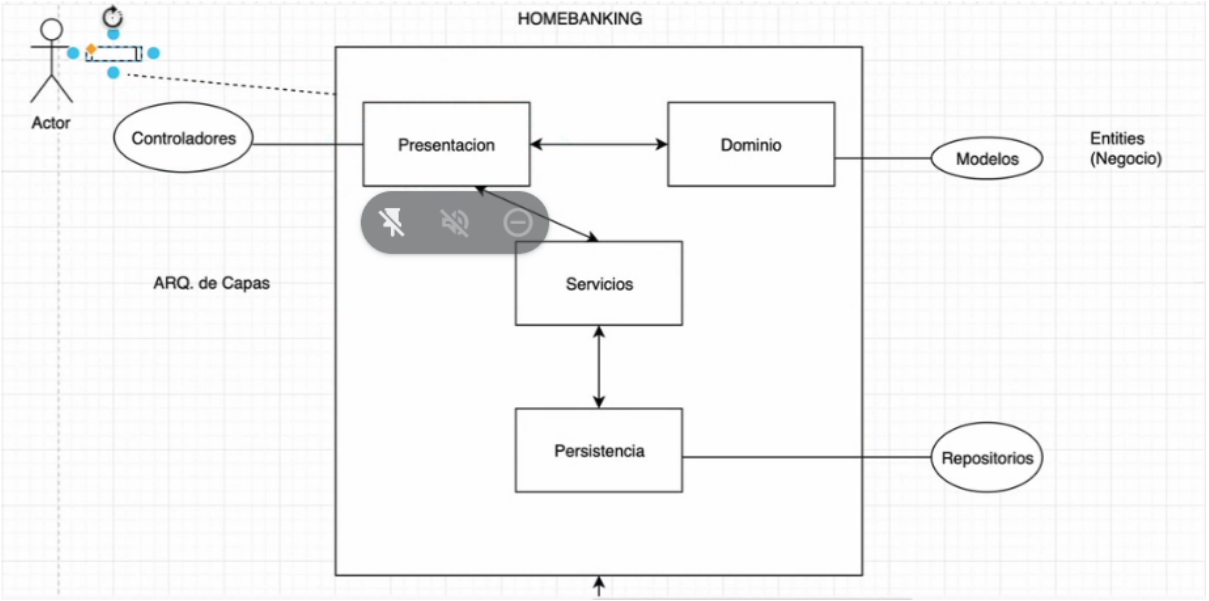
Proyecto Integrador Spring Boot

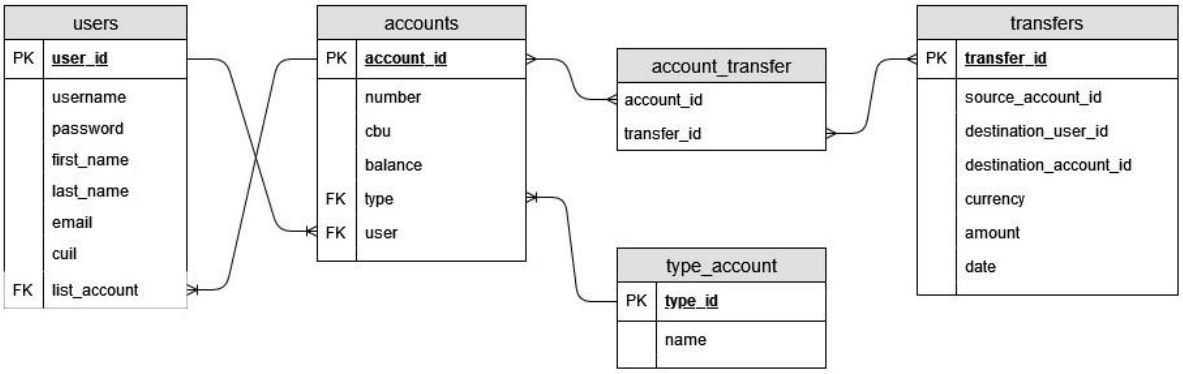
Docente: Cristian Gadea
Alumno: Juan Pablo Rigotti
Fecha: Diciembre 2022
Versión: 3.4.0

Introducción:
Se ha realizado un simulador de Homebanking usando Tecnología JAVA con el Framework Spring Boot y MySQL para la Database tomando como referencia el siguiente modelo N-Layers. Para ejecutar las consultas en la DB usamos el ORM Hibernate.



Entities
User
Account
TypeAccount
Transfer

Mapping
El mapeo de la Database se ha realizado según el siguiente esquema:



Relaciones

users-accounts (OneToMany)	
User	Account (parent - cascade)

`@OneToMany(mappedBy = "user")`
`private Set<Account> ListAccounts = new HashSet<>();`
	`@JsonIgnore`
	`@ManyToOne(fetch = FetchType.EAGER)`
	`@JoinColumn(name="user_id", referencedColumnName = "user_id")`
	`private User user;`

Un usuario puede tener varias cuentas (ca$, cc$, caU$S) Mientras que una cuenta solo puede pertenecer a un usuario La relación es bidireccional La entidad Account es el Parent que cascadea la persistencia en la entidad User (El user debe previamente existir, solo se asigna la nueva account al HashSet ListAccounts)	

accounts-type_account (ManyToOne)	
Account (parent - cascade)	TypeAccount

`@ManyToOne(fetch = FetchType.EAGER)`
`@JoinColumn(name="type_id")`
`private TypeAccount type;`
	n/a

Una Account puede ser de un solo tipo (TypeAccount), mientras que un TypeAccount puede pertenecer a más de una Account La relación no necesita ser bidireccional. La entidad Account es el Parent (owner de la relación)	

<table>
<tr><th colspan="2">accounts-transfers(ManyToMany)</th></tr>
<tr><td>Account</td><td>Transfer</td></tr>
<tr><td>

```
@ManyToMany(fetch = FetchType.EAGER,
cascade = CascadeType.ALL)
@JoinTable(name =
"account_transfer",
      joinColumns =
{@JoinColumn(name="account_id")},
      inverseJoinColumns =
{@JoinColumn(name="transfer_id")}
)
private Set<Transfer> ListTransfer =
new HashSet<>();
```

</td><td>

```
@JsonIgnore
@ManyToMany()
private Set<Account> ListAccounts =
new HashSet<>();
```

</td></tr>
<tr><td colspan="2">Una Account puede tener más de una transferencia asociada, y una transferencia, siempre pertenece a 2 accounts, la de origen y la de destino. Por lo tanto tenemos una relación ManyToMany
La relación es bidireccional (me parece que no puede ser Unidireccional en un ManyToMany), de Account a Transfer, para obtener todas las transferencias de una cuenta, que es la parte que más nos interesa.
El owner de la relación es la Entity Account (de manera de poder hacer el Get de las Transfers de una determinada Account) Para la persistencia, primero se crea la newTransfer, y luego esta se agrega al HashSet ListTransfer para persistir.</td></tr>
</table>

Accounts			Transfers
1		A1 - A2 - 100$ - T20	20
2		A2 - A4 - 50$ - T21	21
3			22
4			
5		A1 - A5 - 150$ - T22	

Models

Controllers
Endpoints

Services

Repositories

Resources
Static

Templates (fragments)

Pendientes:

1. Testing
2. users view:
    a. edit/delete users. Ver el tema de orphan data al remover datos de otras tablas
    b. ver de agregar además de Deposit, Transfer, History, una tabla de Moviments, de manera de ver los ingresos/egresos de capital a la Account, ya sea por Depósito o por Transferencia
    c. Agregar al User un atributo Boolean enable, de manera que si hay que borrar un usuario, lo único que se hace es deshabilitarlo (baja lógica)
3. accounts view:
    a. agregar validación al select (el required de html no funciona)
    b. consultar si la validación del delete por JS es correcta
4. Transfer entity:
    a. para los campos source/destination account, yo defini un account_id, pero debería usar un Objeto del tipo Account, eso hay que corregirlo
5. transfers view:
    a. a partir de la modificación del punto 4, reemplazar los IDs por datos mas representativos, por ejemplo, Account o CBU
    b. Quizás se pueda partir la tabla en 2, una mitad para transferencias salientes y otra para transferencias entrantes
6. moviments view:
    a. esta vista/tabla hay que generarla para tener un histórica como el de transferencias
7. Enviar correo electrónico cuando se realiza una transferencia
8. Agregar Login (con perfiles) de manera de limitar el acceso a los datos
9. Agregar validación de BackEnd
10. Agregar exceptions
11. Agregar/Documentar con Swagger
12. Subir código a Heroku
13. Usar Store Procedure para optimizar las búsquedas