Group 5: Personal Finance

Tra My Sitz, Oleksandr Kotkov, Ovidio Herrera, Robinson Quiroz

STRUCTURE

1. Task definition
2. Functionality & Components
   2.1. Splitwise Integration & Transaction input
   2.2. Currency Conversion (€) & Income Accountability
   2.3. Prediction & Reporting
   2.4. Graphic User Interface
3. Limitations and Improvements

1

Task Definition

TASK DEFINITION

Aim: enhance Splitwise service functionality.

Data source:

- data received from the Splitwise API, user input, currency exchange rates API.
- The solution provides reports of expenses and income
- For the development purpose a Splitwise account of imaginary person Max Mustermann, his Mom and wife were created

2.1

Splitwise Integration and Transaction input

2.1.1 Splitwise integration

Establish connection with Splitwise and .sqlite database

```
def splitwise_sync(s_obj: Splitwise):
    user = s_obj.getCurrentUser()
    user_id = user.getId()
    conn = None
    cursor = None
    try:
        conn = sqlite3.connect(str(user_id) + ".sqlite")
        cursor = conn.cursor()
    except Error as e:
        print(e)
```

2.1.1 Splitwise integration

Create tables in database

```
def create_tables(cursor):
    cursor.execute("CREATE TABLE IF NOT EXISTS Users("
                   "id integer PRIMARY KEY,"
                   "full_name text,"
                   "email text)")
    cursor.execute("CREATE TABLE IF NOT EXISTS Groups_users("
                   "id integer PRIMARY KEY,"
                   "group_name text,"
                   "group_type text)")
```

2.1.1 Splitwise integration

Fill tables in database

```
def fill_tables(s_obj: Splitwise, cursor):
   user = s_obj.getCurrentUser()
   # Insert information about current user in Users table
   cursor.execute("INSERT OR IGNORE INTO Users (id,full_name,email) VALUES (?,?,?)",
                  [user.getId(), user.getFirstName() + " " + user.getLastName(),   user.getEmail()])
   # Get list of friends of a current user, insert information about them in Users table
   other_users = s_obj.getFriends()
   for u in other_users:
       cursor.execute("INSERT OR IGNORE INTO Users (id,full_name,email) VALUES (?,?,?)",
                      [u.getId(), u.getFirstName() + " " + u.getLastName(), u.getEmail()])
# Insert information about groups in Groups_users table
   groups = s_obj.getGroups()
   for g in groups:
       cursor.execute("INSERT OR IGNORE INTO Groups_users (id,group_name,group_type) VALUES (?,?,?)",
                      [g.getId(), g.getName(), g.getType()])
```

2.1.2 Transaction input

```
def insert_category(conn, cursor, category_id: int, category_name: str):
   try:
       cursor.execute("INSERT INTO Categories (id,category_name) VALUES (?,?)",
                  [category_id, category_name])
       conn.commit()
   except sqlite3.IntegrityError as err:
       if str(err) == "UNIQUE constraint failed: Categories.id":
           result = cursor.execute("SELECT * FROM Categories  WHERE id = (?)",
                          [category_id])
           print("Error - a category with id " + str(category_id) + " already exists: " +
str(result.fetchone()))
       if str(err) == "UNIQUE constraint failed: Categories.category_name":
           result = cursor.execute("SELECT * FROM Categories  WHERE category_name = (?)",
                                  [category_name])
           print("Error - a category with name " + str(category_name) + " already exists: " +
str(result.fetchone()))
```

2.1.2 Transaction input

```
def insert_subcategory(conn, cursor, subcategory_id: int, category_id: int, subcategory_name: str):
   try:
       cursor.execute("INSERT INTO Subcategories (id,category_id,subcategory_name) VALUES (?,?,?)",
                      [subcategory_id, category_id, subcategory_name])
       conn.commit()
   except sqlite3.IntegrityError as err:
       if str(err) == "UNIQUE constraint failed: Subcategories.id":
           result = cursor.execute("SELECT * FROM Subcategories  WHERE id = (?)",
                                   [subcategory_id])
           print("Error - a subcategory with id " + str(subcategory_id) + " already exists: " +
str(result.fetchone()))
       if str(err) == "UNIQUE constraint failed: Subcategories.subcategory_name":
           result = cursor.execute("SELECT * FROM Subcategories  WHERE subcategory_name = (?)",
                                   [subcategory_name])
           print("Error - a subcategory with name " + str(subcategory_name) + " already exists: " +
str(result.fetchone()))
 if str(err) == "FOREIGN KEY constraint failed":
           print("Error - a category with id " + str(category_id) + " does not exist.")
```

2.1.2 Transaction input

```
def insert_transaction(conn, cursor, transaction_date: str, group_id: int, subcategory_id: int,
                       description: str, currency_code: str, repeat_interval: str):
   try:
       cursor.execute("INSERT INTO Transactions (expense_date,group_id,subcategory_id,"
                      "description,currency_code,repeat_interval) VALUES (?,?,?,?,?,?)",
                      [transaction_date, group_id, subcategory_id,
description,currency_code,repeat_interval])
       conn.commit()
   except sqlite3.IntegrityError as err:
       if str(err) == "FOREIGN KEY constraint failed":
           print("Error - such group id or subcategory id  does not exist.")
       else:
           print(err)
```

2.1.2 Transaction input

```
def insert_transaction_item(conn, cursor, transaction_id: int, user_id: int, amount: int):
   try:
       cursor.execute("INSERT INTO TransactionItems (transaction_id, user_id, amount) VALUES (?,?,?)",
                      [transaction_id, user_id, amount])
       conn.commit()
   except sqlite3.IntegrityError as err:
       if str(err) == "FOREIGN KEY constraint failed":
           print("Error - such transaction id or user id  does not exist.")
       else:
           print(err)
```

2.2

Currency Conversion and Accountability

2.2.1 Currency Conversion

sql_currency.py

Converts Splitwise Transactions recorded in foreign currency to Euros (€)

```
def  currency(s_obj: Splitwise, settings: dict):
    df_sql = symbol_date_sqldf(s_obj)
    status = fixer_api_latest(settings)
```

2.2.1 Currency Conversion

```
fixer_api_latest()

API today's currency rates -> local json file

if API connection error -> read last saved json file


prevents application interruptions
```

2.2.1 Currency Conversion

APILayer

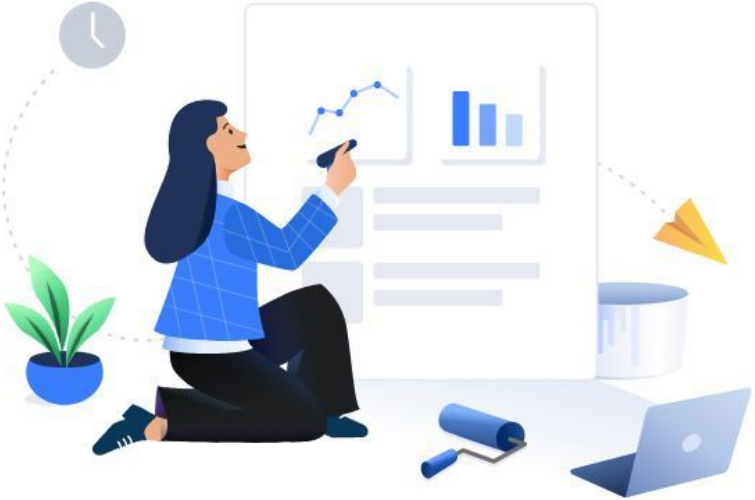
429 Too Many Requests (RFC 6585)

The user has sent too many requests

```
In [17]: get_url
Out[17]: <Response [521]>
```

521 Web Server Is Down

The origin server refused connections



We're just tuning up a few things.

We apologize for the inconvenience but APILayer is currently undergoing planned maintenance. Stay tuned!

2.2.1 Currency Conversion

symbol_date_sqldf()

transactions df (pandas) from local sql (Splitwise)

currency()

converts df foreign currency transactions to €

writes these values in sql database

2.2.2 Accountability

unrect_transac.py

- Allows user to balance his accounts
- Requests from Splitwise debt records for user’s friends in any currency such debt is recorded

2.2.2 Accountability

```
unrec_trans <, =, > income - expense + owes - owed

unrect_trans = 0 -> income + owes = expenses + owed

unrect_trans <[>] 0 -> PseudoIncome <[>] PseudoExpenses
```

2.2.2 unrec transactions and fact balance

```
unrec_trans = income - expense + owes - owed -(-fact_bal)
```

it balances to zero when

```
unrect_trans = -(-fact_bal) = fact_bal
```

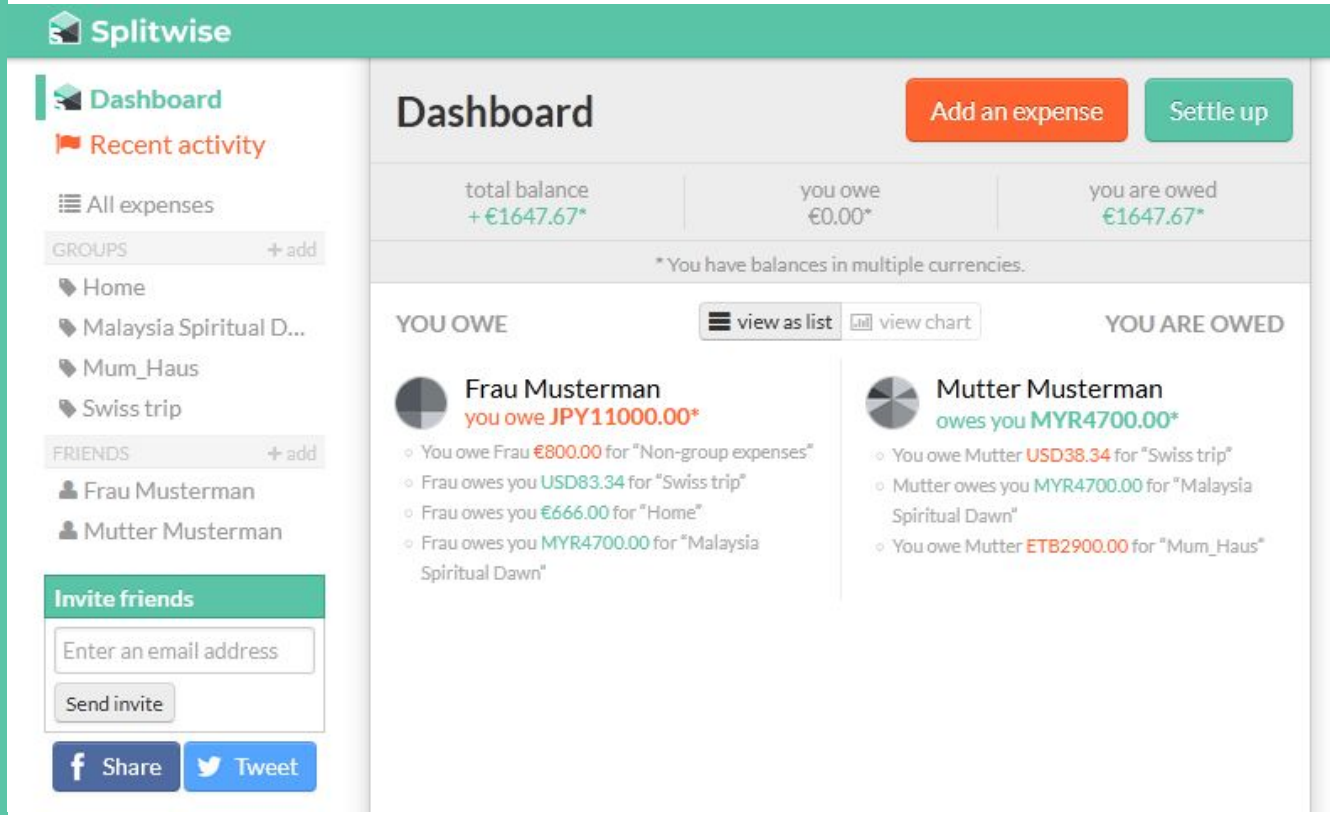
2.2.2 Accountability

```
fact balance will be recorded as income transaction:

    if fact balance > 0 -> positive transaction

    if fact balance < 0 -> negative transaction
```

2.2.2 Owes and Owed conversion to €



YOUR TOTAL BALANCE

you are owed
USD45.00

you are owed
CHF50.00

you are owed
MYR9400.00

you are owed
€1647.67

you owe
ETB2900.00

you owe
DKK1750.00

you owe
SEK2600.00

you are owed
GBP5.00

you owe
JPY11000.00

2.3

Prediction and Reporting

2.3.1 Prediction

Aim: 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.

- Problem: no data before december
- solution: retrieve from *Dec 2022* to *Feb 2023* and predict *Mar 2023*

2.3.1 Prediction

1) <u>Retrieve Data through SQL Query</u>

- Income= Subcategory_ID 101 to 105
- Expenses = rest of subcategory_ID

```
'''SELECT DISTINCT expense_date, subcategory_id, subcategory_name,
sum(base_amount) FROM Transactions AS t INNER JOIN TransactionItems AS
ti ON t.id = ti.transaction_id INNER JOIN Subcategories as s ON s.id =
t.subcategory_id WHERE expense_date BETWEENdate('2023-03-02', '-3
months') AND date('2023-03-02', '-2 months')  AND subcategory_id NOT
BETWEEN 101 AND 105 GROUP BY subcategory_id'''
```

2.3.1 Prediction

2) Sum up all base amount from income and expenses

- get balance amount for prediction model

3) Sum up all base amount from income and expenses

4) create Linear model and plot the output

2.3.2 Reporting

creating pie chart

- get percentage from each subcategory
- creating list to have amount and labels

```
pie_chart = ax[0].pie(list_amount_1, labels = labels1, startangle=90,
wedgeprops={'linewidth': 3.0, 'edgecolor': 'white'}, textprops=dict(color='white'))

plt.title('Expenses last month')

plt.legend(title = "subcategory name:",loc = 'best', prop={'size':8})

pie1 = plt.gcf()
```

2.3.2 Reporting

creating column chart

```
category = ['income', 'expenses']
values = [total_income, total_expenses]

fig = plt.figure(figsize = (5, 5))
plt.bar(category, values, color ='maroon',width = 0.5)
plt.xlabel('category')
plt.ylabel('amount in €')
plt.title('income vs. expenses in last 3 months')
bar = plt.gcf()
```

2.4

Graphic User Interface

2.3 Graphic User Interface

Initializing the User Interface

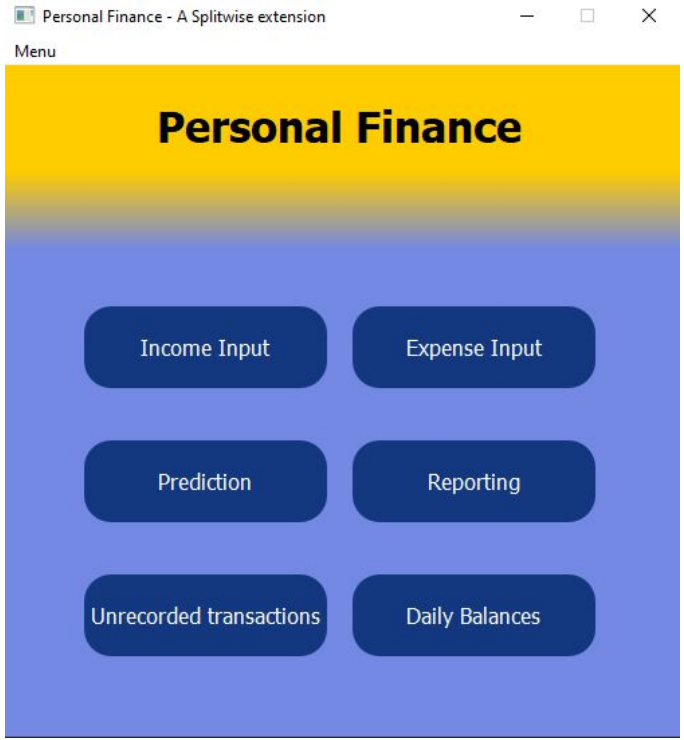
```
def run_app():
    # Run task 1, task 3 #
    run_sync()
    # Initiates App ##
    app = QApplication(sys.argv)
    main_win = MainWindow()
    main_win.show()
    sys.exit(app.exec_())
```

Initializing the Configuration

```
class MainWindow:
        oahq90 *
        def __init__(self):
                self.main_win = QMainWindow()
                self.main_win.setWindowTitle("Personal Finance - A Splitwise extension")
                self.ui = Ui_MainWindow()
                self.ui.setupUi(self.main_win)
                self.ui.stackedWidget.setCurrentWidget(self.ui.home)
```

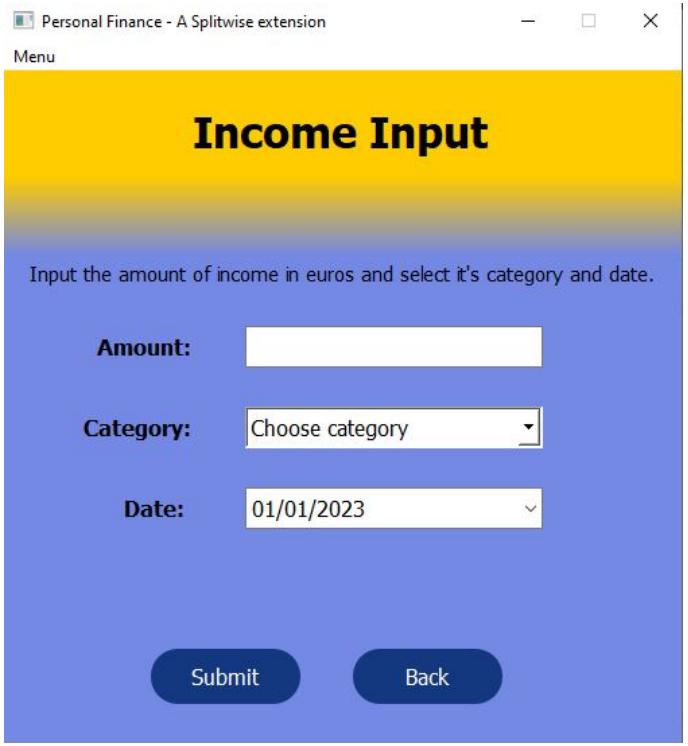
2.3 Graphic User Interface

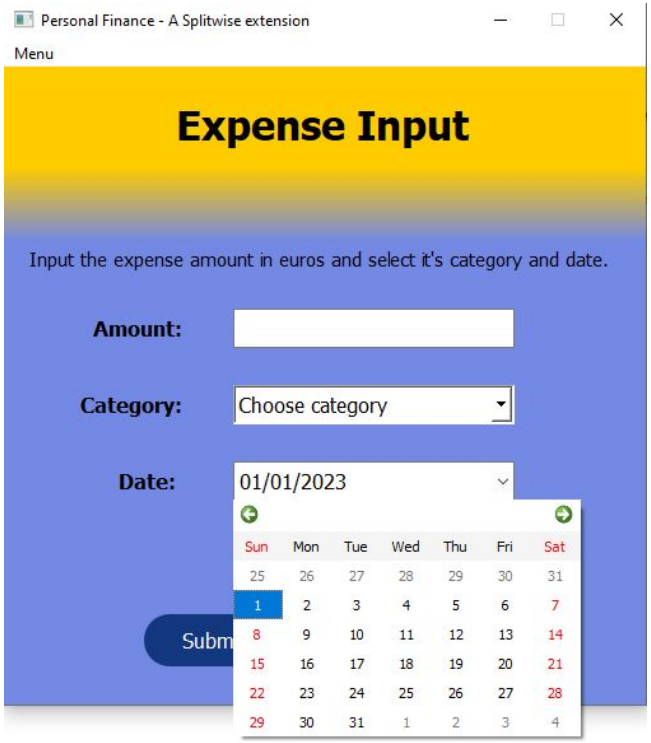
Layout



2.3 Graphic User Interface

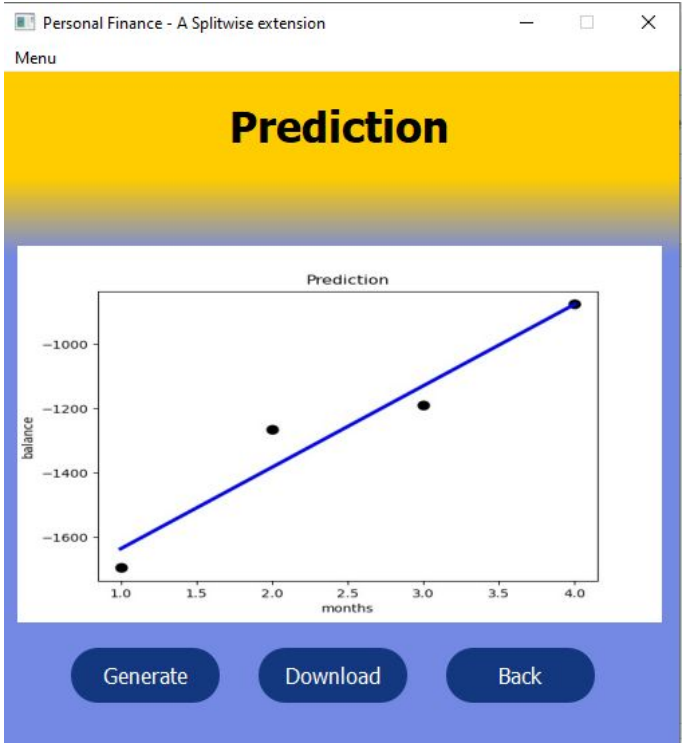
Layout

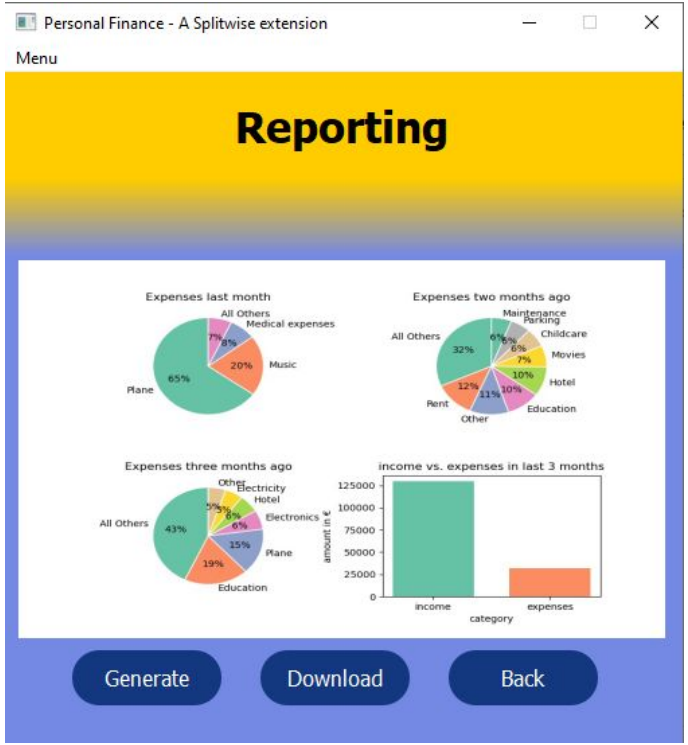




2.3 Graphic User Interface

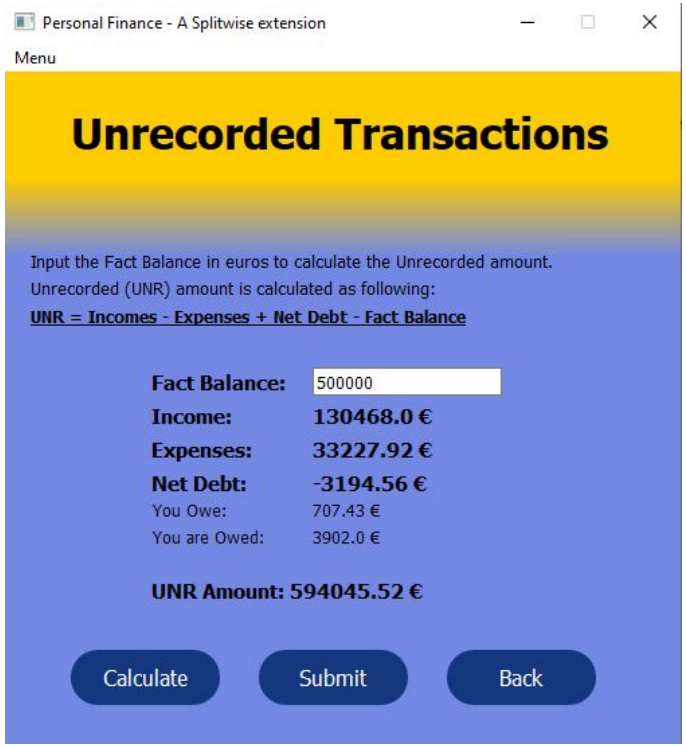
Layout

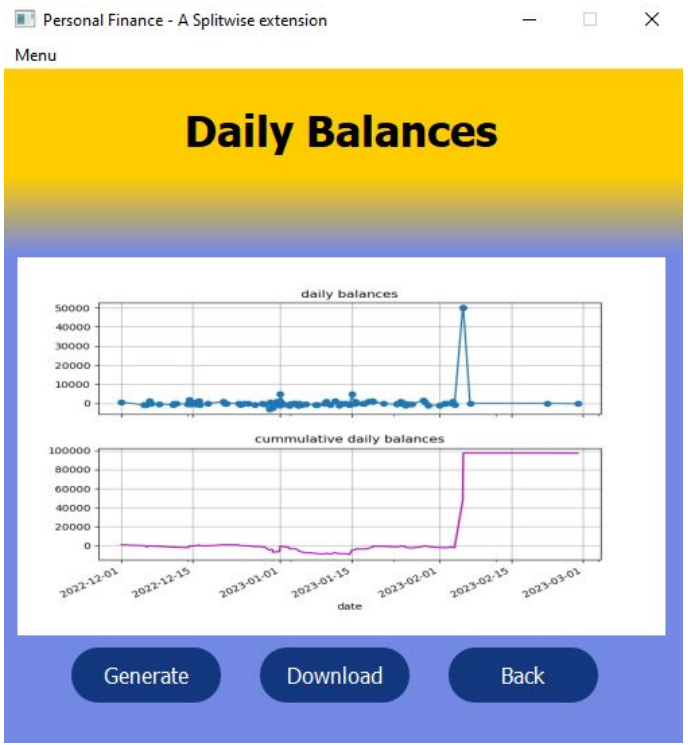




2.3 Graphic User Interface

Layout





3.

Limitations and Improvements

Limitations and Improvement

Limitations	Improvement
● can’t retrieve data before 12/2022 on splitwise	● fixing problem to get more data in the past ➔ more accurate prediction
● used linear regression model: inaccurate and unspecific	● try out other models for better prediction