NILE UNIVERSITY OF NIGERIA

Project:

Build an expense tracker app using python

Course:

GET 204, Object Oriented Programming

Lecturer:

Emmanuel Ali(PhD)

Group:

20232182	Ebube Ekelem
20232100	David Tiwatope
20232621	Samson Mafe

Project Name:

KoboKo

…if you no guide, you go cry

Features(following the instructions and including a pie chart feature)

- User authentication (login and register)
- Add, and delete expense
- Categorize expenses (food, travel, bills, etc.)
- View expenses and total cost
- View categories
- View and save visualization of data, pie chart

```
koboko/
|----README.md
|----requirements.txt
├── main.py                # Entry point for the application
|----assets/                # Where the icon and logo are stored
|---__init__.py
├── models/                # Contains all data models
|   ├── __init__.py
|   ├── user.py            # User model
|   ├── expense.py         # Expense model
|   ├── category.py        # Category model
|   └── report.py          # Report model
├── database/              # Database logic
|   ├── __init__.py
|   └── db.py              # SQLite connection and schema
└── tests/                 # Unit tests
    ├── __init__.py
    ├── test_user.py       # Tests for user model and logic
    ├── test_expense.py    # Tests for expense model and logic
    ├── test_category.py   # Tests for category model and logic
    └── test_report.py     # Tests for report model and logic
```

PURPOSE OF .py FILES

database/db.py

get_connection(): Connects to the SQLite database.

initialize_database(): Creates the users, categories, and expenses tables if they don't already exist.

models/user.py

hash_password: Hashes a plain-text password using bcrypt.

verify_password: Verifies a plain-text password against a hashed password.

save: Saves a new user to the database.

authenticate: Authenticates a user by checking their username and password.

models/expense.py

save: Saves a new expense to the database.

get_expenses_by_user: Retrieves all expenses for a specific user, including category names

models/category.py

save: Saves a new category to the database.

get_all_categories: Retrieves all categories from the database.

models/report.py [Pie Chart Feature]

report.py fetches each user's total spending per category from the database, processes this data, and uses matplotlib to display a pie chart showing the percentage breakdown of expenses by category.

It helps users visually understand where their money goes.

The main function is generate_pie_chart(user_id), which runs all these steps for the logged-in user.

test_user.py

Tests user registration and authentication.

Verifies that users are saved correctly and that login works with the correct password.

test_category.py

Tests adding and retrieving categories.

Ensures that duplicate categories are not allowed.

test_report.py

Tests the pie chart feature.

Ensures that the map can be generated without errors and that the data is aggregated correctly.

test_expense.py

setUp Method: This method runs before each test to set up a clean database state.

It clears existing data and inserts test data (e.g., a test user and category).

test_add_expense Method: This is the actual test for adding an expense.

It creates an Expense object, saves it to the database, and verifies that the data was correctly inserted.

Assertions:

self.assertIsNotNone(result): Ensures that the query returned a result (i.e., the expense was added).

self.assertEqual(result[0], "Lunch"): Verifies that the description matches.

self.assertEqual(result[1], 15.50): Verifies that the amount matches.

Purpose of Test Files

The test files (test_expense.py, test_user.py, etc) are used to verify that your code works as expected. They contain unit tests, which are small, focused tests for individual pieces of functionality

- Verify individual functionality (e.g., adding expenses, user authentication).
- Use setUp to prepare a clean test environment.
- Use assertions to check expected outcomes.

__init__.py

The __init__.py file is used to mark a directory as a Python package. Without it, Python will not treat the directory as a package, and you won't be able to import modules from it.

Even if the file is empty, its presence signals to Python that the directory should be treated as a package.

Why Empty?

If you don't need to initialize anything specific for the package (e.g., no shared variables, imports, or setup logic), the file can remain empty.

It's still required for compatibility with older versions of Python (prior to Python 3.3). In Python 3.3 and later, __init__.py is technically optional, but it's still a good practice to include it for clarity and consistency.

__init__.py: Marks directories as Python packages.

__pycache__.pyc

The __pycache__ directory is automatically created by Python to store compiled bytecode files (with .pyc extensions) for your Python scripts.

These files are generated to improve performance. When you run a Python script, Python compiles it into bytecode, which is faster to execute than interpreting the source code every time.

Can I Delete __pycache__?

Yes, you can delete the __pycache__ directory, but Python will recreate it the next time you run your scripts.

If you want to prevent it from being created (e.g., for a clean workspace), you can set the PYTHONDONTWRITEBYTECODE environment variable:

```
set PYTHONDONTWRITEBYTECODE=1
```

However, this is generally not recommended because it disables Python's performance optimization.

Should I Ignore __pycache__ in Version Control?

Yes, you should add __pycache__/ to your .gitignore file (if using Git) to avoid committing it to your repository.

__pycache__: Stores compiled bytecode for performance; safe to delete but will be recreated.

main.py

main.py manages the app's user interface and workflow using Tkinter, handling user registration, login, and all main features after login.

It connects user actions (like adding/viewing expenses or categories) to the database and reporting functions.

The core logic is in the KoboKoApp class, which controls the app's windows, buttons, and user session.

Extras

conn: Manages the database session (like a bridge between Python and DB).

cursor: Handles query execution and result retrieval (like a tool to interact with data).

Issues Faced

1. The implementation was using terminal-based input (input()), which is not ideal for a GUI application. To fix this, we need to replace the terminal-based interactions with Tkinter input dialogs or entry fields in the GUI window.
   Solution
   Replaced input() with GUI Dialogs:
   Used simpledialog.askstring() to prompt the user for input (e.g., username, password, expense details).
   Used messagebox.showinfo() and messagebox.showerror() to display success or error messages.

2. Icon and Logo insertion, was too big

   Solution

   Reduced size to 256x256px

3. when i run main, the pie chart shows first and i have to close it befor the app opens, or they opened together which is not supposed to be so
   solution
   By removing the direct call to Report.generate_pie_chart(user_id=1), the pie chart will no longer open automatically when the script is run.
   But
   When resolved another problem developed

4. when I removed Report.generate_pie_chart(user_id=1), the pie chart wasn't opening at launch, but the app opened login and register, just one large logo and welcome
   solution,
   there was nothing wrong with code, possible solution is to exempt the logo, but I didn't try that so I worked with issues no3
   update: instead of removing report.generate_pie_chart(user id=1), I just replaced it pass,

5. wanted to create a standalone by writing pyinstaller --onefile --windowed koboko/main.py to terminal after installin pyinstalling but for some reasons, the (.exe) wasn't functioning
   solution
   I might just fashi it, cuz others are saying its not important, ill update u guys if I correct it

6. View expenses is showing unsorted with no result
   Solution
   We used only one geometry manager (grid) consistently in the view_expenses method.

7. Login and register is still there after begin logged in
   Solution
   we added a condition to check the self.logged_in_user_id. If the user is logged in (self.logged_in_user_id is not None), we will skip adding the "Login" and "Register" buttons.

8. Error: UNIQUE constraint failed: users.username
   This error occurs because you are trying to register a user (test) with a username that already exists in the database. The username column in the users table is likely defined as UNIQUE, which prevents duplicate usernames.

9. Error: UNIQUE constraint failed: categories.name
   This error occurs because you are trying to add a category with a name that already exists in the categories table. The name column in the categories table is likely defined as UNIQUE.

   Solution

   I ignored (making them comments) line 1 to 62, they were added to test run the models, this solved issues 8 and 9.