**Project:** PowaFinance

**Group:** Functor

• Slobodeniuk Daniil - Backend developer

• Egorov Dmitry - Android Developer

• Dubinskiy Roman - Frontend developer

**Software Requirements Specification Document**

**Version: 2** **Date: (19/10/2022)**

1. **Introduction**

**Purpose**

Our purpose is to create an app that presents multiple features for the user to take care of his finances:

* add expenses per user
* form groups where users can add expenditure as well
* use third-party investment platform to pay bonds/stocks
* give ability to user to read most interesting new about financial topic

As a result we want to develop an app that not simply has expense keeper, but reaches much more wider scope.

**Scope**

Main goal is to push people in the direction of personal finances. According to the research made by us, lots of people don’t mind write down expenses or even dive deeper in the investment field, if it were in a user-friendly way. We want to provide such an opportunity.

However, we won’t implement investment platform on our own as it is of a huge burden. Also, we’ll use existing Google Pay/Apple Pay systems to ease up paying process when user wants to buy some asset on the aforementioned investment platform.

**Overview**

* In the second section we’ll go over **non-functional** requirements where certain requirements and SLA will be given
* In the third section we’ll show **functional** requirements which will be of a greater interest to developers.

**2. The Overall Description**

**Product Perspective**

Our system will be mostly a self-contained system which has some interactions with third-party systems, but only a few. In particular:

* *Investment platform* to: 1) allow user to buy/sell stocks/.bonds 2) to show desired info a finances assets
* *News agency* to be fetched to present the user top-notch news about finances

**System Interfaces**

Main interfaces to interact with:

* Investment platform: **/process-investment** is a coarse-grained API which is then fine-grained to get the desired result. For example, **/process-investment/observe-user-stocks/1583** - to see all the stocks/bonds of the user
* News agency: **/get-latest-news** - endpoint to get the news

**Interfaces**

* Interaction with *Investment platform* will be done via *endpoint* and their response will be verified via JSON schema
* Interaction with *News agency* will also be done via *endpoint* and same JSON validation will be applied.

**Hardware interfaces**

* In our app we won’t interact with any physical device in literal terms and our interaction with third-party systems will be down via plain API endpoints which abstracts us from need to think through such issues.
* Our app will use PostgreSQL for storing data. Definitely, we’ll use master-slave system with *hot-backup* and ordinary backup. Our db will be in different clusters to make it more fault-tolerant. Required version of PostgreSQL: 12+
* Liquibase is used for migrations purposes. We want to keep history of migration to make it available if some rollback to certain moment will be required. Required version of Liquibase: 4.0.0+

**Product Functions**

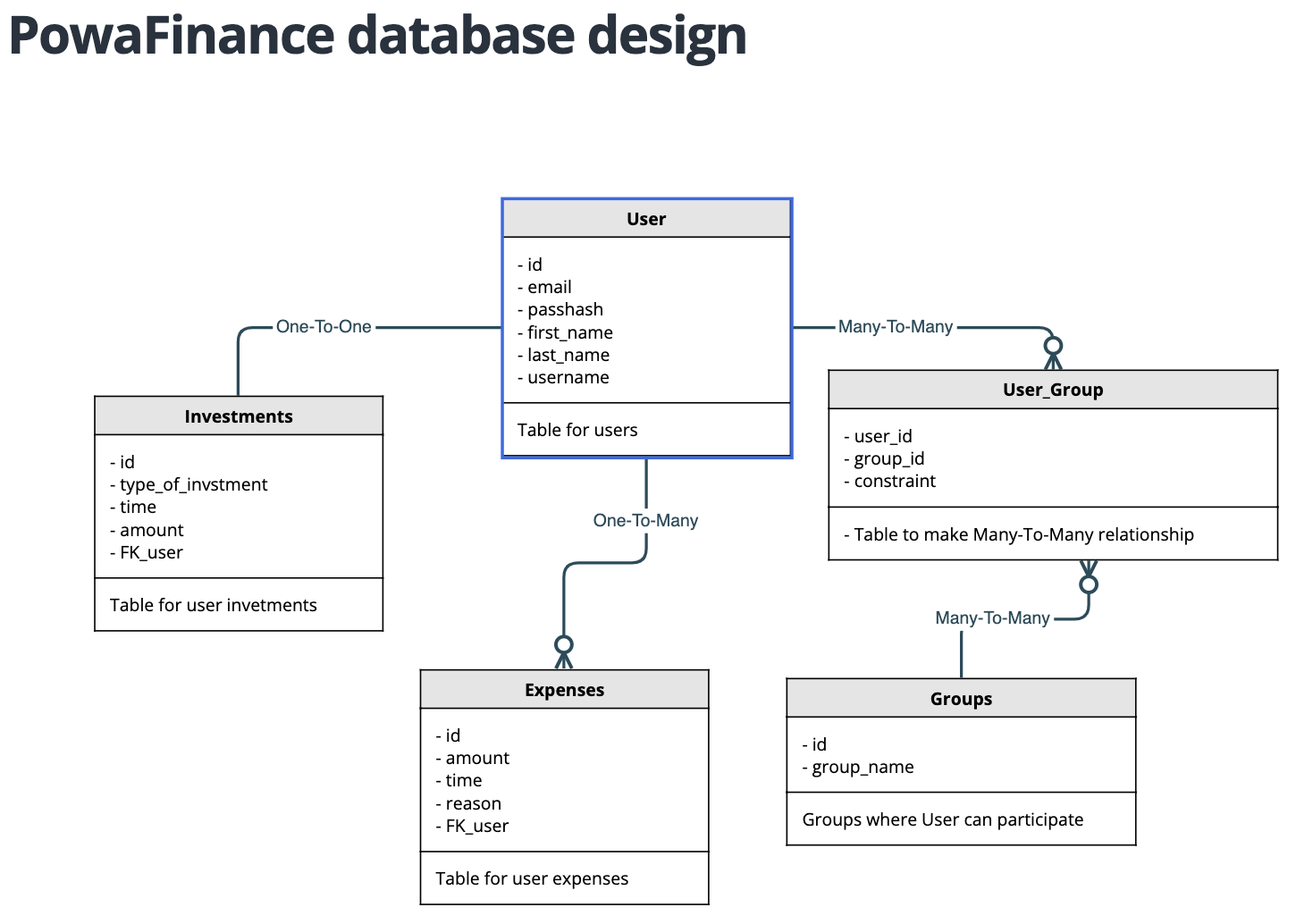
* System will accept users’ expenses and store them in the related database. UI will present buckets which can be filtered by days, weeks and other granularity.
* Special charts for users expenses per categories will appear in the dedicated menu.
* AI models to predict further expenses:
  + Traditional ML models will take history of expenses (at least 1 quarter of data is required) and project the expenses for the next month, next quarter.
    - Types of models: Boosting, Random Forest
* News of the most interesting issues on the financial arena will be fetched from the provided news agent. They will be presented in the dedicated section in the app
* Ability to invest via the investment platform is also put in the separate section in the app. User will see all the data fetched by the API of the investment platform.

**Constraints**

* **Regulatory limitations:** we’ll have access to users’ cards via Google Pay/Apple Pay. Hence we need to make payment process as clear as possible so as not to incur additional checks from the governmental part.
* **Control functions:** as we will leverage third party investment platform, then we need to constantly assess the actions of the platform and whether it doesn’t corrupt our app in any possible way.

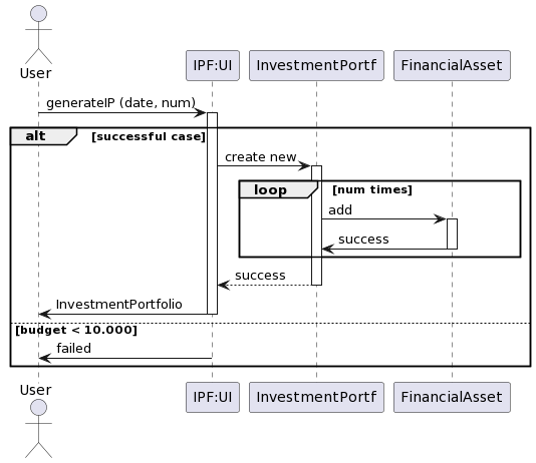
**3. Specific Requirements**

**Internal Interfaces**

* *Microservice* of the main app where user inputs data
* *Microservice* which serves as a wrapper for the database
  + Design of the database:

**External Interfaces**

Specific requirements for *Investment platform*:

* + Special requirements are:
    - Timeout: 15 seconds
    - Backoff: 60 seconds
    - Authentication by *mutual TLS*
  + Design of interaction with Investment platform:

Specific requirements for *News agency*:

* + Special requirements are:
    - Timeout: 10 seconds
    - Backoff: 45 seconds
    - No Authentication required

Content and format for *external interfaces*:

Investment platform:

* + **/process-investment/observe-user-stocks/UUID** 
    - input: user UUID is provided in the endpoint
    - output: JSON of all the stock & bonds
  + **/process-investment/observe-user-stocks/UUID**
    - input:{date: UNIX} in the *payload* to get the data of the particular user for the dedicated period
    - output: JSON of all the stocks & bonds

News agency:

* + **/fetch-news** - plain API to get the latest most interesting news

**Performance requirements**

* Database requirements:
  + standard PostgreSQL isolation is enough for our case
  + no 2PC or Saga is required as we don’t have multiple microservices with sequence of logical transactions
  + all frozen transactions that are not resolved within a minute will be rollbacked
* Interaction with our syste:
  + done via REST over HTTPS where mutual TLS is essential
  + ability to send bigger files is presented via RPC, precisely gRPC. But not a preferable way of communicating. Stubs with .proto are accessible

We will operate with *ZERO-TRUST* approach so additional checks will be all throughout our the system.

**Reliability**

In our case *MTBF* > *MTTR* as repair may take substantial amount of time. At least, expenses collector is a mission-critical part, whilst investment and news are business-critical.

**Availability**

**SLA:** 99.7 is essential as our users need to track their expenses

**SLOs:** 99.5 from *Investment platform* and *News agency*.

**Error budget:** depends on the current quarter SLO performance

**Security**

* Interaction with Investment is done via REST over HTTPS. Mutual TLS is way tom secure our traffic.
* Separate micro service with database will require mutual TLS as well.

### **Organizing the specific requirements**

Classes:

**Main app** microservice:

* + **UserInputParser**: to parse the input and understand what to do
    - List<Expense>
    - UUID of the user
  + **Expense**: instance of expense
    - Date
    - ExpenseName
    - Amount

**Database wrapper** microservice:

- **DatabaseWrapper**: class to deal with input of batch of **Expense**

**Investment platform** microservice:

- **InvestmentUnit**: one unit (i.e. stock or bonds) which has multiple properties

- Date

- UnitName

- Amount

**News Agency** microservice:

- **News**: instance which encompasses lots of news

- Name

- Category