Bank Tech Assignment: MoneyTransfer WebApplication

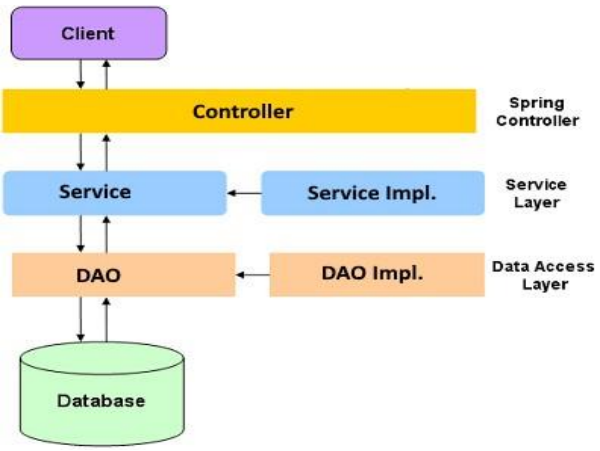
Name: **Lorenzo Gagliani**
GitHub: https://github.com/lorenzosax/moneytransfer
Date: **10/06/2019**
Installing and Deployment project: *described in the README.md file positioned at the root of the project*.

1- Project structure

The project is composed by one parent pom that include two modules:

- **moneytransfer-api**
- **moneytransfer-ui**

For moneytransfer-api, I decided to use the Java Spring Boot Framework version 2.1.9 RELEASE. This decision was made because this framework allows you to create a back-end layered structure (with Controllers, Services and DAOs) in a fast way.
For moneytransfer-ui, I decided to use Angular 8 javascript framework using Typescript language. Although it is a typed language, it does not provide static controls on data types and performs an implicit conversion between types (dynamic type-checking), so with Typescript we add support for static type checking and other features designed for writing complex applications. In this way we can maintain and scale the code more easily.



In the Controller layer, there are the REST APIs, and a client can make REST calls to the backend to prepare/verify/execute a money transfer. (You can find all APIs implemented in postman collection on the root of project).

Under that, in the Service layer, we can find the business logic, where all the calculations and logics happens.

At the end, from the service layer, we call the DAO/repository layer to perform the operation towards the db.
*I used a log4j2 to trace all log information on server. Each Log will have a **requestId** that will be returned in the response of APIs.*

2- Entity objects

In this project there are three kind of objects.

"**BankAccount**" object: this object represents a bank account and it's used to store all bank accounts of the bank's users (Customer).

List of fields:

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

@Column
@NotNull
private BigDecimal availableBalance;

@Column
@NotNull
private String currencyCode;

@Column
@NotNull
private BigDecimal transferLimit;

@Column
@NotNull
private String commissions;

@Column
@NotNull
private String accountNumber;

@Column
@NotNull
private String iban;

@NotNull
@ManyToOne
@JoinColumn(name = "customer_id")
private Customer customer;
```

"**Customer**" object: object that describe a single bank client, its id, its name and lastname (it's very basic information for this project).

List of fields:

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

@Column
@NotNull
private String name;

@Column
@NotNull
private String lastname;
@NotNull
@OneToMany(mappedBy = "customer", fetch = FetchType.EAGER)
private List<BankAccount> bankAccountList;
```

"**Transaction**" object: object used to store all money transfer executed from clients with all the detailed information.
List of fields:

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

@Column
@NotNull
private String trxId;

@Column
@NotNull
private Date createdAt = new Date();

@Column
private Date executedAt;

@Column
@NotNull
private BigDecimal amount;

@Column
@NotNull
private String currencyCode;

@Column
@NotNull
private String paymentReason;

@Column
@NotNull
private String accreditationDate;

@Column
private String cro;

@Column
@NotNull
private String beneficiaryName;

@Column
@NotNull
private String beneficiaryIban;

@NotNull
@OneToOne
private Customer customer;

@NotNull
@ManyToOne
@JoinColumn(name = "bank_account_id")
private BankAccount bankAccount;
```

The relationships between these entities are as follows:

- *BankAccount* has a **ManyToOne** relationship with *Customer* entity;
- *Transaction* has **OneToOne** relationshio with Customer and **ManyToOne** with *BankAccount.*

3- Assumptions

For this project I did an assumption: hardcoded on frontend application the user information of a generic user:

```
currentUserLogged: object = {
  customerId: '123',
  accountNumber: '0001234566'
};
```

This user information corresponds to an user on db which have:

Customer Obj: *id=1, name=Lorenzo, lastname=Gagliani, bank_account_id=1*

BankAccount Obj: *id=1, account_number=0001234566, available_balance=100000, commissions=0, currency_code=EUR, iban= IT34H7895966685880001234566, transfer_limit=10000, customer_id=123, customer_id=1*

(This information will be inserted on db at startup)

4- Business Logic Description

At service layer there are 3 services, to manage Customer, BankAccount and Transaction, so the services are: IBankAccountService, ICustomerService and ITransactionService.

The core service is ITransactionService (with implemented class called TransactionServiceImpl) that have this two methods:

```
@Transactional(propagation = Propagation.REQUIRES_NEW)
public Transaction generateTransactionFromVerify(CustomerBankAccount
customerBankAccount, BankTransferData bankTransferData);
```

```
@Transactional(propagation = Propagation.REQUIRES_NEW)
public Transaction updateAndTerminateTransaction(String trxId, Customer
customer, BankAccount bankAccount);
```

The first create a new transaction on Transaction entity with all basic information and the trxId, is called by CustomerServiceImpl when user do a verify request.

The second one:

- Update an existing transaction to execute Bonifico and generate CRO code that will be send to the client how confirm of the payment.
- Update the available balance of the user that want to do the Bonifico.

4- DAO/Repository Description

For the repository layer I used 3 interfaces, one for entity, that extends CrudRepository class of spring-data module, so there are all basic CRUD method for each entity.

For the Transaction repository, instead, I extends always CrudRepository class and add a new method for retrieve trx by trxId, Customer and BankAccount:

```
@Repository
public interface ITransactionRepository extends CrudRepository<Transaction,
Long> {
    public Optional<Transaction>
findTransactionByTrxIdAndCustomerAndBankAccount(String trxId, Customer
customerId, BankAccount bankAccount);
}
```

```
END OF DOCUMENT
```