CS 635, Fall 2025
Module 4 Assignment

Prepared by:
Leo Paredes (820558278), Brandon Reynolds (821924175)
E-mail: lparedes7353@sdsu.edu , bareynolds@sdsu.edu
San Diego State University

Chosen Project for Refactor:
https://github.com/caesarmario/online-banking-system-with-python/blob/main/Simple%20Online%20Banking%20System.py

Initial Summary of Repo:
This repo is a relatively simple project for an Online Banking System by user caesarmario. The program allows two types of users, Admins and Customers , to perform operations such as login, profile creation, deposits, withdrawals, and viewing transactions. It uses mock databases through .txt files such as in admin_db.txt, customer_db.txt, and transaction_db.txt to simulate working databases for testing.

Initial Code Analysis
After giving a thorough review of the code, we found there are multiple areas that could use refactoring. The first area for improvement is reducing code duplication in places such as in the Admin and Customer functions. These functions reuse the same logic for login, file handling, and menu loops. The second thing we saw could be better is that there was a lack of modularity since all the functions are in one file rather than in different classes or files. Next, there is a lack of error handling especially since there are files being opened and closed to simulate online banking. Lastly there are no test suites that cover the functionality being implemented.

Motivation For Refactoring
The specific portions of the code we want to refactor are admin_login(), customer_login(), admin_menu(), and customer_menu(). The main challenges with the current code implementations of these functions come from how the logic is structured and repeated. Both the admin and customer login functions use almost the same code. This can increase the time it takes to run as the user size grows. The menus have similar problems because they mix input, printing, and logic all in one place. This makes the program hard to read and maintain. Lastly, there are no tests for these functions verifying their outputs. Refactoring can address these issues and increase the robustness of the original code.

Refactoring Plan
The refactor plan we have decided to apply to the repository involves separation and emphasis of functions in the overall banking system and adding testing to verify all functions of the code. The intended changes will be creating two folders: “src” and “tests”. The src folder will have several files that separate functions within the banking system. This includes the database, services, models, and the main file that will execute

the banking system. The database file acts as a data manager by reading, writing, and updating information stored in the text CSV files. It allows the program to easily save and load data directly from the files. The services file contains the logic of loading customers, checking logins, adding customers/admins, and recording transactions. This separation of logic makes the code more organized and easy to test. The models file defines the users login information, customer information, and transaction information. Below is a Unified Modeling Language diagram for the refactoring plan. T

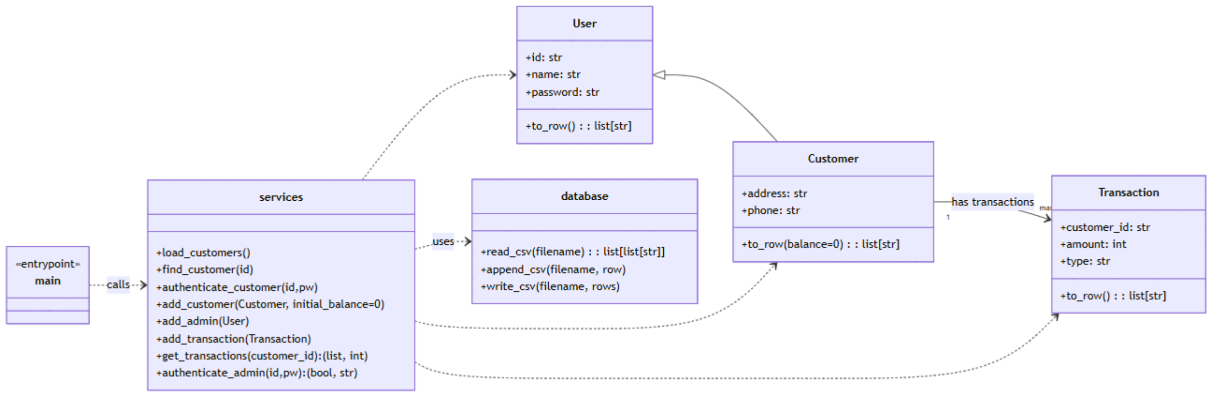
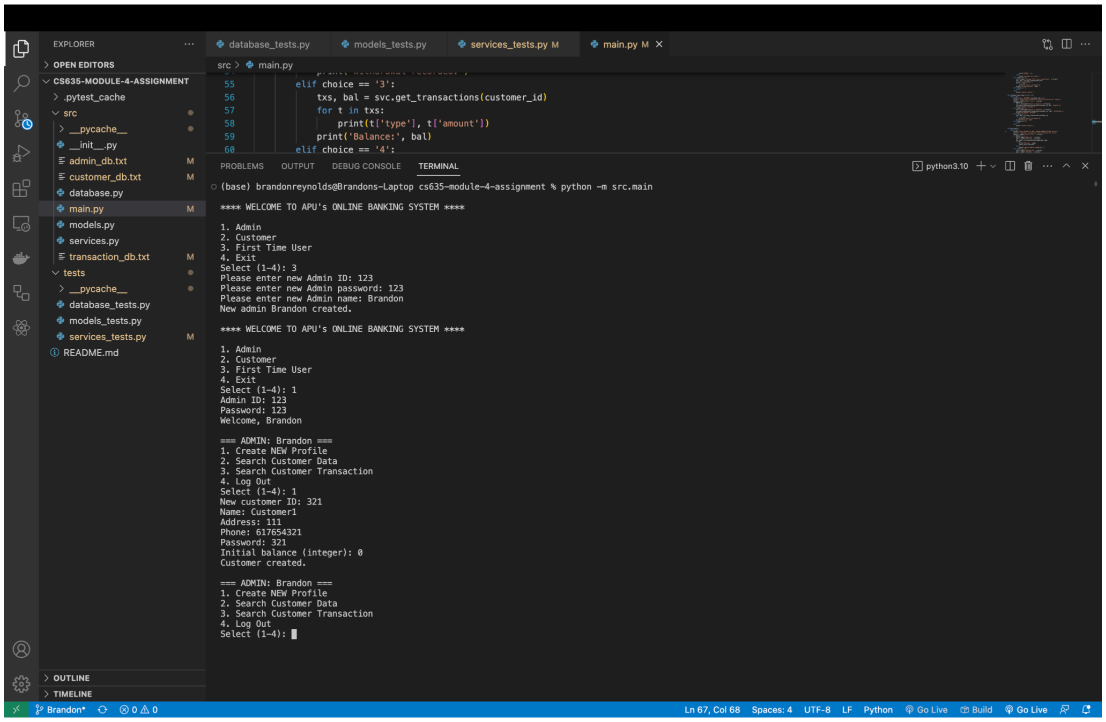

User
+id: str
+name: str
+password: str
+to_row() : : list[str]
«entrypoint»
main
calls
services
+load_customers()
+find_customer(id)
+authenticate_customer(id,pw)
+add_customer(Customer, initial_balance=0)
+add_admin(User)
+add_transaction(Transaction)
+get_transactions(customer_id):(list, int)
+authenticate_admin(id,pw):(bool, str)
uses
database
+read_csv(filename) : : list[list[str]]
+append_csv(filename, row)
+write_csv(filename, rows)
Customer
+address: str
+phone: str
+to_row(balance=0) : : list[str]
has transactions
1
Transaction
+customer_id: str
+amount: int
+type: str
+to_row() : : list[str]

Fig. 1. UML Diagram of Refactored Banking System

The expected outcomes after refactoring are an organized and maintainable repository. There will be a README file that clearly explains the purpose of the repository, its contents, and how to use it. And lastly, there will be unit tests that provide code coverage of the banking system.

Outcome Analysis

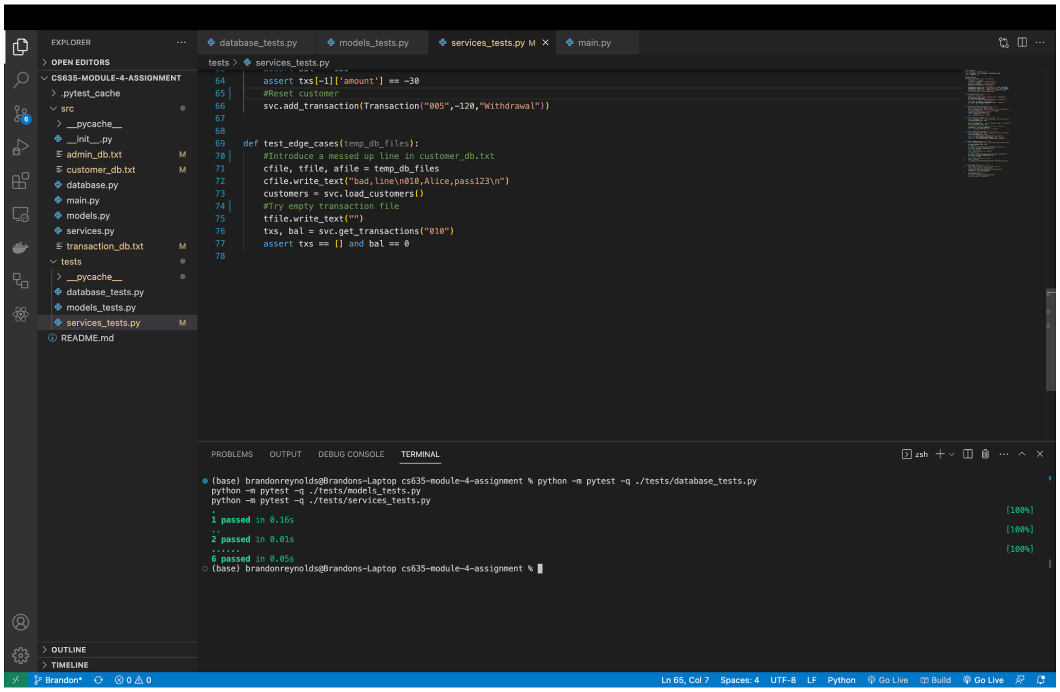
The refactoring process was a success. The overall outcome was an organized repository with clear separation of file purposes. The original repository had a long README file and a monolith script that contained the entire banking system. The refactored version has a documentation, source, and test folders. This has improved code readability and increased the overall robustness of the code. A challenge faced was adding the feature of dynamically adding a new user's credentials. The original banking system had the admin credentials hard coded. In order to dynamically add admins, we had to implement a first time user option to save admin credentials. This removed the dependency on hard coded credentials and improved the overall banking systems performance. Below is a screenshot of the banking system being used after running main and tests using Pytest.



```python
        elif choice == '3':
            txs, bal = svc.get_transactions(customer_id)
            for t in txs:
                print(t['type'], t['amount'])
            print('Balance:', bal)
        elif choice == '4':
```

```
(base) brandonreynolds@Brandons-Laptop cs635-module-4-assignment % python -m src.main

**** WELCOME TO APU's ONLINE BANKING SYSTEM ****

1. Admin
2. Customer
3. First Time User
4. Exit
Select (1-4): 3
Please enter new Admin ID: 123
Please enter new Admin password: 123
Please enter new Admin name: Brandon
New admin Brandon created.

**** WELCOME TO APU's ONLINE BANKING SYSTEM ****

1. Admin
2. Customer
3. First Time User
4. Exit
Select (1-4): 1
Admin ID: 123
Password: 123
Welcome, Brandon

=== ADMIN: Brandon ===
1. Create NEW Profile
2. Search Customer Data
3. Search Customer Transaction
4. Log Out
Select (1-4): 1
New customer ID: 321
Name: Customer1
Address: 111
Phone: 617654321
Password: 321
Initial balance (integer): 0
Customer created.

=== ADMIN: Brandon ===
1. Create NEW Profile
2. Search Customer Data
3. Search Customer Transaction
4. Log Out
Select (1-4):
```

Fig. 2. Terminal Output of Banking System



```python
    assert txs[-1]['amount'] == -30
    #Reset customer
    svc.add_transaction(Transaction("005",-120,"Withdrawal"))


def test_edge_cases(temp_db_files):
    #Introduce a messed up line in customer_db.txt
    cfile, tfile, afile = temp_db_files
    cfile.write_text("bad,line\n010,Alice,pass123\n")
    customers = svc.load_customers()
    #Try empty transaction file
    tfile.write_text("")
    txs, bal = svc.get_transactions("010")
    assert txs == [] and bal == 0
```

```
(base) brandonreynolds@Brandons-Laptop cs635-module-4-assignment % python -m pytest -q ./tests/database_tests.py
python -m pytest -q ./tests/models_tests.py
python -m pytest -q ./tests/services_tests.py
.                                                                                          [100%]
1 passed in 0.16s
..                                                                                         [100%]
2 passed in 0.01s
......                                                                                     [100%]
6 passed in 0.05s
(base) brandonreynolds@Brandons-Laptop cs635-module-4-assignment %
```

Fig. 3. Pytest Results

Conclusion and Recommendations

Overall, this assignment was a great learning experience with applying refactoring techniques. It was satisfying for us to change a monolith script to an organized code base with increased flexibility and functionality. The best practices we identified are to separate functions into their own files when needed. It helps improve code structure and aid with unit testing. For ongoing maintenance, it would be best to increase unit testing with verbose results. This will increase the overall robustness of the base and help the developer understand what is working. For future refactor efforts, we would add the ability to delete customers and admins. This would close the loop of simulating a real-world system.