde a um menu onde o utilizador pode escolher a API REST de cada caso de uso que pretende aceder estando representado em *MenuController.java*

Caso de uso D (*VotacoesEmCursoController.java*):

```
@Controller
public class VotacoesEmCursoController {

    @Autowired private PropostaRepository propostaRepository;

    @Autowired VotacoesEmCursoHandler vCH = new VotacoesEmCursoHandler(propostaRepository);

    @GetMapping("/votacoesEmCurso")
    public ModelAndView votacoes(final Model model) {
        ModelAndView mv = new ModelAndView(viewName:"votacoesEmCurso");
        mv.addObject(attributeName:"propostas", vCH.pedirListaDeVotacoes(Aprovacao.EM_CURSO));
        return mv;
    }
}
```

- **@Controller:** permite criar um Map do model object e encontrar uma view na classe *controller* do caso de uso D
- **@Autowired:** fornece controlo sobre onde e como a ligação entre os beans deve ser realizada sendo usado neste caso para o repositório e handler envolvidos neste caso de uso: *propostaRepository* e *vCH* (*VotacoesEmCursoHandler*)
- **@GetMapping("/votacoesEmCurso"):** corresponde a um pedido HTTP com verbo GET e URL */votacoesEmCurso*, ou seja, obtém uma representação das propostas de lei em votação atualmente

Caso de uso G (*ConsultarProjetosLeiController.java*):

```
@Controller
public class ConsultarProjetosLeiController {

    @Autowired private ProjetoDeLeiRepository projetoDeLeiRepository;

    @Autowired
    private ConsultarProjetosLeiHandler cplh =
        new ConsultarProjetosLeiHandler(projetoDeLeiRepository);

    @GetMapping("/consultarProjetosLei")
    public ModelAndView projetosDeLei(final Model model) {
        ModelAndView mv = new ModelAndView(viewName:"consultarProjetosLei");
        mv.addObject(attributeName:"projetosDeLei", cplh.consultarProjetosDeLeiNaoExpirados());
        return mv;
    }
}
```

- **@Controller:** permite criar um Map do model object e encontrar uma view na classe *controller* do caso de uso G
- **@Autowired:** fornece controlo sobre onde e como a ligação entre os beans deve ser realizada sendo usado neste caso para o repositório e handler envolvidos neste caso de uso: *projetoDeLeiRepository* e *cplh* (*ConsultarProjetosLeiHandler*)
- **@GetMapping("/consultarProjetosLei"):** corresponde a um pedido HTTP com verbo GET e URL */consultarProjetosLei*, ou seja, *obtem uma representação dos projetos de lei não expirados*

Caso de uso H (*ApoiarProjetoController.java*):

```
@Controller
public class ApoiarProjetoController {

    @Autowired ProjetoDeLeiRepository plr;

    @Autowired PropostaRepository pr;

    @Autowired CidadaoRepository cr;

    @GetMapping("/apoiarProjeto")
    public String apoiarProjetoPagina() {
        return "apoiarProjeto";
    }

    @PostMapping("/apoiarProjeto")
    public void apoiarProjeto(@RequestParam("id") String projectId) {
        ApoiarProjetoHandler aph = new ApoiarProjetoHandler(plr, pr);
        List<Cidadao> listaDeCidadaos = cr.getListCidadao();
        Random rand = new Random();
        int randomCidadao = rand.nextInt(listaDeCidadaos.size());

        Optional<ProjetoDeLei> pp = plr.findById(Long.valueOf(projectId));
        if (pp.isPresent()) {
            ProjetoDeLei pp2 = pp.get();
            System.out.println(pp2.getProposto().getNome());
            aph.apoiarprojeto(pp2, listaDeCidadaos.get(randomCidadao));
        }
        System.out.println("Votou no projeto com ID: " + projectId);
    }
}
```

- **@Controller:** permite criar um Map do model object e encontrar uma view na classe *controller* do caso de uso H
- **@Autowired:** fornece controlo sobre onde e como a ligação entre os beans deve ser realizada sendo usado neste caso para os repositórios e handlers envolvidos neste caso de uso: *plr* (*ProjetoDeLeiRepository*), *pr* (*PropostaRepository*) e *cr* (*CidadaoRepository*)
- **@GetMapping("/apoiarProjeto"):** corresponde a um pedido HTTP com verbo GET e URL */apoiarProjeto*, ou seja, obtém um projeto de lei que pode ser apoiado por cidadãos
- **@PostMapping("/apoiarProjeto"):** corresponde a um pedido HTTP com verbo POST e URL */apoiarProjeto*, ou seja, adiciona o conteúdo do body com um novo elemento de *apoiarProjeto*

Caso de uso J (*VotarPropostaController.java*):

```
@Controller
public class VotarPropostaController {

    @Autowired PropostaRepository pr;

    @Autowired DelegadoPropostaRepository dr;

    @Autowired CidadaoRepository cr;

    @GetMapping("/votarProposta")
    public String votarPropostaPagina() {
        return "votarProposta";
    }

    @PostMapping("/votarProposta")
    public void votarProposta(
        @RequestParam("id_prop") String propId, @RequestParam("decision") String decision)
        throws Exception {
        VotarPropostaHandler vph = new VotarPropostaHandler(pr, dr);
        Proposta propCorr = pr.findById(Long.valueOf(propId));
        Boolean votacao = true;
        if (decision.equals(anObject:"T")) {
            votacao = true;
        } else if (decision.equals(anObject:"F")) {
            votacao = false;
        }
        List<Cidadao> listaCid = cr.getList();
        Random rand = new Random();
        Cidadao cidadaoRandom = listaCid.get(rand.nextInt(listaCid.size()));
        pr.save(vph.votarNumaProposta2(cidadaoRandom, propCorr, votacao));
    }
}
```

- **@Controller:** permite criar um Map do model object e encontrar uma view na classe *controller* do caso de uso J
- **@Autowired:** fornece controlo sobre onde e como a ligação entre os beans deve ser realizada sendo usado neste caso para os repositórios e handlers envolvidos neste caso de uso: *pr* (*PropostaRepository*), *dr* (*DelegadoPropostaRepository*) e *cr* (*CidadaoRepository*)
- **@GetMapping("/votarProposta"):** corresponde a um pedido HTTP com verbo GET e URL */votarProposta*, ou seja, *obtem* uma listagem das votações e escolhe a que lhe interessa
- **@PostMapping("/votarProposta"):** corresponde a um pedido HTTP com verbo POST e URL */votarProposta*, ou seja, adiciona o conteúdo do body com um novo elemento de *votarProposta*

Menu para navegar pelos casos de uso (*MenuController.java*):

```
@Controller
public class MenuController {
    @GetMapping({"/", "/menu"})
    public String menuPrincipalPagina() {
        return "menu";
    }

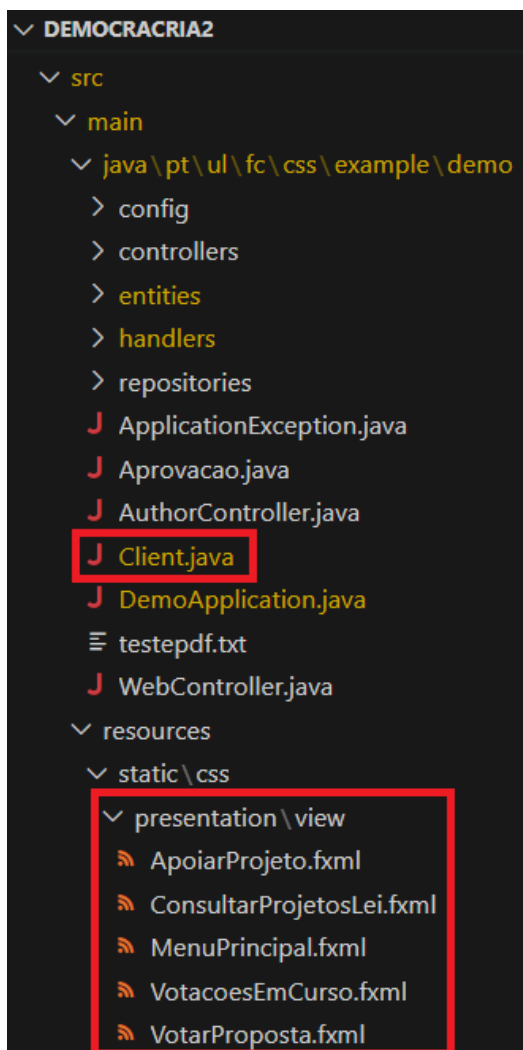
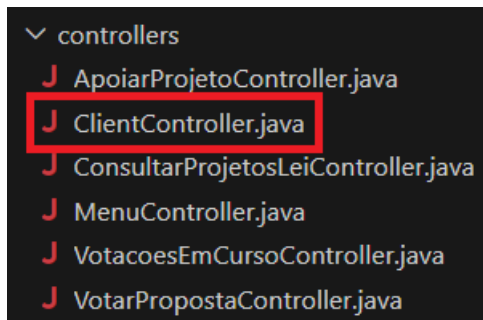
    @GetMapping("/escolherDelegado")
    public String escolherDelegadoPagina() {
        return "escolherDelegado";
    }

    @GetMapping("/apresentarProjetoLei")
    public String apresentarProjetoLeiPagina() {
        return "apresentarProjetoLei";
    }
}
```

- **@Controller:** permite criar um Map do model object e encontrar uma view na classe *controller* para o menu principal
- **@GetMapping ({"/", "/menu"}):** corresponde a um pedido HTTP com verbo GET e URL */menu*, ou seja, obtém uma representação do menu principal
- **@GetMapping ({"/escolherDelegado"}):** corresponde a um pedido HTTP com verbo GET e URL */escolherDelegado*, ou seja, obtém uma representação da escolha de vários delegados
- **@GetMapping ({"/apresentarProjetoLei"}):** corresponde a um pedido HTTP com verbo GET e URL */apresentarProjetoLei*, ou seja, obtém uma representação da apresentação de um projeto de lei

Aplicação desktop usando JavaFX (F_{2D}):

Os ficheiros correspondentes à aplicação desktop usando JavaFX são uma classe Client, um ClientController e os ficheiros fxml para cada caso de uso (neste caso, D, G, H e J) como é possível verificar nas imagens abaixo:



Classe Client:

Corresponde à Startup Class que integra os vários componentes como é possível observar na imagem abaixo:

```
public class Client extends Application {  
  
    @Override  
    public void start(Stage primaryStage) throws Exception {  
        String prefix = "/static/css/presentation/view/";  
  
        BorderPane root = new BorderPane();  
        FXMLLoader testeLoader = new FXMLLoader(getClass().getResource(prefix + "MenuPrincipal.fxml"));  
  
        root.setCenter(testeLoader.load());  
  
        ClientController controller = testeLoader.getController();  
  
        Scene scene = new Scene(root);  
        primaryStage.setScene(scene);  
        primaryStage.show();  
    }  
  
    Run | Debug  
    public static void main(String[] args) {  
        launch(args);  
    }  
}
```

Estende javafx.application.Application:

```
public class Client extends Application
```

Carrega o ficheiro FXML correspondente ao menu principal onde se pode aceder aos FXML de cada caso de uso, o que cria um nó e um controlador e define-se como se organizam os nós criados:

```
String prefix = "/static/css/presentation/view/";  
  
BorderPane root = new BorderPane();  
FXMLLoader testeLoader = new FXMLLoader(getClass().getResource(prefix + "MenuPrincipal.fxml"));  
  
root.setCenter(testeLoader.load());  
  
ClientController controller = testeLoader.getController();
```

Usa o nó raíz de modo a definir uma scene e colocá-la numa primary stage:

```
Scene scene = new Scene(root);  
primaryStage.setScene(scene);
```

Por último, lança-se a aplicação:

```
primaryStage.show();  
}  
  
Run | Debug  
public static void main(String[] args) {  
    launch(args);  
}
```

Classe ClientController:

Define o *Controller*, contendo também o código que se tem de executar quando o utilizador interage com um determinado componente do UI bem como os atributos onde são injetados nós da vista:

```
public class ClientController {  
  
    @FXML private ListView<String> listView1;  
  
    @FXML private ListView<String> listView2;  
  
    @FXML private ListView<String> listView3;  
  
    @FXML private ListView<String> listView4;  
  
    @FXML private ListView<String> listView5;  
  
    @FXML private ListView<String> listView6;  
  
    @FXML private ListView<String> listView7;  
  
    @FXML private ListView<String> listView8;  
  
}
```

```
String prefix = "/static/css/presentation/view/";  
  
public void redirectToVotacoesEmCurso(ActionEvent event) {  
    try {  
        FXMLLoader loader = new FXMLLoader(getClass().getResource(prefix + "VotacoesEmCurso.fxml"));  
        Parent root = loader.load();  
        ClientController controller = loader.getController();  
        controller.loadData();  
        Stage stage = (Stage) ((Button) event.getSource()).getScene().getWindow();  
        stage.setScene(new Scene(root));  
        stage.show();  
    } catch (IOException e) {  
        e.printStackTrace();  
    }  
}
```

Ficheiros FXML:

Definem a View do respetivo componente e têm ligação explícita com a Controller Class, neste caso, ClientController.

```
<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.scene.control.Button>
<?import javafx.scene.control.Label>
<?import javafx.scene.control.ListView>
<?import javafx.scene.layout.AnchorPane>
<?import javafx.scene.layout.ColumnConstraints>
<?import javafx.scene.layout.GridPane>
<?import javafx.scene.layout.RowConstraints>

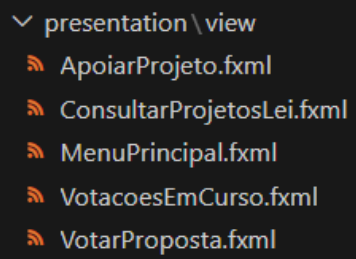
<AnchorPane prefHeight="700.0" prefWidth="1400.0" xmlns="http://javafx.com/javafx/19" xmlns:fx="http://javafx.com/fxml/1" fx:controller="pt.ul.fc.css.example.demo.controllers.ClientController">
    <children>
        <GridPane prefHeight="700.0" prefWidth="1500.0" AnchorPane.bottomAnchor="0.0" AnchorPane.leftAnchor="0.0" AnchorPane.rightAnchor="0.0" AnchorPane.topAnchor="0.0">
            <columnConstraints>
                <ColumnConstraints hgrow="SOMETIMES" maxWidth="513.0" minWidth="10.0" prefWidth="133.0" />
                <ColumnConstraints hgrow="SOMETIMES" maxWidth="513.0" minWidth="10.0" prefWidth="133.0" />
                <ColumnConstraints hgrow="SOMETIMES" maxWidth="513.0" minWidth="10.0" prefWidth="133.0" />
                <ColumnConstraints hgrow="SOMETIMES" maxWidth="513.0" minWidth="10.0" prefWidth="135.0" />
                <ColumnConstraints hgrow="SOMETIMES" maxWidth="513.0" minWidth="10.0" prefWidth="121.0" />
                <ColumnConstraints hgrow="SOMETIMES" maxWidth="513.0" minWidth="10.0" prefWidth="51.0" />
                <ColumnConstraints hgrow="SOMETIMES" maxWidth="513.0" minWidth="10.0" prefWidth="152.0" />
                <ColumnConstraints hgrow="SOMETIMES" maxWidth="513.0" minWidth="10.0" prefWidth="118.0" />
                <ColumnConstraints hgrow="SOMETIMES" maxWidth="513.0" minWidth="10.0" prefWidth="154.0" />
                <ColumnConstraints hgrow="SOMETIMES" maxWidth="513.0" minWidth="10.0" prefWidth="153.0" />
                <ColumnConstraints hgrow="SOMETIMES" maxWidth="513.0" minWidth="10.0" prefWidth="150.0" />
            </columnConstraints>
            <rowConstraints>
                <RowConstraints maxHeight="345.0" minHeight="10.0" prefHeight="60.0" vgrow="SOMETIMES" />
                <RowConstraints maxHeight="345.0" minHeight="10.0" prefHeight="60.0" vgrow="SOMETIMES" />
                <RowConstraints maxHeight="673.0" minHeight="10.0" prefHeight="600.0" vgrow="SOMETIMES" />
            </rowConstraints>
            <children>
                <Label fx:id="ID" prefHeight="267.0" prefWidth="282.0" text="ID" GridPane.rowIndex="1" />
                <Label prefHeight="267.0" prefWidth="282.0" text="Titulo" GridPane.columnIndex="1" GridPane.rowIndex="1" />
                <Label prefHeight="267.0" prefWidth="282.0" text="ID" GridPane.columnIndex="2" GridPane.rowIndex="1" />
                <Label prefHeight="267.0" prefWidth="282.0" text="Data de Emissão" GridPane.columnIndex="3" GridPane.rowIndex="1" />
                <Label prefHeight="267.0" prefWidth="282.0" text="Data de Expiração" GridPane.columnIndex="4" GridPane.rowIndex="1" />
                <Label prefHeight="267.0" prefWidth="282.0" text="Apoios" GridPane.columnIndex="5" GridPane.rowIndex="1" />
                <Label prefHeight="267.0" prefWidth="282.0" text="Texto Descritivo" GridPane.columnIndex="6" GridPane.rowIndex="1" />
                <Label prefHeight="267.0" prefWidth="282.0" text="Estado de Expiração" GridPane.columnIndex="7" GridPane.rowIndex="1" />
                <Label prefHeight="267.0" prefWidth="282.0" text="Lista de Apoiantes" GridPane.columnIndex="8" GridPane.rowIndex="1" />
                <Label prefWidth="282.0" text="Delegado Proponente" GridPane.columnIndex="9" GridPane.rowIndex="1" />
                <Label prefHeight="60.0" prefWidth="88.0" text="Tema" GridPane.columnIndex="10" GridPane.rowIndex="1" />
                <ListView fx:id="cListView1" prefHeight="200.0" prefWidth="200.0" GridPane.rowIndex="2" />
                <ListView fx:id="cListView2" prefHeight="200.0" prefWidth="200.0" GridPane.columnIndex="2" GridPane.rowIndex="2" />
                <ListView fx:id="cListView3" prefHeight="200.0" prefWidth="200.0" GridPane.columnIndex="3" GridPane.rowIndex="2" />
                <ListView fx:id="cListView4" prefHeight="200.0" prefWidth="200.0" GridPane.columnIndex="4" GridPane.rowIndex="2" />
                <ListView fx:id="cListView5" prefHeight="200.0" prefWidth="200.0" GridPane.columnIndex="5" GridPane.rowIndex="2" />
                <ListView fx:id="cListView6" prefHeight="200.0" prefWidth="200.0" GridPane.columnIndex="6" GridPane.rowIndex="2" />
                <ListView fx:id="cListView7" prefHeight="200.0" prefWidth="200.0" GridPane.columnIndex="7" GridPane.rowIndex="2" />
                <ListView fx:id="cListView8" prefHeight="200.0" prefWidth="200.0" GridPane.columnIndex="8" GridPane.rowIndex="2" />
                <ListView fx:id="cListView9" prefHeight="200.0" prefWidth="200.0" GridPane.columnIndex="9" GridPane.rowIndex="2" />
                <ListView fx:id="cListView10" prefHeight="200.0" prefWidth="200.0" GridPane.columnIndex="10" GridPane.rowIndex="2" />
                <Button mnemonicParsing="false" prefHeight="45.0" prefWidth="163.0" text="Voltar menu" onAction="#redirectToVoltar" />
            </children>
        </GridPane>
    </children>
</AnchorPane>
```

```
<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.scene.control.Button>
<?import javafx.scene.control.Label>
<?import javafx.scene.layout.AnchorPane>
<?import javafx.scene.text.Font>

<AnchorPane prefHeight="700.0" prefWidth="1200.0" xmlns="http://javafx.com/javafx/19" xmlns:fx="http://javafx.com/fxml/1" fx:controller="pt.ul.fc.css.example.demo.controllers.ClientController">
    <children>
        <AnchorPane prefHeight="700.0" prefWidth="1200.0">
            <children>
                <Button layoutX="450.0" layoutY="91.0" mnemonicParsing="false" prefHeight="50.0" prefWidth="300.0" text="Lista de votacoes em curso" onAction="#redirectToVotacoesEmCurso">
                    <font>
                        <font name="System Bold" size="20.0" />
                    </font>
                </Button>
                <Label layoutX="501.0" layoutY="14.0" prefHeight="50.0" prefWidth="199.0" text="Democracia 2.0" textAlignment="CENTER">
                    <font>
                        <font name="System Bold" size="27.0" />
                    </font>
                </Label>
                <Button layoutX="450.0" layoutY="182.0" mnemonicParsing="false" prefHeight="50.0" prefWidth="300.0" text="Apresentar um projeto de lei">
                    <font>
                        <font name="System Bold" size="20.0" />
                    </font>
                </Button>
                <Button layoutX="450.0" layoutY="364.0" mnemonicParsing="false" prefHeight="50.0" prefWidth="300.0" text="Consultar projetos de lei" onAction="#redirectToConsultarProjetosLei">
                    <font>
                        <font name="System Bold" size="20.0" />
                    </font>
                </Button>
                <Button layoutX="450.0" layoutY="521.0" mnemonicParsing="false" prefHeight="50.0" prefWidth="300.0" text="Apoiar projetos de lei" onAction="#redirectToApoiarProjeto">
                    <font>
                        <font name="System Bold" size="20.0" />
                    </font>
                </Button>
                <Button layoutX="450.0" layoutY="555.0" mnemonicParsing="false" prefHeight="50.0" prefWidth="300.0" text="Escolher delegado">
                    <font>
                        <font name="System Bold" size="20.0" />
                    </font>
                </Button>
                <Button layoutX="450.0" layoutY="746.0" mnemonicParsing="false" prefHeight="50.0" prefWidth="300.0" text="Votar numa proposta" onAction="#redirectToVotarProposta">
                    <font>
                        <font name="System Bold" size="20.0" />
                    </font>
                </Button>
            </children>
        </AnchorPane>
    </children>
</AnchorPane>
```

Neste projeto existem ficheiros FXML correspondentes a cada caso de uso, bem como um ficheiro FXML para o menu principal que permite navegar para cada FXML associado a um caso de uso.



```
presentation \ view
  ApoiarProjeto.fxml
  ConsultarProjetosLei.fxml
  MenuPrincipal.fxml
  VotacoesEmCurso.fxml
  VotarProposta.fxml
```

Scheduled Tasks (C):

As anotações correspondentes às Scheduled Tasks estão presentes nos métodos dentro das classes correspondentes aos handlers para cada caso de uso. Neste caso para os casos de uso F e K.

Caso de uso F (classe *FecharProjetosLeiHandler*):

```
@Scheduled(fixedDelay = 3000)
public List<ProjetoDeLei> fecharProjetosDeLei(Date data) throws InterruptedException {
    List<ProjetoDeLei> listaProjetosDeLei = pdlr.getList();
    for (ProjetoDeLei pdl : listaProjetosDeLei) {
        if (pdl.expirado(data)) {
            pdl.setEstado(estado:true);
            pdlr.save(pdl);
        }
    }
    System.out.println("Começando - " + LocalTime.now());
    Thread.sleep(millis:1000);
    System.out.println("Terminando - " + LocalTime.now());
    return listaProjetosDeLei;
}
```

- **@Scheduled(fixedDelay = 3000):** permite agendar tarefas num determinado período ou intervalos de tempo, neste caso o método *fecharProjetosDeLei* é executado a cada 3 segundos sendo que uma tarefa espera que a anterior termine de executar para começar a sua execução

Caso de uso K (classe *FecharVotacaoHandler*):

```
@Scheduled(fixedDelay = 3000)
public void fecharVotacoes(Date data) throws ApplicationException, InterruptedException {
    List<Proposta> listaPropostas = pr.getList();
    for (Proposta pr : listaPropostas) {
        if (!pr.expirado(data)) {
            throw new ApplicationException(
                message: "A(s) votação(ões) não podem ser fechados porque ainda não expiraram");
        }
        try {
            pr.fecharProposta(this.cidades.getList());
        } catch (Exception e) {
            throw new ApplicationException(message: "Erro ao fechar a Proposta", e);
        }
    }
    System.out.println("Começando - " + LocalTime.now());
    Thread.sleep(1000);
    System.out.println("Terminando - " + LocalTime.now());
}
```

- **@Scheduled(fixedDelay = 3000):** permite agendar tarefas num determinado período ou intervalos de tempo, neste caso o método *fecharVotacoes* é executado a cada 3 segundos sendo que uma tarefa espera que a anterior termine de executar para começar a sua execução