# Project Overview:

The goal is to develop a desktop application that simplifies the budgeting process for users, especially those with little to no prior experience. The app focuses on a daily budget that supports long-term goals, making it easy for users to manage their finances.

# Technical Requirements:

1. Platform: Desktop application for Windows and macOS, developed using Python and PyQt.
2. Language: Python for both front-end and back-end development, with PyQt for the user interface.
3. Database: Local file storage using a compatible database system, such as SQLite or SQLAlchemy, for seamless integration with Python.
4. Data Import: CSV import functionality for transaction data, allowing users to manually download and import their bank statements in CSV format.

# Functional Requirements:

1. Dashboard, double-click functionality, running balance, accrual-basis accounting, expensing large items over time, split transactions, gross income recognition, dividend/capital gains income, cost-basis, unrealized capital gains, dollar-weighted return, asset allocation, data validation, error handling, CSV import, sorting, conditional formatting
2. Portfolio tab: Update the tab to utilize appropriate functions or libraries for pulling historical end-of-month values of assets.
3. Net Worth tab: Update the tab to display a Net Worth chart with historical data.
4. Income Section: Users can input their income sources, including salary, tips, and frequency (monthly, weekly, or bi-weekly).
5. Expenses Section: Users can input various category types of expenses, including utilities, bills, and other expenditures. This will allow for easier overview.
6. Financing and Loan Overview: Allows for input, updating, of all current loans, and monthly payments towards those loans. If there is a type, it can vary between auto, personal, etc.
7. Big Purchase Planning: Users can input a large purchase and adjust their budget accordingly to accommodate it.
8. Budget Summary: A summary page displaying the user's daily, weekly, and monthly budget, as well as a breakdown of income, expenses, and savings.

# Non-Functional Requirements:

1. Performance: The app should load quickly and provide a smooth user experience.
2. Scalability: Single user only.
3. Security: All user data should be stored securely, and the app should follow best practices for data encryption and protection.
4. Usability: The app should be intuitive and easy to use, even for those with no prior budgeting experience.

# Development Phases:

Phase 1: Project planning, resource allocation, and design of the app UI/UX.

Phase 2: Front-end and back-end development, database design, and implementation of CSV import functionality.

Phase 3: Development of core features, including dashboard, income and expense sections, financing and loan overview, big purchase planning, and budget summary.

Phase 4: Integration of additional features, such as double-click functionality, running balance, accrual-basis accounting, expensing large items over time, split transactions, gross income recognition, dividend/capital gains income, cost-basis, unrealized capital gains, dollar-weighted return, asset allocation, data validation, error handling, sorting, conditional formatting, and portfolio and net worth tabs.

Phase 5: Testing, bug fixing, optimization, and ensuring security and usability best practices.

Phase 6: Deployment, user onboarding, and post-launch support.

# Milestones:

Milestone 1: Completion of Phase 1 (UI/UX design)

Milestone 2: Completion of Phase 2 (Front-end and back-end development, database design, and CSV import functionality)

Milestone 3: Completion of Phase 3 (Core feature development)

Milestone 4: Completion of Phase 4 (Integration of additional features and functionalities)

Milestone 5: Completion of Phase 5 (Testing, optimization, and ensuring security and usability best practices)

Milestone 6: App launch and completion of Phase 6 (Deployment and support)

# Development Platform

This application is and will be for personal use. The bank that I use has 2FA, therefore a simple free API will not due. I don’t plan on checking the finances, more than once a month. Applications that are similar run $3-15 a month.

There were four choices for desktop application, C#-WPF, Java-JavaFX, HTML-Electron, Python-PyQT. One of the primary concerns for this application is maintenance over the years. Which rules out a few of these. Deciding to first continue the project with Python for the following factors:

Ease of Use: Python is known for its readability and ease of use. PyQt and PySide offer a straightforward way to create desktop applications with a gentle learning curve.

Cross-Platform Support: PyQt and PySide are both cross-platform, which means you can create a single application that runs on Windows, macOS, and Linux without significant changes to the code.

Deployment: Python offers various tools to package and distribute your application, such as PyInstaller or cx\_Freeze. These tools allow you to create standalone executables for your target platforms, simplifying the deployment process.

Active Community: Python has a large and active community, which means you'll have access to numerous resources, libraries, and support when building your application.

Maintenance: Python's readability and maintainability make it easier to manage and update your code in the future.