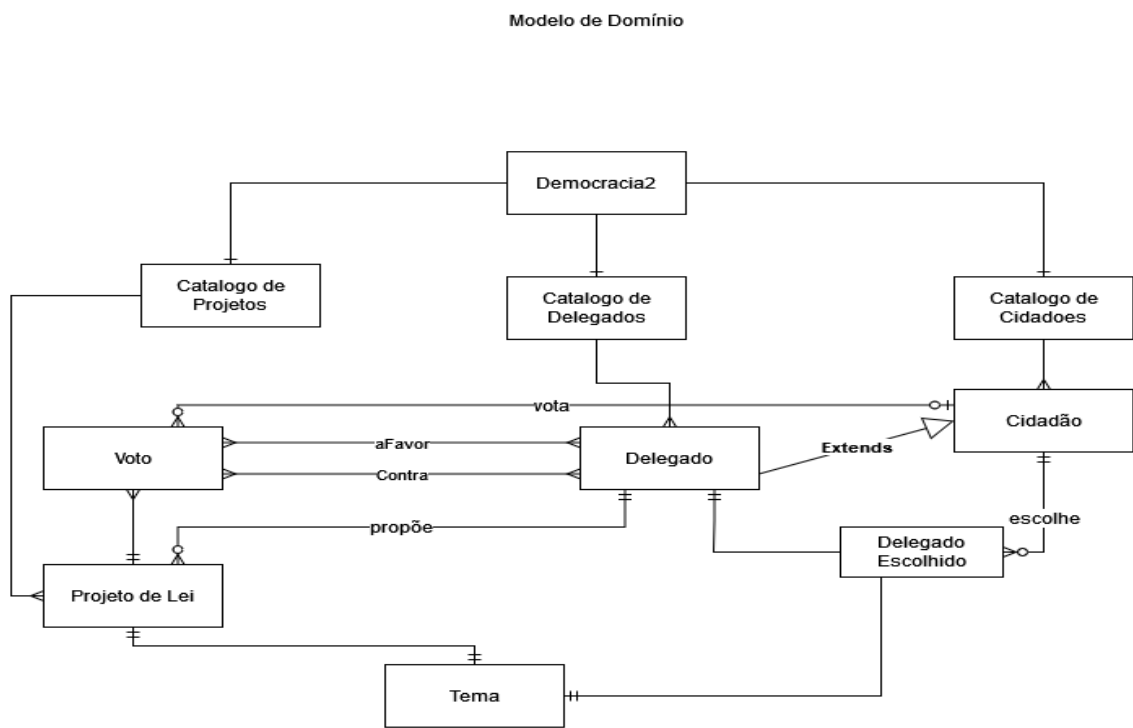
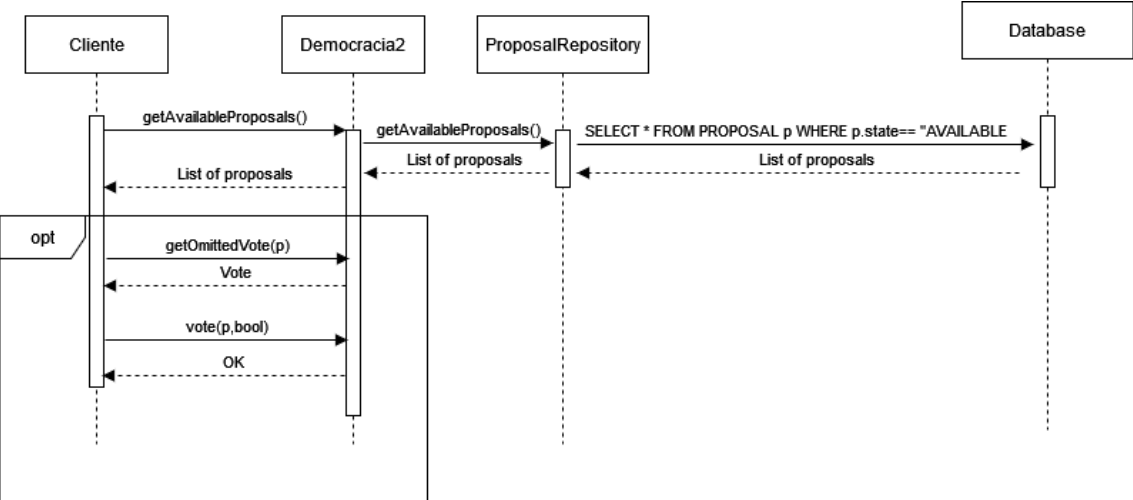


Modelo de domínio

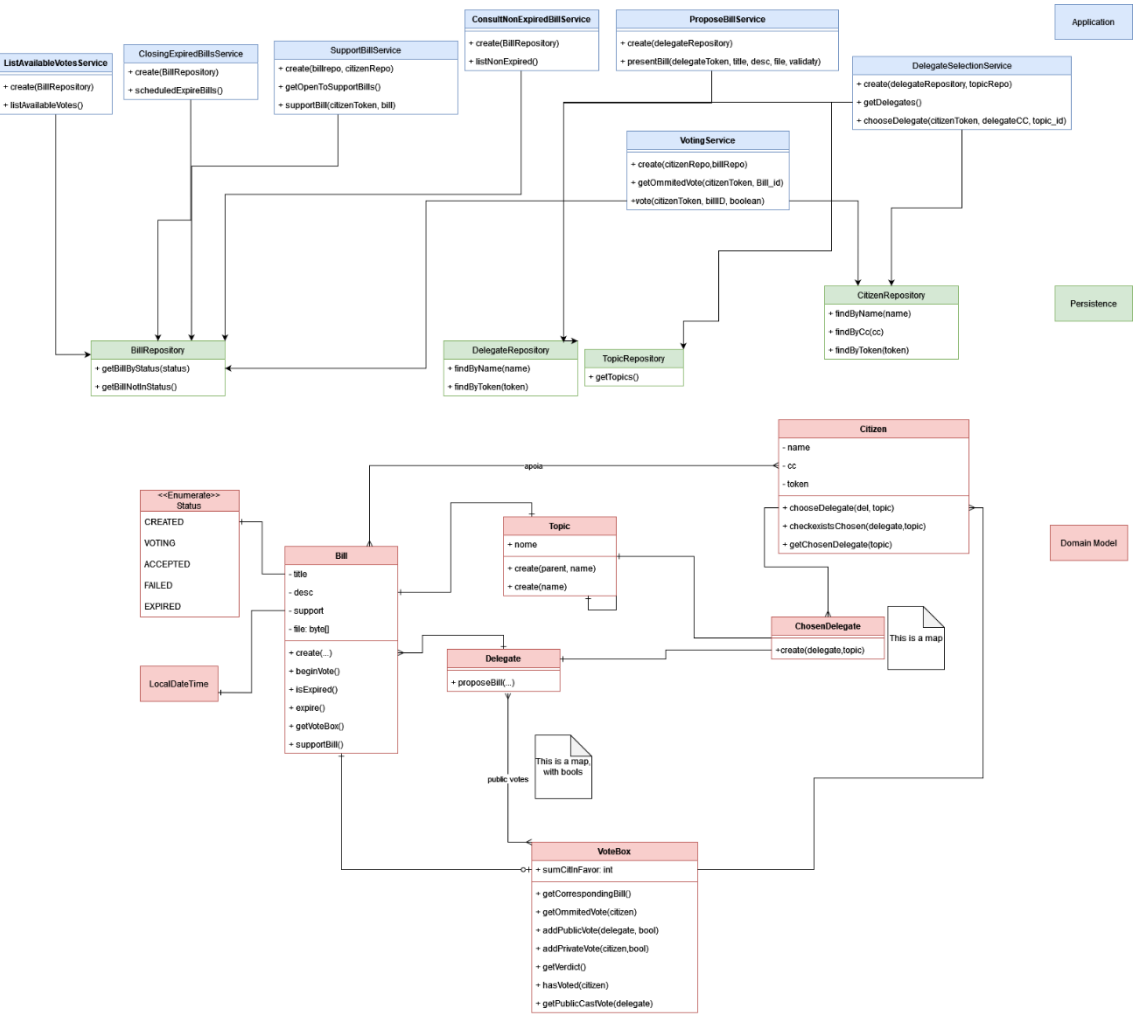


Versão 2

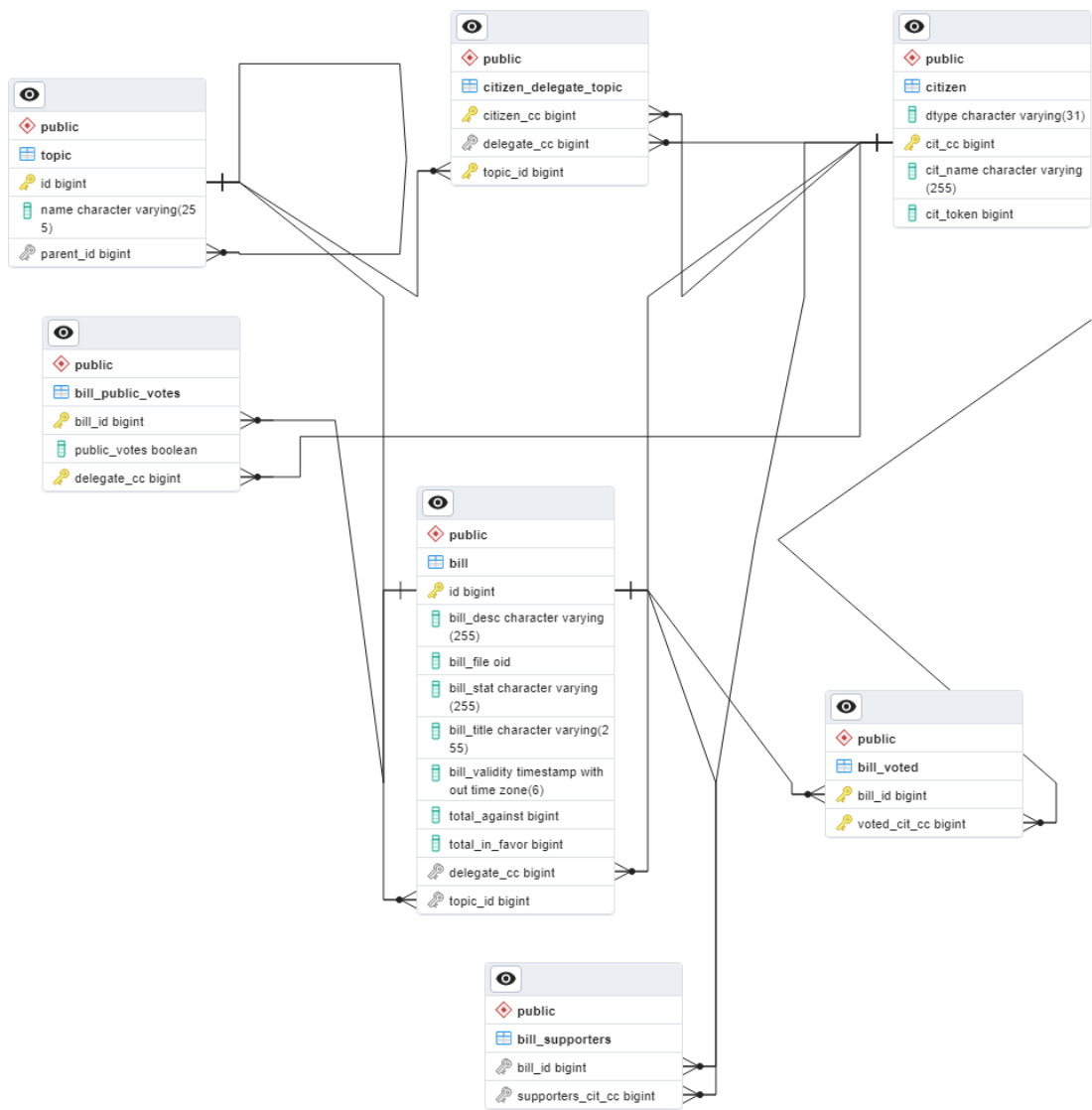
SSD caso de uso de J:



# 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.

```
@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.

```
@Entity
public class Delegate extends Citizen {

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

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

### Mapeamento especial no cidadão:

```
@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

```
@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

```
@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:

```
@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

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

Além disso inclui um embeddable:

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

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

A VoteBox tem este aspeto:

```
@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.