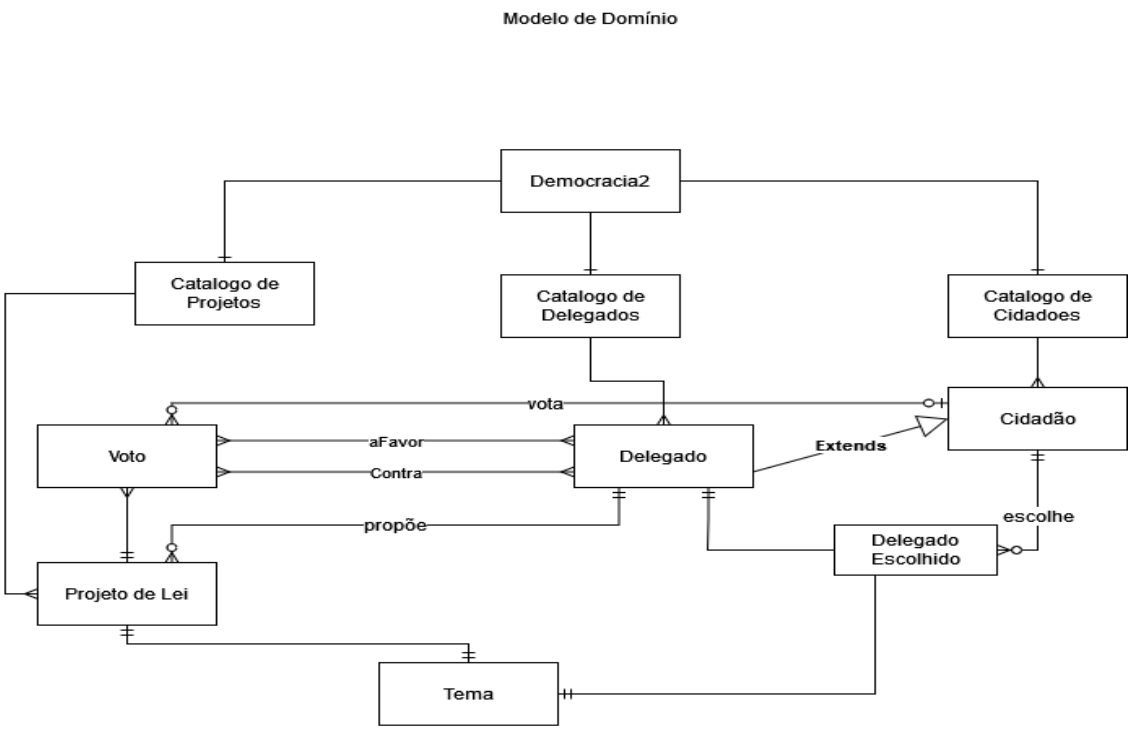
Modelo de domínio



SSD caso de uso de J:

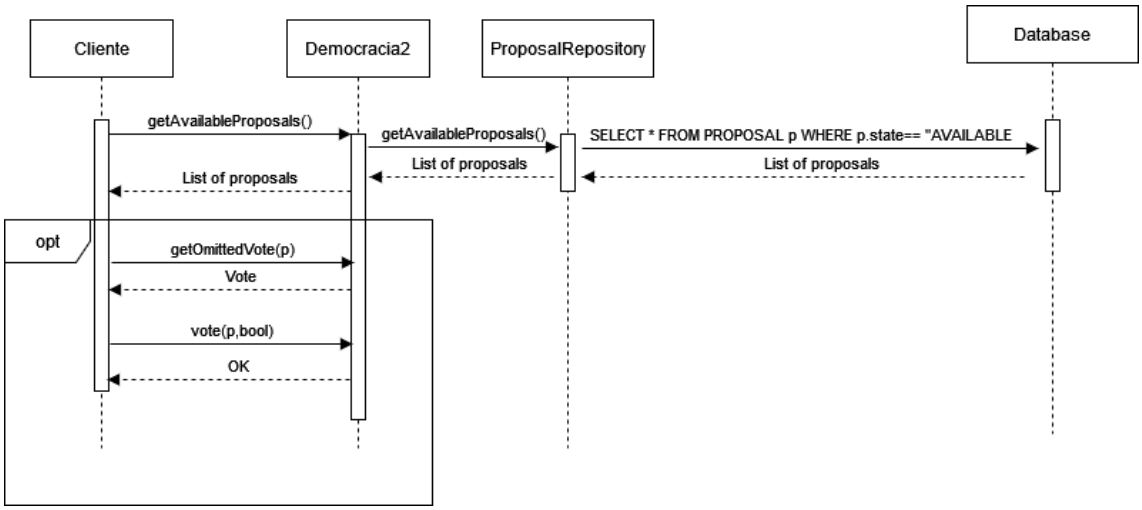
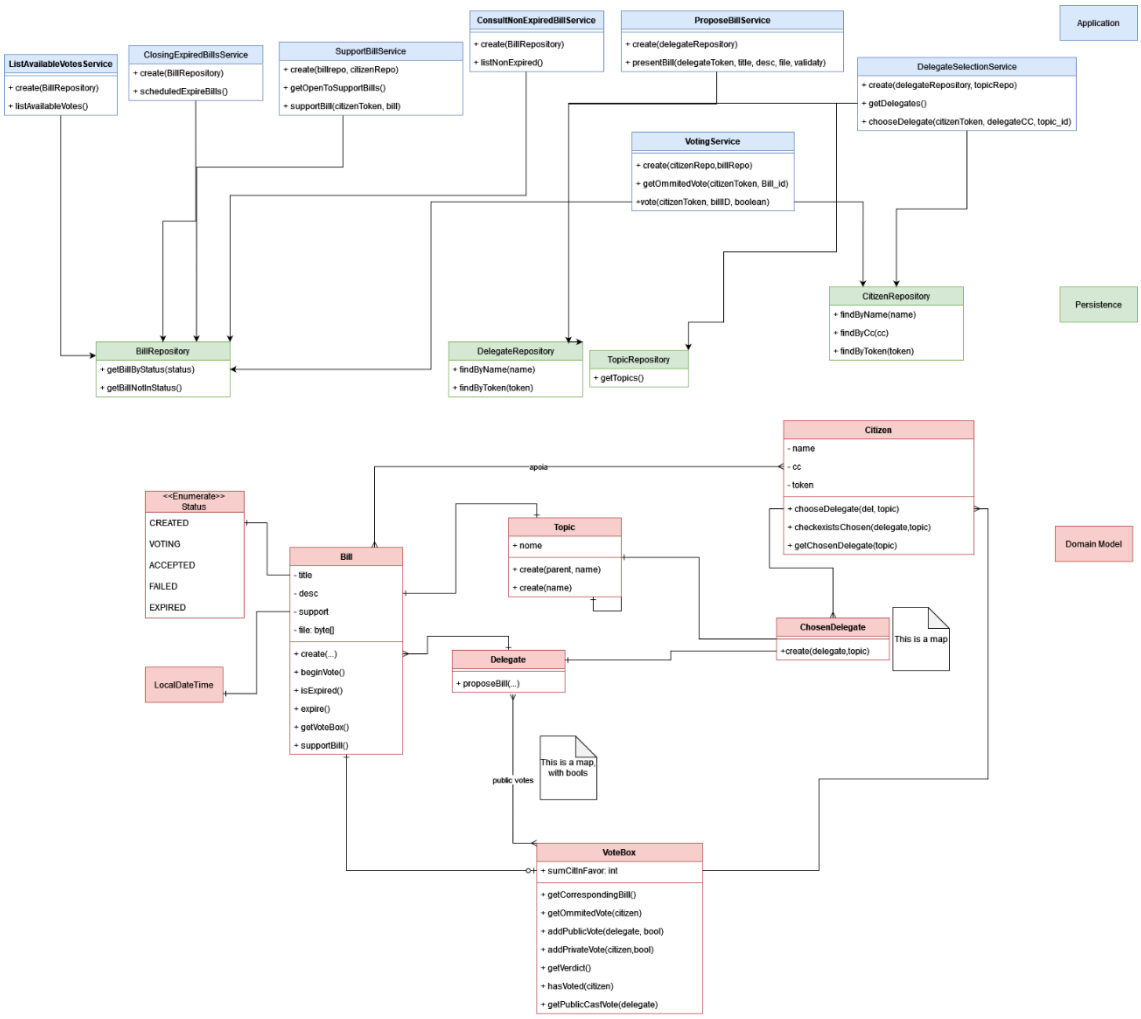
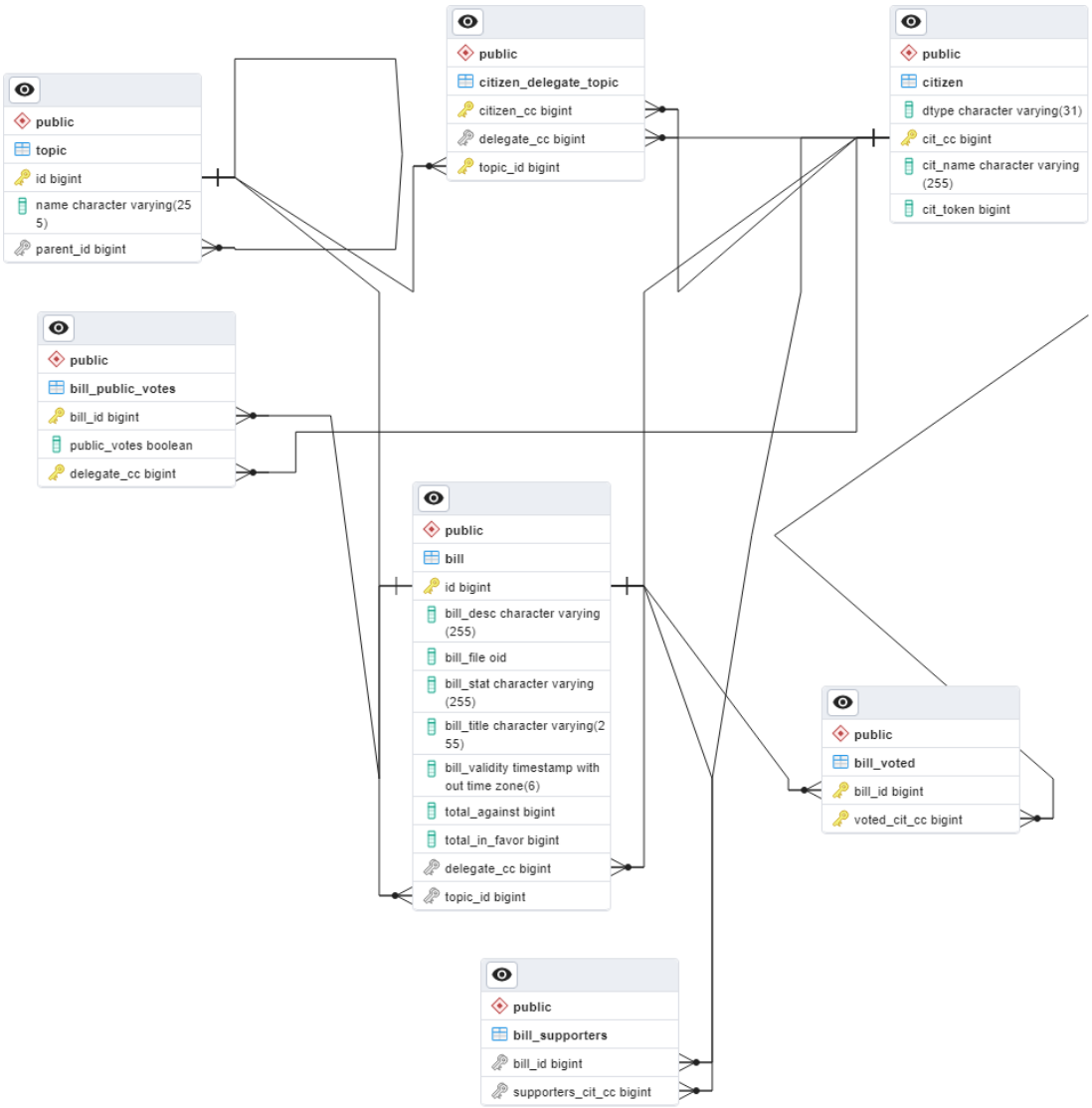


Diagrama de classes



Mapeamento para base de dados



Entidades

Cidadão e Delegado:

Ambos cidadão e delegado são entidades, pois são independentes e apresentam relações com outros objetos.

```java
@Entity
@Inheritance(strategy = SINGLE_TABLE)
public class Citizen {
```

A estratégia de inheritance é com single table inheritance uma vez, que o delegado que também é entity, apenas tem mais um campo que é uma oneToMany com mappedBy, ou seja, nem é sequer uma coluna a mais na table de delegado se existisse, mas sim na Bill.

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

  public Delegate() {
    // No-argument constructor
  }

  @OneToMany(mappedBy = "proponent")
  private List<Bill> bills;
```

Mapeamento especial no cidadão:

```java
@OneToMany
@JoinTable(
    name = "citizen_delegate_topic",
    joinColumns = {@JoinColumn(name = "citizen_cc", referencedColumnName = "CIT_CC")},
    inverseJoinColumns = {@JoinColumn(name = "delegate_cc", referencedColumnName = "CIT_CC")})
@MapKeyJoinColumn(name = "topic_id")
private Map<Topic, Delegate> chosenDelegates;
```

Para mapear o mapa é criado uma tabela, com colunas que contém o cc do cidadão, o tópico, e o delegado. E a chave primária é uma chave composta pelo cidadão e o tópico.
Aqui está a tabela criada:

```
create table citizen_delegate_topic (citizen_cc bigint not null, delegate_cc
bigint not null, topic_id bigint not null, primary key (citizen_cc, topic_id))
```

Topic

```java
@Entity
public class Topic {

  @Id @GeneratedValue() private Long id;

  @ManyToOne
  @JoinColumn(name = "parent_id")
  private Topic parent;
```

Entidade uma vez que é independente das Bills. Com uma relação consigo própria em que um pai pode ter vários filhos. Desta forma podemos começar uma procura pelo tópico mais específico.

Bill

```java
@Entity
public class Bill {

  public Bill() {
    // No-argument constructor
  }

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

A bill é uma entidade, pois é independentemente: e tem as seguintes relações com outras entidades:

```java
@ManyToOne
@JoinColumn(name = "delegate_cc", nullable = false)
private Delegate proponent;

@ManyToMany(cascade = CascadeType.ALL)
private List<Citizen> supporters;
```

A relação com o delegado.

Os supporters é uma relação many to many com o cidadão embora este não guarde as bills suportadas pois não havia razão nos casos de uso. Além de ter um cascade pois é uma tabela diferente que não interessa sem uma bill.

A data de validade é um temporal para armazenar na base de dados

```java
  @Column(name = "BILL_VALIDITY")
  @Temporal(TemporalType.TIMESTAMP)
  private LocalDateTime validity;
```

Além disso inclui um embeddable:

```java
@Embedded private VoteBox voteBox;
```

Uma vez que as caixas de voto não fazem sentido sem uma bill que correspondente.

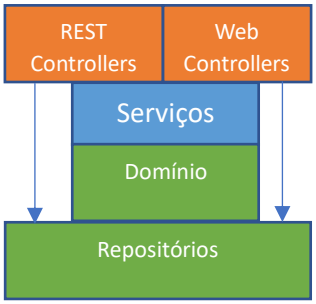
A VoteBox tem este aspeto:

```java
@Embeddable
public class VoteBox {

  @ElementCollection
  @MapKeyJoinColumn(name = "delegate_cc")
  private Map<Delegate, Boolean> publicVotes;

  @ManyToMany(cascade = CascadeType.ALL)
  private Set<Citizen> voted;

  private long totalInFavor; // number of citizens in favor
  private long totalAgainst;
```

Mais uma vez é mapeado um mapa, que desta vez é um elemento collection de booleans (votos), e é criada uma tabela que inclui estes dados em que o id da bill e do delegado formam uma chave composta para poder encontrar o voto do delegado para a bill.

Relatório componentes fase 2

Server:



Os controllers só interagem diretamente com os repositórios apenas para encontrar um cidadão específico, ou os Tópicos.

REST

Componente REST sobre as bills utiliza os seguintes serviços:

- `ListAvailableVotesService`
- `ConsultNonExpiredBillService`
- `ProposeBillService`
- `SupportBillService`
- `VotingService`
- `DemoService`

Disponibilizando os seguintes endpoints, sempre devolvendo DTOs, exceto no caso de Topic uma vez que não tem nenhuma lógica de aplicação:

- `GET /bills/votable`: Que devolve uma lista de bills votáveis
- `GET /topics`: Devolve uma lista de tópicos
- `GET /bill/{id}`: Devolve uma lista de id especifico
- `GET /bills/open`: Todas os projetos ainda não expirados
- `POST /bills`: Para criar uma bill, recebendo no body a nova bill
- `POST /bill/support`: Para apoiar uma bill
- `GET /bill/{citizenId}/voted/{billId}`: Para perceber se um cidadão já votou ou não numa bil
- `POST /bill/vote`: Semelhante a suportar, mas com um voto boolean

Além disso à mais dois endpoints para criar dados para a demo

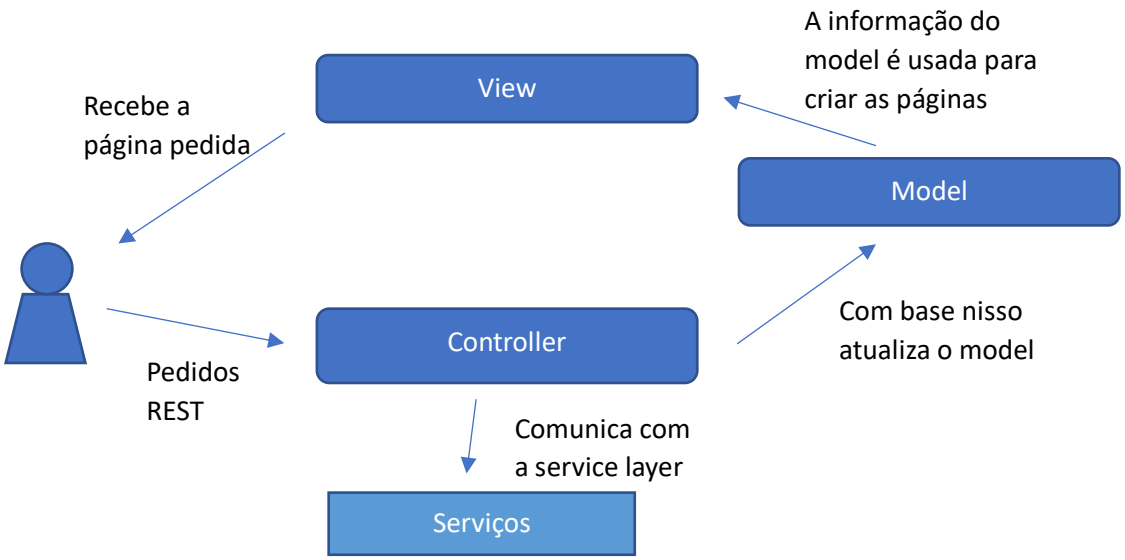
- `GET /test/votable/bills`: Gera projetos de lei votáveis para fins de teste.
- `GET /test/supportable/bills`: Gera projetos de lei suportáveis para fins de teste.

Depois existem endpoints específicos para o user, de forma a podermos indentificá-lo.

`GET /citizen/{name}`: Procurar um cidadão por nome.

Existem alguns endpoints que não foram pedidos como o criar uma bill.

WEB



A parte do thymeleaf funciona de forma semelhante, tendo controllers para a parte dos cidadões e das bills. O html é construído com um layout onde se substitui o conteúdo, pelo necessário, ou uma tabela com as bills ou uma bill específica ou um form para criar uma bill.

Como não há ainda forma de login apenas se pede o nome num form, e de seguida guarda-se na session o token de quem se ligou, que mais uma vez como não há login é apenas o id do cidadão.

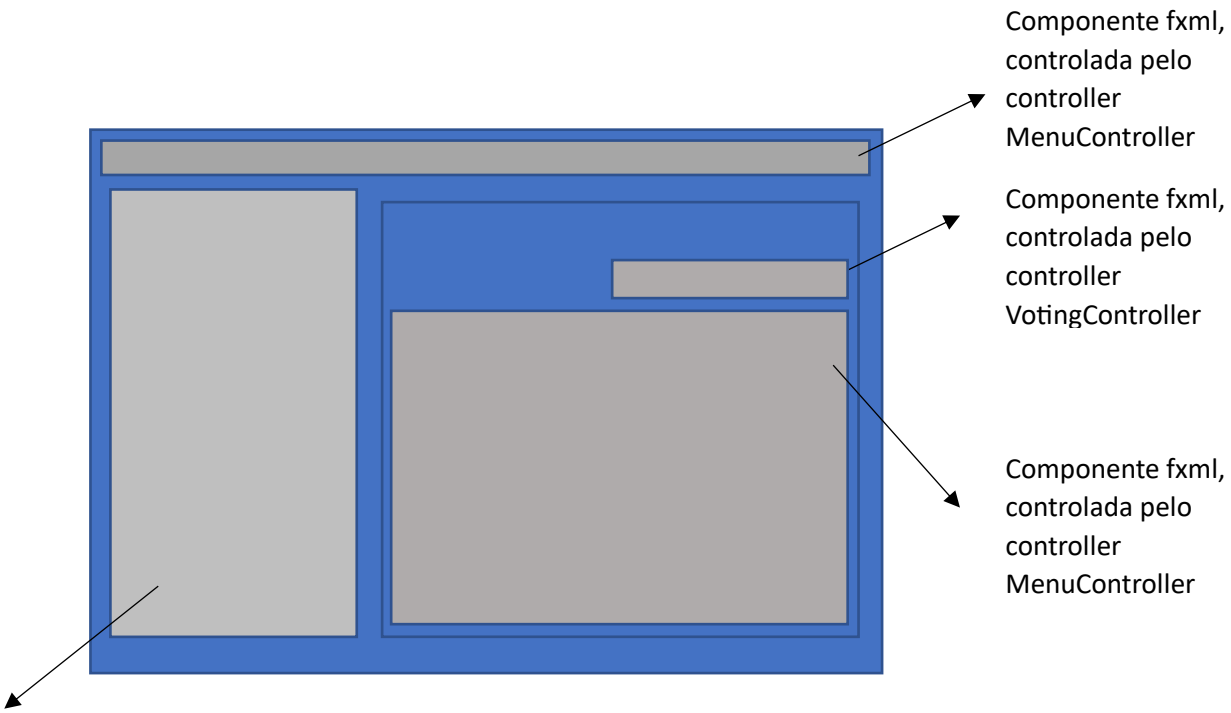
A parte da web utiliza os serviços diretamente, uma vez que o server thymeleaf corre no mesmo sítio que a base de dados.

Os endpoints são semelhantes aos do rest no entanto devolvem páginas em vez de DTOs, além disso há mais alguns endpoints para escolher um delegado,e autenticar-se (session).

App Nativa

A app comunica com o server através de pedidos REST, assim é essencial o server estar a correr com o run.sh, para que os dados possam ser mostrados corretamente. Assim bills criadas com a web aparecem na app Nativa.

JavaFX



A nossa app é essencialmente uma única página onde existem elementos controlados individualmente.
Para partilhar dados entre controllers, existe um data model, para interagir com eles além disso, esse model, interage com o server atravém de pedidos rest.

ListController: Pede ao data model a lista a mostrar, que pode ser bills abertas ou votáveis

Menu controller: Dispõe de várias opções para manipulara a lista a ser mostradas além disso, é aqui que podem ser geradas bills votáveis e abertas de exemplo, apenas para propósitos de demonstração.

Voting Controller: De acordo com a lista a ser mostrada e o cidadão com login, dispõe o botão correto, quer seja suportar ou votar. Além de mostrar qual o voto Omitido, ou se já votou.

Nota: Apesar de na app nativa não ter sido pedido para escolher um delegado, pode se escolher um através da web, assim mesmo na app nativa o voto omitido aparece.