

Relatório do Projeto 1

Trabalho realizado por:

Diogo Pinto fc55179

Ivo Estrela fc51051

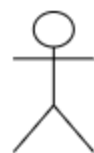
Thiago Duarte fc53636

Mapeamento usado no projeto (ORM)

Neste projeto tomámos as seguintes decisões nas anotações JPA para o mapeamento:

- O uso de **@ID** com a strategy **@GeneratedValue** para facilitar a criação de e inserção de identificadores na Base de Dados, ajudando também a consistência e persistência de informação para as Entidades já que este atributo fica imutável e sem significado lógico.
- O uso do **@Lob** para especificar que o atributo *fileData: byte []* da entidade Bill é mapeada para a Base de Dados em binário permitindo assim a persistência de ficheiros grandes como um PDF
- O uso **@Inheritance** com a strategy **TABLE_PER_CLASS** para mapear a herança entre o Citizen e o Delegado para haver distinção entre essas duas classes em que cada classe vai ter uma tabela com uma coluna por cada atributo herdado ou não. Isto implica o eficiente uso do espaço disponível e o acesso rápido aos dados o que é relevante para uma aplicação que poderá ter bastantes registos.
- Para as relações usamos várias anotações como **@OneToOne** no caso do Bill e do Poll sendo essa relação bidirecional especificado no Poll com um `mappedBy`, **@ManyToMany** como no caso das entidades DelegateTheme e Citizen (também bidirecional) e também **@OneToMany** como na relação Poll-Citizen/Bill-Citizen, sendo estas anotações implementadas para o mapeamento das relações numa tabela relacional. E para propagar as mudanças significativas nas entidades que tem associações usamos o **@Cascade(CascadeType.ALL)** para propagar todas essas mudanças à(s) entidade(s) associada(s).
- Uso de outras anotações mais triviais como o **@Entity** para as classes anotadas serem mapeadas numa tabela, **@Enumerated** para mapear enumerados...

SSD - Use Case J: Votar numa proposta



Voter

:Democracia2.0

getActivePolls()

lista dos bills em votação (polls)

opt

se não quiser ver voto do delegado não realiza opt

checkDelegateVote(poll,voter)

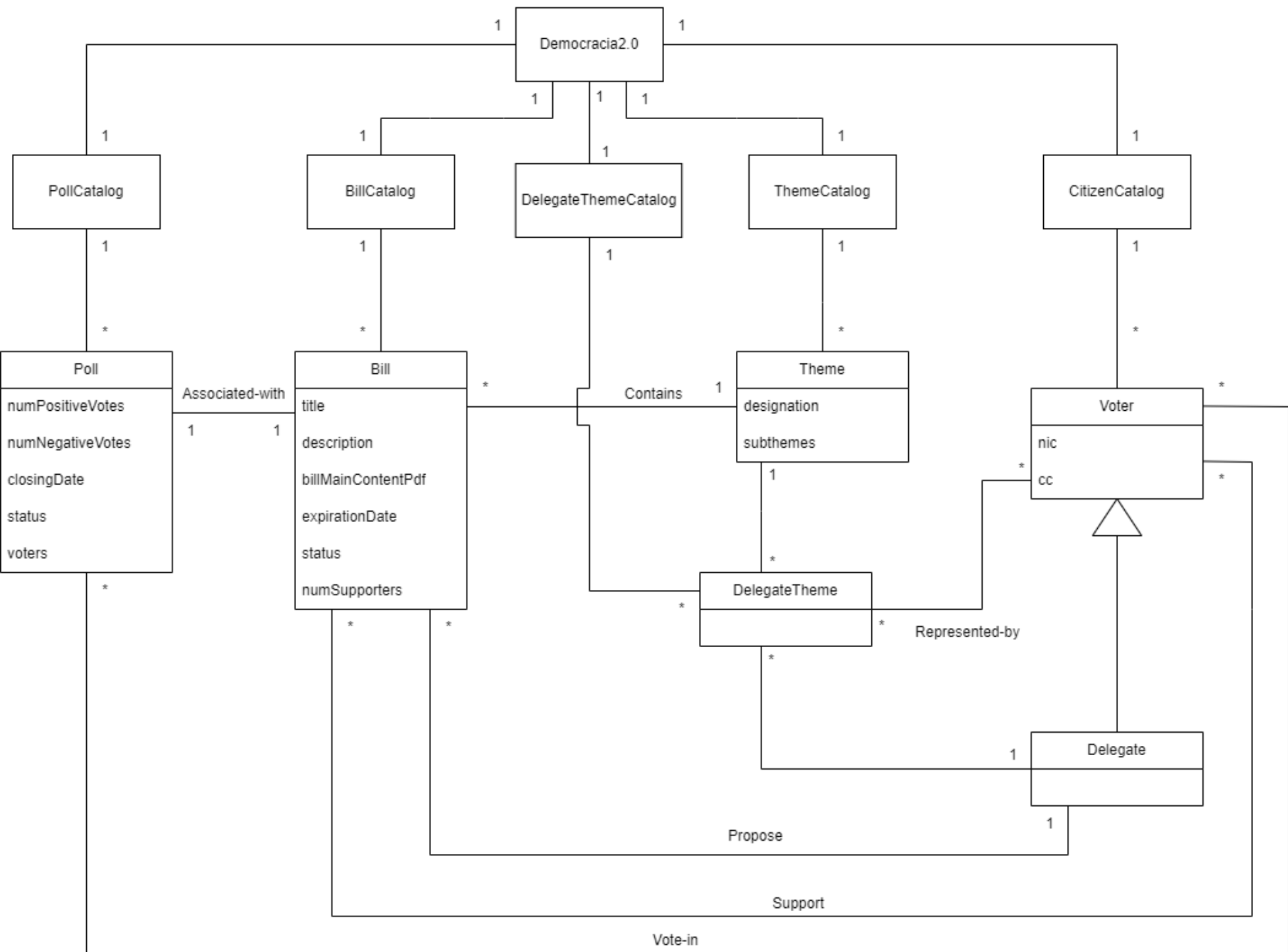
delegate vote

opt

se quiser votar

vote(poll, voter.id, option)

Domain Model



The diagram illustrates the architecture of a Democracy2 application, organized into five main layers: facade, dtos, services, business, and data.

- facade**: Contains `ApplicationException`.
- dtos**: Contains `ThemeDTO`, `PollDTO`, `BIIDTO`, and `DelegateDTO`.
- services**: Contains `Democracy2` (with `- log Logger`), `ListActivePolls Service`, `VoteOnProposedBill Service`, `SupportBills Service`, `ConsultBills Service`, `ProposeBill Service`, and `ChooseDelegate Service`.
- business**: Contains **handlers** and **data**.
 - handlers**: Includes `ListActivePollsHandler`, `CloseExpiredBillsHandler`, `ConsultBillsHandler`, `ClosePollsHandler`, `VoteActivePollsHandler`, `SupportBillsHandler`, `ProposeBillHandler`, and `ChooseDelegateHandler`.
 - data**: Includes `Poll`, `PollCatalog`, `Bill`, `BillCatalog`, `Theme`, `ThemeCatalog`, `DelegateTheme`, `DelegateThemeCatalog`, `Citizen`, `CitizenCatalog`, and `Delegate`.

Key relationships and associations include:

- `Democracy2` depends on all service classes.
- Service classes depend on their corresponding handler classes.
- Handler classes depend on the data objects they manipulate.
- Data objects are associated with their respective catalogs (e.g., `Poll` with `PollCatalog`, `Bill` with `BillCatalog`, `Theme` with `ThemeCatalog`, `DelegateTheme` with `DelegateThemeCatalog`, `Citizen` with `CitizenCatalog`).
- `Delegate` is a specialization of `Citizen`.