Personal Finance Project Description

1 SCOPE

Aim: enhance Splitwise service functionality with expenses and income analysis.

Data source: data received from the Splitwise API, user input, currency exchange rates API.

The solution provides reports of expenses and income.

Functionality:
Accounting of income
Recalculation of transaction amounts in default currency (Euro)
Reporting in pdf format of expenses vs income, expenses per category
Textual user interface

Components:

settings.txt
[userID].sqlite
[date_time].pdf
sync.py
income.py
base_calc.py
reporting.py
personal_finance.py
unrec_transact.py

Database description:

SQLite database has the name [userID].sqlite where *userID* is the id of the current user in Splitwise (of Max in our example)

Users table:

userID (integer) - the id of the *User* from Splitwise
name (text) – first and last name of the user (e.g., Ann Mustermann)

Groups table:

groupID (integer) - the id of the *Group* from Splitwise
group (text) – the name of the *Group* (e.g., Business Trip)

Categories table:

categoryID (integer) - the id of the *Category* from Splitwise
category (text) – the name of the *Category* (e.g., Transportation)

Subcategories table:

subcategoryID (integer) - the id of the *Subcategory* from Splitwise
category (text) – the name of the *Category* (e.g., Transportation)

Transactions table:

transactionID (integer) – the id of *Expense* from Splitwise
date (dd.mm.yyyy) – the *Expense* date
groupID (integer) – the id of the *Group* to which *Expense* belongs
subcateogryID (integer) – the id of the *Subcategory* to which *Expense* belong
description (text) – the description of *Expense*
currency (text) – the currency code of Expense
repeatInterval – the repeat interval of expense
updated (dd.mm.yyyy hh:mm:ss) – the date time *Expense* was last updated in Splitwise

TransactionItems table:

itemID (int) – unique id generated for every *Item* by the program during the sync
transactionID (int) – the id of the *Expense* to which *Transaction item* relates
userID (int) – the id of the *User* to whom the *Transaction item* relates
amount (float) – share of the *User* in the *Expense*
baseAmount (float) – left empty/unchanged during the Splitwise Sync process. Share calculated in the default currency (Euro)

For the development purpose a Splitwise account of imaginary person Max Mustermann, his Mom and wife should be created by the team. Please provide the credentials in the materials of the first presentation.

2 TASKS

TASK 1: SPLITWISE SYNC

Developers build a solution that allows synchronizing data from Splitwise service to the local SQLite database using the Python SDK for Splitwise.

Input: *settings.txt*, *API*

Output: *settings.txt*
SQLite tables*:*
Transactions
TransactionItems
Users
Groups
Categories
Subcategories

Settings.txt is the plain txt file which stores the authorization information for Splitwise API (for Max in our case)

Algorithm 1 (Splitwise Sync process):

1. Read the *settings.txt*
2. If [userID].sqlite does not exist, create it with all the necessary tables
3. Update *Users, Groups, Categories, Subcategories tables* with the data from Splitwise
4. Insert rows related to new *Expenses* into *Transactions* and *TransactionItems* tables
5. Delete rows related to deleted *Expenses* from *Transactions* and *TransactionItems* tables
6. Update rows related to updated *Expenses in Transactions* and *TransactionItems*

Possible exceptions should be handled.

TASK 2: INCOME INPUT

Developers implement a mechanism of income data input into the SQLite database. The user can input the data by interacting with the textual interface.

Input: *user input*

Output: SQLite tables:
Categories
Subcategories
Transactions
TransactionItems

Algorithm 2 (Income Input process):

1. If row (100, Income) does not exist in Categories table, insert it
2. If rows (101, 100, Salary), (102, 100, Business), (103, 100, Gifts), (104, 100, Grants), (105, 100, Other) do not exist in Subcategories table, insert them
3. Get the input from the user regarding the category, date, amount of the income and add it to the database

Possible exceptions should be handled.

TASK 3: DEFAULT CURRENCY

Developers create a mechanism that calculates an equivalent of amount in Euro for each transaction. API https://fixer.io/ is used to get the historical currency exchange data.

Input: *settings.txt,* historical currency exchange data
SQLite tables:
Transactions
TransactionItems

Output: SQLite tables:
TransactionItems

Algorithm 3 (Recalculation in default currency process):

1. By scanning the SQLite database identify the necessary time and currencies scope
2. Get the currency historical rate data for the necessary scope using the API
3. Recalculate and insert the baseAmount for every record in TransactionItems table

Possible exceptions should be handled.

TASK 4: UNRECORDED TRANSACTIONS

Developers create a mechanism that calculates the amount that the user forgot to record in Splitwise and inserts this amount as an income or an expense into the DB.

Input: *settings.txt,* data retrieved from the Splitwise
SQLite tables:
Transactions
TransactionItems

Output: *Splitwise remote data*
SQLite tables:
Transactions
TransactionItems

Unrecorded amount is calculated as:

= Incomes - Expenses + Net Debt - Fact Balance

where:

Incomes = sum of baseAmount for all records in *TransactionItems* table which are related to the *Transactions* of Category Income
Expenses = sum of baseAmount for all records in *TransactionItems* table which are not related to the *Transactions* of Category Income
Net Debt = user owes – user owed (these data is retrieved from the Splitwise via API)
Fact Balance is the current balance on user's accounts, inputted manually by the user in default currency.

Unrecorded Transactions process is defined in *unrec_transact.py.*

Example:

Incomes:	3650
Expenses:	1288
Net Debt:	661-23 = 638
Fact Balance:	16000
Unrecorded amount:	3650 – 1288 + 638 – 16000
=>	13000 of unrecorded Income

TASK 5: PREDICTION

Developers create a mechanism to predict user's balance for the next year based on the (repeating) expenses, income for the recent months as well as current balance. Prediction is saved as the plot in .pdf.

Input: SQLite database

Output: [date_time]_prediction.pdf

Prediction process is defined in *unrec_transact.py* and starts after Unrecorded Transactions process.

TASK 6: REPORTING

Developers implement a routine of aggregating the data from the database, visualizing it and saving as the [date_time].pdf file.

Input: SQLite database

Output: [date_time].pdf with figures

These figures should be included into the pdf:

pie charts of expenses per subcategory for the last 3 months (one chart per month);
column chart showing income vs expenses for the last 3 months.

TASK 7: USER INTERFACE

Developers integrate all solutions and provide the basic console interface for interaction with the application. Upon start the program runs the Splitwise Sync (Task 1) process followed by the Default Currency process (Task 3). After that programs offers the user actions to choose.

The program should offer input values for the user where possible, e.g. income subcategories.

User Interface process is defined in *personal_finance.py* and starts by running the file.