Telerik Academy

Virtual Wallet

Team Project Assignment

Virtual Wallet Project

Project Description

VIRTUAL WALLET is a web application that enables you to continently manage your budget. Every user can send and receive money (user to user) and put money in his **VIRTUAL WALLET** (bank to app). **VIRTUAL WALLET** has a core set of requirements that every team must implement and a variety of optional features.

Functional Requirements

Basic requirements

There are regular users and admin users. All users have username, email, password, photo, credit/debit card, phone number. A user can login/logout, change their password, email, photo, add and change their credit/debit card, make transfers to other users and view the history of their transfers.

- Users can view and edit their profile information, except their username, which is selected on registration and cannot be changed afterwards. The required fields for registration are username, email and phone number.

- Each user can register one credit or debit card, which is used to transfer money into their Virtual Wallet. The transfer is done by the provided REST API with separate documentation. **DO NOT use actual credit card information!**

- Users transfer money to other users by entering another user's phone number, username or email and desired amount to be transferred. Users can search by phone number, username or email in order to select the recipient user for the transfer.

- Users can view a paginated list of their transactions filtered by period in time, recipient and direction (incoming or outgoing) and sort them by amount and date.

- Admin users can see and search by phone number, username or email a paginated list of all users and block or unblock them. A blocked user, can do everything as normaly, except to make transactions.

- Admin users can view a paginated list of all user transactions filtered by period in time, sender, recipient and direction (incoming or outgoing) and sort them by amount and date.

In addition to the basic requirements, please choose from the optional requirements features of value of at least 5 points. If you have ideas for features, that are not on the list, please contact us with a short description of the feature, to receive our value estimation for it.

REST API

In order to collaborate with the QA team or provide other developers with your service, you need to develop a REST API. The **REST API** should leverage **HTTP** as a transport protocol and clear text **JSON** for the request and response payloads.

API documentation is the information that is required to successfully consume and integrate with an API. Use Swagger to document yours.

Front-end

Public Part

The public part of your application should be accessible without authentication. This includes the application start page, the user login and user registration form.

Private Part (regular users)

Registered users should have private part in the web application accessible after **successful login**.

Once logged in users have access to the following pages:

- View/Edit Profile, where users can view and edit their personal information.
- Add Funds, where the user can register a credit or debit card. A registered card can be removed or edited. Once a card is registered, the user has the option to add money to their Virtual Wallet. The user has to select the amount, which they want transferred into their Virtual Wallet. After the amount is selected the user can review the details of the transaction and choose to confirm it or go back to edit it.
- Crate Transactions Page, where the user can search the application's users by phone number, username or email. The user list should be pageable and should only expose the user's username and profile picture (you should not be able to find out all emails and phone numbers of all users with names starting with "a"). When the user selects the recipient, they are prompted to enter the transfer amount. Once user and amount are entered, the user can review the transaction details and confirm the transaction or go back to edit it. Blocked users do not have access to the page, instead they are show a message explaining that their account has been blocked and that they should contact the application support.
- Transaction History Page, where the user can enter a start and end date and optionally the recipient and/or direction of the transaction (incoming or outgoing). When submitting the search criteria, the user is shown a paginated list of their transactions, matching the criteria and has the option to sort them by date and amount.

Administration Part

System administrators should have two additional pages:

- Manage Users, which contains a paginated list of all users (username, email and phone number). The list can be filtered by username, email or phone number. Next to each user row, there is a button to block or unblock the user, based on their current status.

- Master Transaction History Page, which has the same functionality as the Transaction History Page but contains the transactions history of all users and has an additional filter by sender in addition to the one for recipient.

Technical Requirements

General development guidelines include, but are not limited to:

- Use **IntelliJ IDEA**
- Following **OOP** principles when coding
- Following **KISS, SOLID, DRY** principles when coding
- Following **REST API** design best practices when designing the REST API (see Apendix)
- Following **BDD** when writing tests
- You should implement sensible **Exception handling** and propagation
  - Every time you use System.out.println or e.printstacktrace() in a none-joking, serious manner, a kitten **dies**

Database

The data of the application MUST be stored in a relational database – **MySQL/MariaDB**. You need to identify the core domain objects and model their relationships accordingly.

Backend

- **JDK version 1.8**
- Use tiered project structure (separate the application **components in layers**)
- Use **SpringMVC** and **SpringBoot** framework
- Use **Hibernate/JPA (and/or Spring Data)** in the persistence (repository) layer
- Use **Spring Security** to handle user registration and user roles
- Service layer (i.e. “business” functionality) should have **at least 80% unit test coverage**

Frontend

- Use **Spring MVC Framework with Thymeleaf template engine** for generating the UI
- Use **AJAX** for making asynchronous requests to the server where you find it appropriate
- You may change the standard theme and modify it to apply own web design and visual styles. For example you could search and use some free html & css template to make your web application look good.
- You may use Bootstrap or Materialize

Note: Please keep in mind that your web application should be build using **Spring MVC Framework with Thymeleaf** template engine. The **REST API** is only for external usage of your services.

Optional Requirements

Each optional feature has assigned points relative to its difficulty. **Each team must implement features for at least 5 points.**

1. Email Verification (2 pt.) – In order for the registration to be completed, the user has to verify their email by clicking on a link send to their email by the application. Before verifying their email, users cannot make transactions.
2. Large Transaction Verification (2 pt.) – In order to complete transactions over a certain amount (up to you), the user is prompted to enter a verification code, sent to their email. The code should be unique for the transaction and have expiration time.
3. Refer a Friend (2 pt.) – A user can enter email of people, not yet registered for the application and invite them to register. The application sends to that email a registration link. If a registration from that email is completed and verified, both users receive a certain amount (up to you) in their virtual wallet. Invitations have an expiration time and a user can take advantage of that feature a limited number of times (up to you).
4. Identity Verification (2 pt.) – In order for the user registration to be completed, the user has to submit a photo of their id card and a selfie. Users with administrator rights should have a page where they can view all users waiting for verification, review the photos they submitted and approve or reject them. Before being approved, users cannot make transactions. **DO NOT upload actual photos of id cards!**
5. Joint Virtual Wallets (3 pt.) – User can create joint virtual wallets. They function as the regular wallets, however, multiple users can use them. The original creator of the wallet has an administration panel for the wallet, where they can grant or revoke other user's access to spend or add money to the wallet. When making a transaction or adding money to wallet, users with access to multiple wallets have to select, which one to use. The Transaction History Page show which wallet was used for the transaction.
6. Recurring Transactions (2 pt.) – Users can set up recurring transactions. When creating a transaction, the user has the option to select an interval of time on which the transaction is repeated automatically. Users have a page, where they can view all their recurring transactions and cancel them. Users have to be notified if their transactions failed for some reason.
7. Contacts List (2 pt.) – In addition to searching through all the application users, a user can create a contacts list. A user can add another user to their contacts list either from the transaction profile search or from the Transactions History Page. On the Create Transaction Page the user has to select if the transaction is from the contacts list or from the application users list. The user has a contact list administration page, where they can remove users from the list.
8. Multiple Virtual Wallets (2 pt.) – A user can create more than one wallet. When creating a transaction, the user is prompted to select, which wallet to use. The Transaction History Page show which wallet was used for the transaction. The user can set a default wallet, which is preselected when creating transactions.

9. Multiple Cards (1 pt.) – A user can register multiple credit and or debit cards, from which to add funds to their accounts. When adding funds to their wallet, the user is prompted to select from which bank account to do so.
10. Overdraft (3 pt.) – A user can enable overdrafts on their wallets. If overdraft is enabled, the user's Virtual Wallet balance can go below 0, up to a certain amount (up to you) when making transactions. At 00:00 on the first of each month wallets with enabled overdraft are charged interest. How you calculate the interest is up to you. If a user's total Virtual Wallet balance doesn't go above 0 for a number (up to you) of consecutive months, the user's account is blocked until enough money is added to it to cover the negative balance. Admin users have a page where they can change modifiers for the interest rate for future Savings Wallets (already existing ones should not be affected).
11. Savings Wallet (3 pt.) – User can create Virtual Savings Wallets. The user choses an amount and a duration for the wallet and is shown the interest rate they are going to receive. The user can review the Savings Wallet creation and confirm it or go back and edit the details. The interest rate should vary based on the duration and amount saved (how you calculate it is up to you). Once the selected duration for the account passes, the saved amount plus the interest is automatically added to the wallet, from which they were taken from. The user has a page where they can view their Savings Wallets.
12. Spending Categories (2 pt.) – When creating a transaction, a user can select a category for the transfer (Rent, Utilities, Eating out etc.). The user has a page to manage their categories. They can add, edit or delete them. The user also has a reports page where they can select a period of time and see a breakdown of their spending by category.
13. Currency Support (1 pt.) – When creating their Virtual Wallet users, can choose a currency for it. The currency exchange rate is shown on transactions between different currencies. The exchange rate and supported currencies are managed by admin users.
14. Easter eggs (1 pt.) – Creativity is always welcome and appreciated. Find a way to add something fun and/or interesting, maybe an Easter egg or two to you project to add some variety.
15. For the World to See (2 pt.) – Host your project online.

Deliverables

Provide link to a **Git** repository with the following information:

- Link to the **Trello** board
- Commits in the GitLab repository should give a good overview of how the project was developed, which features were created first etc. and the people who contributed. Contributions from all team members MUST be evident through the git commit history!
- Read https://dev.to/pavlosisaris/git-commits-an-effective-style-guide-2kkn and https://chris.beams.io/posts/git-commit/ for a guide to write good commit messages
- The repository MUST contain the complete application source code and any run scripts
- The project MUST have at least **README.md** documentation describing how to build and run the project
- Screenshots of the major application user facing screens with some data on them
- **URL** of the application (if hosted online)

Project Defense

Each team will have a **public defense** of their work to the trainers and students. It includes a live **demonstration** of the developed web application (please prepare sample data). Also, each team will have a defense of their project with the trainers where they must explain the application structure, major architectural components and selected source code pieces demonstrating the implementation of key features.

Expectations

You MUST understand the system you have created.

Any defects or incomplete functionality MUST be properly documented and secured.

It's OK if your application has flaws or is missing one or two MUST's. What's not OK is if you don't know what's working and what isn't and if you present an incomplete project as functional.

Do not use real data, phone numbers or card numbers for testing!

If you have and idea for an optional feature, that is not on the list - contact us to evaluate it

Some things you need to be able to explain during your project defense:

- What are the most important things you've learned while working on this project?
- What are the worst "hacks" in the project, or where do you think it needs improvement?
- What would you do differently if you were implementing the system again?

Appendix

Guidelines for designing good REST API

https://blog.florimondmanca.com/restful-api-design-13-best-practices-to-make-your-users-happy

Guidelines for URL encoding

http://www.talisman.org/~erlkonig/misc/lunatech%5Ewhat-every-webdev-must-know-about-url-encoding/

Always prefer constructor injection

https://www.vojtechruzicka.com/field-dependency-injection-considered-harmful/