# **Exercise: Smart Wallet**

*A* ***Smart Wallet Web App*** *is a digital platform that enables users to manage their finances, perform secure fund transfers, and monitor transactions conveniently from their browser. You will have to create an application which has several pages and core components.*

**

## **Model**

*This is the model layer of the application. There are some objects for you to implement first.*

### **User**

* **id** – an **UUID.**
* **username** – a **String,** the username of the user
* **firstName** – a **String,** the first name of the user
* **lastName** – a **String,** the last name of the user
* **profilePicture** – a **String,** URL containing link to picture of the user
* **email** – a **String,** email of the user
* **password** – a **String,** password of the user
* **role** – a **UserRole,** enumerated value(**ADMIN**, **USER**)
* **country** – a **Country,** enumerated value(BULGARIA, GERMANY, FRANCE)
* **isActive – a boolean** value which indicates whether the **User** is active
* **createdOn – LocalDateTime,** the date and time the **User** accountwas initialized
* **updatedOn – LocalDateTime,** the date and time the **User** account was updated
* **subscriptions** – a **List** of **Subscription** containing user's subscriptions
* **wallets** – a **List** of **Wallet** containing user's wallets

### **Wallet**

* **id** – an **UUID**
* **owner** – a **User,** the owner of the **Wallet**
* **status** – a **WalletStatus,** enumerated value(**ACTIVE, INACTIVE**)
* **balance** – a **BigDecimal,** the amount available in the wallet
* **currency** – a **Currency** of the amountin the wallet
* **createdOn – LocalDateTime,** the date and time the new **Wallet** was created
* **updatedOn – LocalDateTime,** the date and time the **Wallet** state was updated

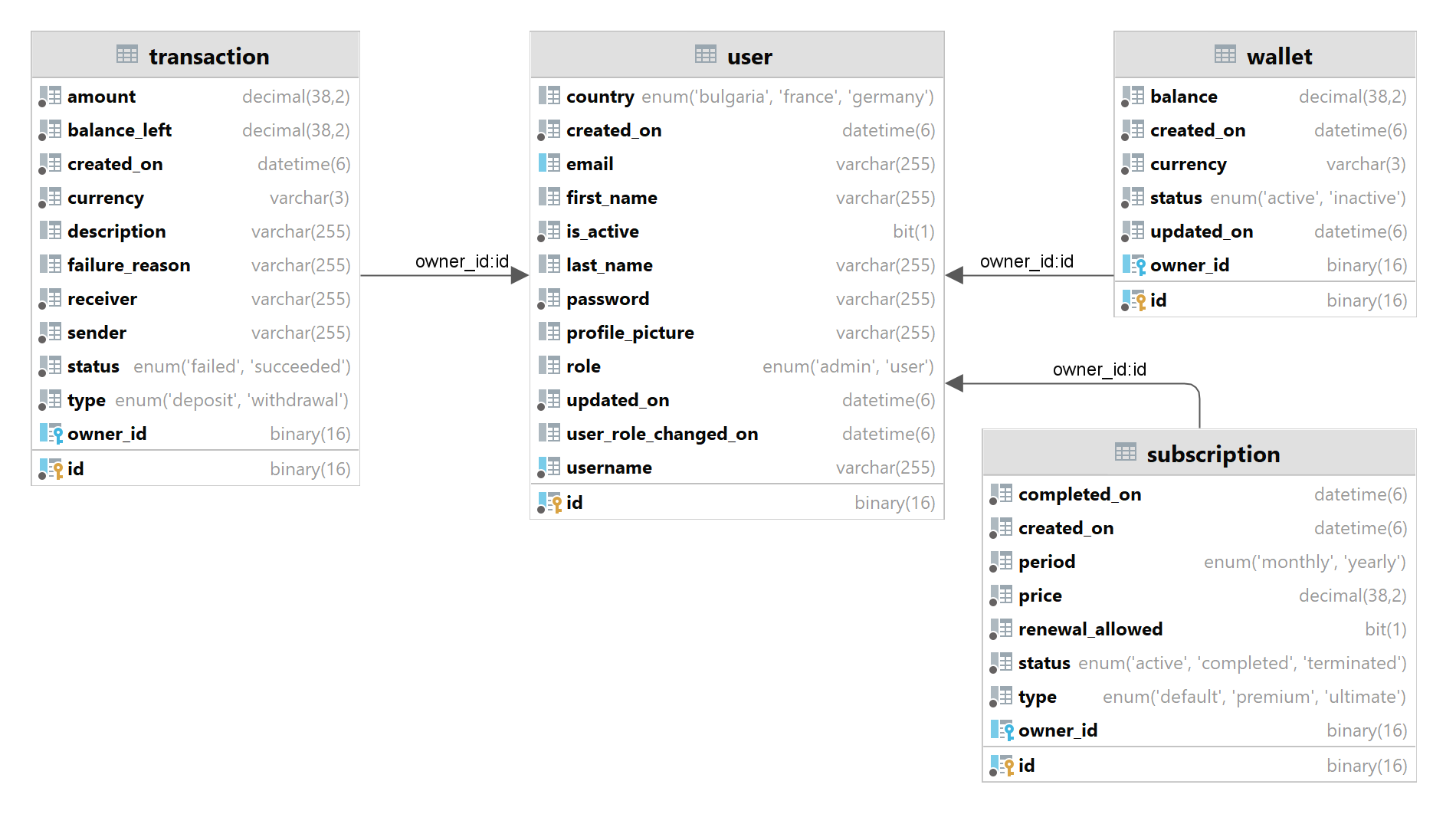
### **Subscription**

* **id** – an **UUID**
* **owner** – a **User,** the owner of the **Subscription**
* **status** – a **SubscriptionStatus,** enumerated value(**ACTIVE**, **COMPLETED**, **TERMINATED**)
* **period** – a **SubscriptionPeriod,** enumerated value(**MONTHLY**, **YEARLY**)
* **type** – a **SubscriptionType,** enumerated value(**DEFAULT**, **PREMIUM**, **ULTIMATE**)
* **price – a BigDecimal,** theprice of the subscription the user paid
* **renewalAllowed – a boolean** value which indicates whether the **Subscription** plan can be **automatically** renewed by our system - *if the user buys a* ***monthly subscription****, then the system will* ***automatically renew*** *their subscription when the time comes, if the subscription is* ***yearly****, there will be* ***no automatic renewal*** *for the subscription*
* **createdOn – LocalDateTime,** the date the **Subscription** was created
* **completedOn – LocalDateTime,** the date the **Subscription** was completed - *that could happen due to subscription change*

### **Transaction**

* **id** – an **UUID.**
* **owner** – a **User,** the user for which the **Transaction** was initiated for
* **sender** – a **String,** identifier of the wallet **from** which we take the money
* **receiver** – a **String,** identifier of the wallet **to** which we give the money
* **amount** – a **BigDecimal,** the amount of the **Transaction**
* **balanceLeft** – a **BigDecimal,** the remaining amount after the **Transaction**
* **currency** – a **Currency** usedfor the **Transaction**
* **type** – a **TransactionType,** enumerated value(**DEPOSIT**, **WITHDRAWAL**)
* **status** – a **TransactionStatus,** enumerated value(**SUCCEEDED**, **FAILED**)
* **description** – a **String,** description of the **Transaction**
* **failureReason** – a **String,** the reason for the failed **Transaction** - *in case the transaction can’t be executed for some reason*
* **createdOn – LocalDateTime,** the date the **Transaction** was made

**ER Diagram**



## **Data Access**

The application must support basic functionality involving **CRUD** (Create, Read, Update, Delete) operations. You must implement repositories for every entity to persist data

## **Business Logic**

### **Registering a New User**

When a new user registers, they need:

1. **Account Creation:** Validate the username to ensure its unique and store the user’s details securely. You must consider persisting user’s sensitive data in a secure way!
2. **Default Wallet Creation:** Automatically create a wallet for the user
3. **Default Subscription Setup:** Assign a free subscription to the user upon registration

**Skeleton for the Method:**



*The* ***register*** *method handles the entire flow. It takes a* ***RegisterRequest*** *object (with fields like* ***username****,* ***password****, and* ***country****) and returns a fully initialized* ***User*** *object.*

**Registration UI**

**A screenshot of a login form

Description automatically generated**

**Registration UI - validation constraint violated**

A screenshot of a login screen

Description automatically generated

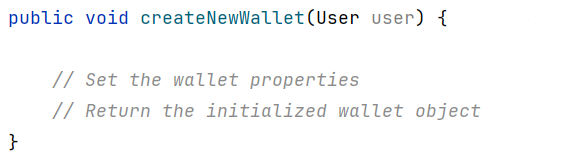
### **Wallet Creation**

Each user is assigned a wallet during registration. The wallet should be initialized with values defined in your configuration properties:

* **Initial Balance:** **€20**
* **Default Status:** **ACTIVE**
* **Currency: Euro (EUR)**

This ensures every user starts with a functional financial account that is ready to use.

**Skeleton for the Wallet Creation Logic:**

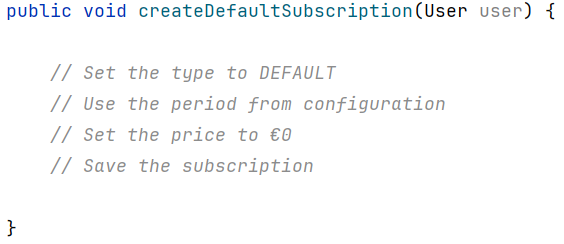
****

### **Subscription Creation**

Subscriptions provide access to app features. Every user starts with a **free default subscription**, configured as follows:

* **Type:** DEFAULT
* **Period:** From configuration (e.g., MONTHLY)
* **Price:** €0 (free by default)
* **Renewal Eligibility:** Renewals are allowed only for monthly subscriptions

**Skeleton for Subscription Creation Logic:**



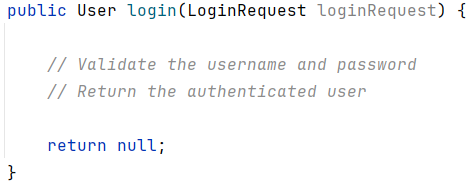
### **Login**

The login functionality validates a user’s credentials and ensures that only authorized users gain access

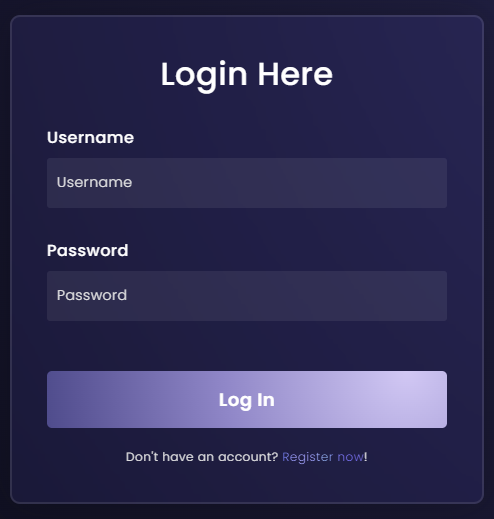
**Steps to Consider:**

* Verify the username exists
* Confirm the password matches securely
* Return the logged-in user

**Skeleton for the Login Method:**



**Login UI**

****A screenshot of a login screen

Description automatically generated

**After successful login:**

**A screenshot of a computer

Description automatically generated**

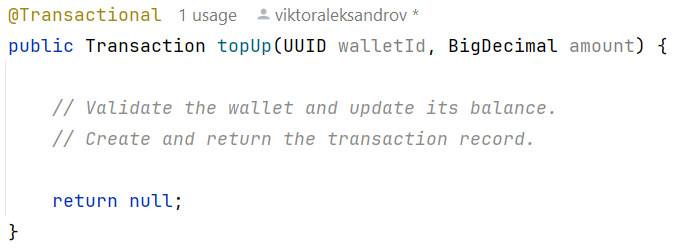
### **Top-Up: Adding Funds to Wallets**

Users can increase their wallet balance by performing a top-up operation.

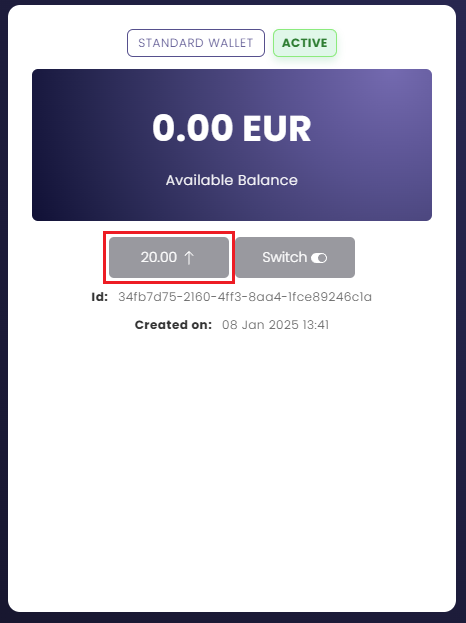
**Steps to Consider:**

* Retrieve the wallet using its ID the user wants to top-up
* Verify the wallet is **ACTIVE** before proceeding, **INACTIVE** wallets are not eligible for top-up
* Add the top-up amount to the wallet balance
* Record the transaction and save it for auditing purposes

**Skeleton for the Method:**



**Wallet before top-up:**



**Confirmation of successful top-up:**

A screenshot of a computer screen

Description automatically generated

**Wallet after top-up:**

A screenshot of a phone

Description automatically generated

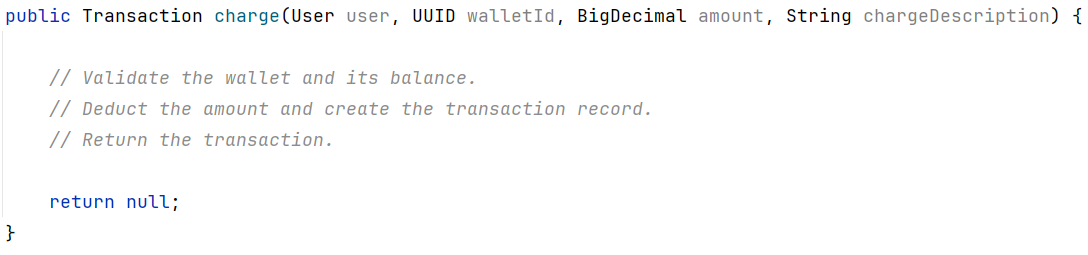
### **Charge: Deducting Funds from Wallets**

A charge represents an outgoing payment from a wallet

**Steps to Consider:**

* Retrieve the wallet using its ID
* Ensure the wallet is **ACTIVE** and has sufficient funds
* Deduct the specified amount from the wallet balance
* Record the transaction and save it

**Skeleton for the Method:**

****

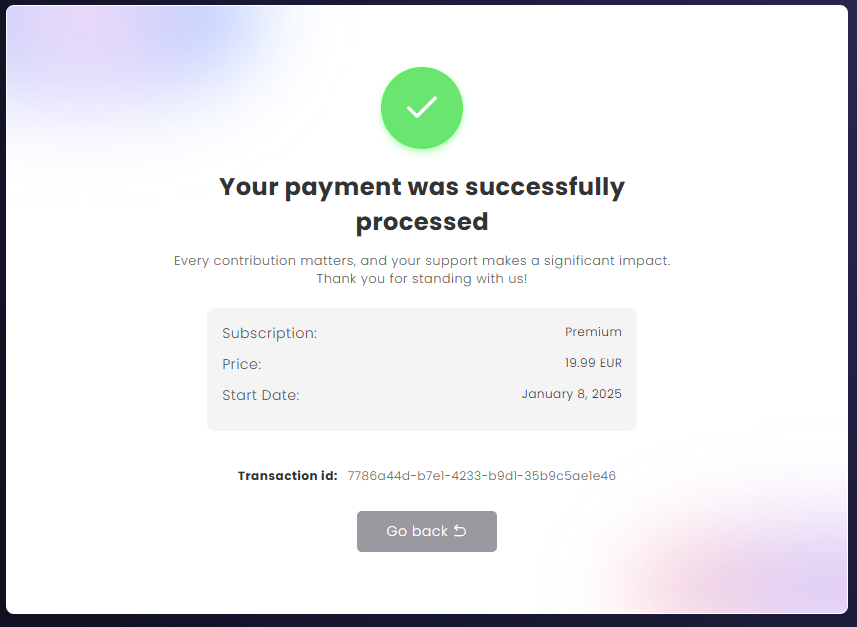
**NO MATTER THE CASE** - you need always to **return** a transaction for that charge even if it is a **failed** or **succeeded** transaction for tracking purposes.

**PREMIUM subscription charge:**

Screens screenshot of a screenshot of a website

Description automatically generated

**Confirmation of successful transaction:**



**Unsuccessful transaction**

