Database “transaction.db” tables

users table sample

user_id	username	password
1	‘azcarraga’	‘azcarraga123’
2	‘mirasol’	‘mirasol123’
3	‘nicolas’	‘nicolas123’

transactions table sample

transaction_id	transaction_date	transaction_type	transaction_category	transaction_amount	transaction_descriptions	user_id
1	'2024-01-05'	'savings'	'Monthly Allowance'	5000.00	'Salary savings'	1
2	'2024-01-05'	'expense'	'Bills'	2500.00	'Electricity bill'	1
3	'2024-01-02'	'income'	'Salary'	25000.00	'Monthly paycheck'	1
4	'2024-01-02'	'investment'	'Stocks'	5000.00	'Initial stock investment'	1
5	'2024-03-10'	'savings'	'Monthly Allowance'	5000.00	'Salary savings'	2
6	'2024-03-07'	'expense'	'Bills'	2500.00	'Electricity bill'	2
7	'2024-03-04'	'income'	'Salary'	25000.00	'Monthly paycheck'	2
8	'2024-03-01'	'investment'	'Stocks'	5000.00	'Initial stock investment'	2
9	'2024-05-10'	'savings'	'Monthly Allowance'	5000.00	'Salary savings'	3
10	'2024-05-01	'expense'	'Bills'	2500.00	'Electricity bill'	3
11	'2024-04-01'	'income'	'Salary'	25000.00	'Monthly paycheck'	3
12	'2024-03-10'	'investment'	'Stocks'	5000.00	'Initial stock investment'	3

Backend (Requirements)

1. class **UserRepository**

---

2. class **UserManager**

---

3. dataclass **Transaction**
    - Just for storing data
    - Holds data about a certain transaction
    - Has attributes in the decorator
    - Don't have methods

    attributes:
    - *t_id*: **int**
    - *t_date*: **str**
    - *t_type*: **str**
    - *t_category*: **str**
    - *t_amount*: **float**
    - *t_description*: **str**

---

4. class **TransactionRepository**
    - The class that directly interacts with the database
    - Can read, write, update, and delete to the database
    - Uses the **Transaction** class to store data

    methods:

    + **getAllTransactions(): # for profile page # history page**
        - Takes in one parameter:
            - **current_user_id** (an **int**)
        - Searches for rows in the **{transactions}** table of the database where:
            - **current_user_id** matches **user_id** column
        - Creates a **Transaction** object for each matching row
        - Appends all **Transaction** objects to a List
        - Returns the list (a **List[Transaction]**)

    + **getTransactionsByType(): # for profile page # for edit page # for history page**
        - Takes in two parameters:
            - **current_user_id** (an **int**)
            - **t_type** (a **str**)

- Searches for rows in the **{transactions}** table of the database where:
  - **current_user_id** matches **user_id** column
  - **t_type** matches **transactoin_type** column
- Creates a **Transaction** object for each matching row
- Appends all **Transaction** objects to a List
- Returns the list (a **List[Transaction])**

+ **getTransactionsByCategory(): # for history page**
  - Takes in two parameters:
    - **current_user_id** (an **int**)
    - **t_category** (a **str**)
  - Searches for rows in the **{transactions}** table of the database where:
    - **current_user_id** matches **user_id** column
    - **t_category** matches **transaction_category** column
  - Creates a **Transaction** object for each matching row
  - Appends all **Transaction** objects to a List
  - Returns the list (a **List[Transaction])**

+ **getRecentTransactions(): # for home page**
  - Takes in two parameters:
    - **current_user_id** (an **int**)
    - **t_count** (an **int**)
  - Searches for rows in the **{transactions}** table of the database where:
    - **current_user_id** matches **user_id** column
  - Retrieves a specific number of recently inserted rows indicated by **t_count**
  - Creates a **Transaction** object for each row
  - Appends all **Transaction** objects to a List
  - Returns the list (a **List[Transaction])**

+ **addTransaction(): # for add page**
  - Takes in two parameters:
    - **current_user_id** (an **int**)
    - **transaction** (a **Transaction** object)
  - Converts the **transaction** object into a tuple
  - Inserts the tuple into the **{transactions}** table of the database:
  - Sets the **user_id** column to **current_user_id**

+ **modifyTransaction(): # for edit page**
  - Takes in three parameters:
    - **current_user_id** (an **int**)
    - **t_id** (an **int**)
    - **transaction** (a **Transaction** object)
  - Converts the **transaction** object into a tuple
  - Updates the row in the **{transactions}** table of the database using the tuple where:
    - **current_user_id** matches **user_id** column
    - **t_id** matches **transaction_id** column

+ **deleteTransaction():**
    - Takes in two parameters:
        - **current_user_id** (an **int**)
        - **t_id** (an **int**)
    - Deletes the row in the **{transactions}** table of the database where:
        - **current_user_id** matches **user_id** column
        - **t_id** matches **transaction_id** column

---

5. dataclass **Finance**

- Just for storing data
- Holds data about a user's finances
- Has attributes in the decorator
- Don't have methods

attributes:

- *total_income*: **float**
- *total_expenses*: **float**
- *total_savings:* **float**
- *total_investment*: **float**

---

6. class **TransactionManager**

- Handles the app's logic
- Initializes **TransactionRepository**
- Calculates data
- Creates graph
- Uses the **Transaction** and **Finance** classes to store data

methods:

+ **calculateOverallFinance(): # for profile page**
    - Takes in one parameter:
        - **user_id** (an **int**)
    - Uses the **getAllTransactions(user_id)** method of the **TransactionRepository** object to get all of the user's transactions
    - Calculate the total amount of each transaction type separately
    - Creates a **Finance** object containing the totals of each type
    - Returns the **Finance** object

+ **calculateOverallBalance(): # for profile page # for home page**
    - Takes in one parameter:
        - **user_id** (an **int**)
    - Uses **calculateOverallFinance(user_id)** to get **overall_fiance** (a **Finance** object)
    - Uses **total_expenses** and **total_income** of the **overall_finance** object
    - Subtracts **total_expenses** from **total_income**
    - Returns the result (a **float**)

+ **calculateMonthlyFinances(): # for home page**
    - Takes in one parameter:
        - **user_id** (an **int**)
    - Uses the **getTransactionsByType(user_id)** method of the **TransactionRepository** object to get all of the user's transactions for each type:
        - “expense”, “savings”, “income”, “investment”
    - Stores each result (a **List[Transactions]**) separately (one for each type)
    - Sort and group the transactions from the list by **year_month** (one dictionary for each type):
        - Each dictionary maps **year_month** (a **str**) to **List[Finance]**
        - For **year_month** gap that doesn’t have transactions for a given type, map the **year_month** to a list containing a dummy **Transaction** object where all attributes are set to 0.0 (a **float**)
        - Ex:

          expenses = {

          “2024-01”:[transaction], # transaction obj containing all 0s

          “2024-02”:[transaction1, transaction2, transaction3…],

          “2024-04”:[transaction], # transaction obj containing all 0s,

          “2024-05”:[transaction1, transaction2, transaction3…],

          …

          “2025-01”:[transaction1, transaction2, transaction3…],

          …

          }

          savings = {…}

          income = {...}

          investment = {...}
    - For each year-month:
        - Calculates the total amount of each transaction type
        - Creates a **Finance** object containing the totals of each transaction type
    - Appends each **Finance** object to a dict:
        - The dictionary maps **year_month** (a **str**) to the **Finance** object containing the totals for a specific month
        - Ex:

          monthly_finances = {

          “2024-01”:total_finance

          “2024-02”:total_finance,

          “2024-03”:total_finance,

          “2024-04”:total_finance,

          …

          “2025-01”:total_finance,

          …

          }
    - Returns the dictionary (a **Dict[str, Finance]**)

+ **calculateQuarterlyFinances(): # for home page**
    - Takes in one parameter:
        - **user_id** (an **int**)
    - Uses the **getTransactionsByType(user_id)** method of the **TransactionRepository** object to get all of the user's transactions for each type:
        - “expense”, “savings”, “income”, “investment”
    - Stores each result (a **List[Transactions]**) separately (one for each type)
    - Sort and group the transactions from the list by **year_quarter** (one dictionary for each type):
        - Each dictionary maps **year_quarter** (a **str**) to **List[Finance]**
        - For **year_quarter** gap that doesn’t have transactions for a given type, map the **year_quarter** to a list containing a dummy **Transaction** object where all attributes are set to 0.0 (a **float**)
        - Ex:

          expenses = {

          “2024-Q1”:[transaction], # transaction obj containing all 0s

          “2024-Q2”:[transaction1, transaction2, transaction3…],

          “2024-Q4”:[transaction], # transaction obj containing all 0s,

          “2025-Q1”:[transaction1, transaction2, transaction3…],

          …

          “2025-Q4”:[transaction1, transaction2, transaction3…],

          …

          }

          savings = {…}

          income = {...}

          investment = {...}
    - For each year-quarter:
        - Calculates the total amount of each transaction type
        - Creates a **Finance** object containing the totals of each transaction type
    - Appends each **Finance** object to a dict:
        - The dictionary maps **year_quarter** (a **str**) to the **Finance** object containing the totals for a specific quarter
        - Ex:

          quarterly_finances = {

          “2024-Q1”:total_finance

          “2024-Q2”:total_finance,

          “2024-Q3”:total_finance,

          “2024-Q4”:total_finance,

          …

          “2025-Q4”:total_finance,

          …

}

- Returns the dictionary (a **Dict[str, Finance]**)

+ **createMonthlyGraph(): # for home page**
  - Takes in one parameter:
    - **monthly_finances** (a **Dict[str, Finance]**)
  - Plots the monthly data into a grouped bar graph using the **{matplotlib}** library
    - With title, labels, and legend
  - Returns the result (a **matplotlib.figure.Figure**)

+ **createQuarterlyGraph(): # for home page**
  - Takes in one parameter:
    - **quarterly_finances** (a **Dict[str, Finance]**)
  - Plots the quarterly data into a grouped bar graph using the **{matplotlib}** library
    - With title, labels, and legend
  - Returns the result (a **matplotlib.figure.Figure**)