MrPenn

SINGLE SOFTWARE DESIGN DESCRIPTION DOCUMENT

Revision 1 – 15/02/2020

Abbo Giulio Antonio

CONTENTS

[1 Introduction 3](#_Toc33220099)

[1.1 Purpose 3](#_Toc33220100)

[1.2 Overview 3](#_Toc33220101)

[1.2.1 Goals 3](#_Toc33220102)

[1.2.2 Requirements 3](#_Toc33220103)

[1.2.3 Assumptions 4](#_Toc33220104)

[1.2.4 Other relevant data from RSD 4](#_Toc33220105)

[1.3 Definitions 4](#_Toc33220106)

[1.4 Acronyms and abbreviations 4](#_Toc33220107)

[1.5 Revision history 4](#_Toc33220108)

[2 References 5](#_Toc33220109)

[3 Context 6](#_Toc33220110)

[3.1 Use case 6](#_Toc33220111)

[4 Composition 7](#_Toc33220112)

[4.1 Functional decomposition 7](#_Toc33220113)

[4.2 Run time decomposition 7](#_Toc33220114)

[5 Logical 8](#_Toc33220115)

[6 Interactions 8](#_Toc33220116)

[7 Patterns 8](#_Toc33220117)

[8 Algorithm 9](#_Toc33220118)

[9 Resources 9](#_Toc33220119)

[10 User Interface 9](#_Toc33220120)

[11 Implementation notes 9](#_Toc33220121)

# Introduction

## Purpose

The purpose of the MrPenn system is to allow its users to store their money transactions, then they will have access to their personal balance and some personal statistics or graphs based on this data. All the data is personal and confidential, thus is not shared between users.

## Overview

### Goals

* G1: The system allows users to privately store their transactions.
* G2: Each user can see, update and delete each of his transactions.
* G3: The user can request statistics based on his transactions.
  + Transaction that have not been returned.
  + Total balances for each entity.
  + Partial monthly balances for each entity and category.

### Requirements

* R1: The user is prompted to log in if necessary
* R2: The user can sign up to the system
* R3: The user can add a new transaction with:
  + Id
  + Positive amount
  + Origin and destination entities
  + Date
  + Optionally one or more categories
  + Optional notes
  + Whether must be returned
  + Optional returning id
* R4: The backend check that the new transaction is correct and stores it
* R5: For each transaction, the value is subtracted to the source entity
* R6: For each transaction, the value is added to the destination entity
* R7: For each transaction, the value is added to its categories
* R8: The data is only accessible by the user that created it
* R9: The user can see, update and delete each of his transactions
* R10: The user can specify preferred entities
* R11: The user can specify preferred categories
* R12: The user can add some of the preferred entities to a “total”, that is their sum
* R13: The “total” is shown with other entities in the statistics, but cannot exist in transactions
* R14: When an entity is set as preferred, the user is asked for its initial balance
* R15: Each category allows to specify if its value is positive or negative
* R16: The user can request the monthly partial balances for each entity
* R17: The user can request the monthly partial balances for each category
* R18: The user can request the up-to-this-month balances for each entity
* R19: The user can request the up-to-this-month balances for each category
* R20: The user can request the total balances for each entity
* R21: The user can see some statistics:
  + Transaction that have not been returned
  + Graph with preferred entities balances by month
  + Partial monthly balances for the current and previous month, for preferred entities
  + Partial monthly balances for the current and previous month, for preferred categories
  + Total balance for the preferred entities
  + Graph with the monthly spending for each category

### Assumptions

* D1: The system relies on the abstraction it is built on for the security of the connection.
* D2: The user has a device and the MrPenn app.
* D3: The user is connected to the internet most of the time.
* D4: The external services behave correctly.

### Other relevant data from RSD

* U1: User sign up.
* U2: User log in.
* U2: New transaction.
* U3: Request transactions.
* U4: Request statistic data.
* UI1: The UI adheres for the most part to the Material guidelines, especially for usability.
* UI2: A transaction that must be returned allows to open a precompiled form to insert the returning transaction.
* UI3: When adding a transaction, the user can specify which transaction is returning.
* O1: The data must be portable.
* O2: The back end is always backward-compatible (from the viewpoint of the front end).
* O3: The system will adhere to the principal GDPR guidelines.

## Definitions

* System: the software to be.
* User: an individual that uses the app.
* Transaction: an exchange of money from a certain origin to a certain destination.
* (Personal) balance: the sum of all the transactions for an entity or category.
* Entity: the source or destination of the money in a transaction.
* Category: labels for the transactions.
* To return: a sum of money is to be returned if it must be given back eventually.
* Initial balance: for an entity, the initial sum of money, used to produce statistics.
* Total: a special entity that is the sum (in the context) of other selected entities.
* Positive or negative category: whether the transactions in this category are incomes or outgoings.

## Acronyms and abbreviations

* Gn: goal number n.
* Rn: requirement number n.
* Dn: domain assumption number n.
* Un: use case number n.
* UIn: user interface requirement number n.
* On: other requirement number n.

## Revision history

* Version 1: first version.

# References

The structure of this document is inspired on those proposed in IEEE 1016.

Material Design by Google.

GDPR regulations.

# Context

This design document focuses on the front end, the data will be stored in an external back end, which will mainly implement three function: authorization, data storage and security checks.

Some actions are not considered here since they are not basic functions, for example password reset, closing the account, updating a transaction and similar.

## Use case

Figure 1 represents the use cases of the front end and their relationship with the backend functions. The red arrows represent an indirect communication with the back end, through other front endo components and the network.



Figure 1 – UML use case diagram for the front end.

Here follows a specification of each use case, the actors are not specified since there is only the user.

#### U1: User sign up

Precondition: the user has a device and wants to use the app.

Execution: the user inserts the required data and sends it.

Postcondition: if the operation was successful, the user is signed in; otherwise he receives an error message.

Exceptions: the operation can fail if the user is not connected to the internet or the sever is not reachable.

Notes: the completion of the sign-up process is left to the server, that stores the data and can send a confirmation email.

#### U2: User login

Precondition: the user has a device with the app and has already signed up.

Execution: the user inserts the data and sends it.

Postcondition: if the operation was successful, the user is signed in; otherwise he receives an error message.

Exceptions: the operation can fail if the user is not connected to the internet or the sever is not reachable.

#### U2: New transaction

Precondition: the user has signed in.

Execution: the user inserts the data and sends it.

Postcondition: the data is checked by the server and if correct it is stored. If not, the client receives an error message.

Exceptions: the operation can fail if the user is not connected to the internet or the sever is not reachable.

Notes: the server can perform additional operations when it receives the transaction. The data could be encrypted.

#### U3: Request transactions

Precondition: the user has signed in.

Execution: the user requests a list of the transaction he has inserted. The server sends it and the client displays it.

Postcondition: The user can see the requested transaction in detail and modify them.

Exceptions: the operation can fail if the user is not connected to the internet or the sever is not reachable.

Notes: when the user modifies or deletes a transaction, the server could do some operations on the stored data.

#### U4: Request statistic data

Precondition: the user has signed in.

Execution: the app requests some data to the server or uses the data previously cached to create the required statistics or graph.

Postcondition: the user can see the statistics or graph.

Exceptions: the operation can fail if the user is not connected to the internet or the sever is not reachable.

Notes: the client could keep some data cached.

# Composition

As already stated, the back end will be taken off the shelf; the client will use the corresponding package for the communication with the servers. The use of external dependencies in the client will be masked with adapters when it could make sense in the future to change them. Adapter will not be used when dealing with package that are the de facto standard. The user interface will be split in widgets.

## Functional decomposition

Figure 2 represents the generic component diagram for the front end, in red the connection with the backend. Note that the communication with the backend relies on the used dependencies and is not implemented directly, it could be bidirectional.

The widgets use the Back end Interface to access the data, the Back end Handler can use a Model Handler to cache locally the data and an Encryption Handler to encrypt the sensitive data before sending it to the server.



Figure 2 – UML component diagram

## Run time decomposition

There are no relevant aspects regarding run time decomposition.

Figure 3 shows that the front end application, specified in the component diagram in Figure 2, is deployed on a mobile device and communicates with the backend which is on an external server not managed by the system.



Figure 3 – UML deployment diagram

# Logical

Interfaces

Model

Database

# Interactions

Sequence diagram

# Patterns

Among the many used patterns, the only one worth mentioning is the caching approach.

Caching will be performed because of the nature of the backend implementation, which will imply certain costs related to the flow and storage of data; for this reason, it makes sense to reduce the requests to the server as much as possible. All the data will be cached except for the transactions.

On insertions, updates and deletes the data will be forwarded immediately to the logic that sends it to the back end, the cached data is invalidated. This is because the server can do some other operations and will send a notification of the updated files. On notification by the back end, the local cache is updated. On requests by the widget the system should return the cached data without querying the back end.

# Algorithm

The only relevant algorithm is the one used in the encryption of the sensitive data. This should be a symmetric encryption, where only the front end has the key, stored or inserted by the user, and the backend will never have access to it.

Encrypting the whole transaction makes it impossible to efficiently query the data in the future, so only parts of the files can be encoded, namely, not the labels but only the fields. This means that the domain would be very small (categories names, entities names, notes).

Symmetric encryption with a small domain can pose multiple problems, for this reason, this approach is rejected.

The solution proposed is dual: a key-value system for the names of entities and categories, and a symmetric encryption for the notes.

The symmetric encryption

# Resources

Resource utilization. UML OCL.

# User Interface

User flow diagrams.

# Implementation notes