Phase 3 Project Report – GillPay™: A Personal Finance Tracker for Students

Course: ISTM 601
Team: Goofy Goldfishes
Phase 3 Due Date: September 28, 2025

Project Title and Team Information

Project Title: GillPay™: A Personal Finance Tracker for Students

- Team Members & Roles (Phase 3):
    - Ruben Alvarez – Team Leader
    - Ally Herleth – Developer 1 (Create reports)
    - Matt Bennett – Developer 2 (Implement data validation)
    - Jo Ann Kern – Developer 3 (Build exception handling)
    - Enoch Adegbola – Tester/Documenter

Overview of Features

In Phase 3, GillPay™ moves beyond core transaction recording to introduce reporting capabilities and robust exception handling. The new features allow users to generate two meaningful reports, expense by category and spending by month. These reports are intended to help students better understand their financial behavior, identify patterns, and adjust their spending habits accordingly.

To support this functionality, enhanced data validation ensures that all user inputs are accurate, consistent, and free from corruption. Exception handling mechanisms are also implemented so the system can manage errors such as corrupted CSV files, invalid dates, or empty datasets without interrupting program execution. Reports and error-handling features are accessible through the existing command-line interface (CLI), keeping the tool lightweight and straightforward to use.

Main Program File

The main entry point for GillPay™ is the file **main.py**. This file contains the program's primary menu system and calls supporting modules for handling transactions and storage.

To run the program, users must ensure they are working in a Python 3.10+ environment. Once installed, the program can be started with the command:

python main.py

In *gillpay_service.py*, we added one dependency, the PrettyTable library, to improve report readability and presentation.

File Descriptions

The submission for Phase 2 contains several files that together make up the GillPay™ system:

- **/src/main.py** - The main entry file, which presents users with a command-line menu and directs input to the appropriate functions.
- **/src/model/transaction.py** - Defines the Transaction class and associated functions, including user input collection and input validation; This is the container for the transaction data.
- **/src/gillpay_service.py** - Handles the business logic for the GillPay application and interacts with the transaction_dao.py to help formulate the data.
- **/src/dao/transaction_dao.py** - Handles the reading and writing of the data for the CSV file.
- **/data/gillpay_data.csv** - A persistent storage file that records all income and expense entries in comma-separated format.
- **/docs/goofygoldfishes_project_report_phase2.pdf** - The documentation for this phase of the project.

Each of these files plays a distinct role in the project architecture, and together they support the primary functionality introduced in this phase.

User Instructions

As shown below, we have added important instructions for the user to follow in our *README.md* file. It explains how the user can run the *main.py* file that is located within the **src**/.

```
# Important - Phase 3 Process Only
- Run main.py file located in src/
- Below guidance will be used for Phase 4 deliverable
```

To use GillPay™, the user should first run the main program file as described above. Upon launch, the user is presented with a simple menu:

1 → Add a transaction
2 → View account summary
3 → View Expense by Category Report
4 → View Spending by Month Report
5 → Exit the program (Farewell!)

When adding a transaction, the program prompts the user for the date, description, category, amount, and type of transaction. Once the information is provided, the program validates the input and records the transaction in gillpay_data.csv.

For example:

```
Please make a choice:
1: Add Transaction
2: Account Summary
3: Expense by Category Report
4: Spending by Month Report
5: Farewell!
Action:
```

The resulting entry is stored in the CSV file as:

```
type,category,amount,date
income,job,12.15,2025/09/14
```

By selecting 2 from the main menu, the user can quickly view totals for all income and expense entries to date, as well as the resulting net savings.

Selecting 3 generates the user's report for expenses by category, dividing it into two columns "Category" and "Amount" where the category declares the type of expense and the amount contains the total amount for each category.

Selecting 4 generates the user’s report for spending by month which is divided into 4 columns, “Month”, “Income”, “Expense”, “Net”. The month states which month of the year it is in YYYY-MM format, the income is the total income for the month, the expense contains the total expenses for the month, and the net is the total net income for the month.

Assumptions and Design Decisions

The team decided to maintain CSV as the storage format for Phase 3 to ensure portability and compatibility with existing modules. Reports were designed to work directly from this dataset, allowing users to generate summaries without requiring additional setup.

Input validation was expanded to include stricter checks on dates (enforcing YYYY/MM/DD), transaction types (income or expense only), amounts (numeric), and categories (verified). The CLI was retained to keep development focused on backend reliability before moving to the Tkinter interface planned for Phase 4.

Challenges and Solutions

Phase 3 presented several challenges. A major concern was ensuring that incorrect or corrupted data did not interfere with reporting accuracy. To address this, the team implemented exception handling that intercepts invalid inputs and provides clear error messages with instructions for correction.

Another challenge was aligning multiple developers’ contributions across validation, reporting, and error handling required consistent communication and coordination. GitHub was again used for version control, while Discord meetings ensured timely collaboration and integration. Our team also used the same agreed-upon IDE, PyCharm, to consistently develop and test the application.

Use of Generative AI

Generative AI tools such as ChatGPT were used to brainstorm report formats, develop sample validation rules, and draft exception-handling structures. The AI provided helpful starting points for pseudocode and error management strategies, which were then refined and customized by the team to meet project requirements.

All AI-generated suggestions were tested, adapted, and documented to ensure they aligned with the project's technical and functional goals.

Future Improvements

Looking ahead, additional reports such as category-based spending trends or monthly breakdowns could be added to further enrich the user experience.

Visual enhancements, including Turtle-based representations of reports, will be developed in Phase 4 to make financial patterns more engaging. The team also plans to expand category functionality based on the Phase 2 feedback so that we can generate category recommendations to improve consistent outputs and user experience.

Aggie Honor Code Statement

Statement: "An Aggie does not lie, cheat, or steal, or tolerate those who do".

Purpose: To unify the aims of all Texas A&M students toward a high code of ethics and personal dignity, promoting loyalty to truth and confidence in one another.

Application: The code applies to all academic and personal pursuits, emphasizing honesty and integrity.