Banking Mobile

Final Review

CS - 4850 - 02

Fall 2025

Professor Sharon Perry

November 22, 2025

SP-26 Blue Budget

Charles LaPlatney

Kevyn Lopez

Camen McDaniel

Zoe Faju

Sarbani Adhikari

Table of Contents

Github Link: Link - https://github.com/1e1ouch/Budget-App

Website Link: Link - Blue Budget - CS 4850 Project

1.Introduction

The goal of the project is to design and implement a personal finance tool that allows users to view account information, build simple budgets, and track their spending habits through an intuitive and easily navigable mobile interface. Blue Budget was built using Flutter and Dart for cross-platform compatibility, offering the ability to run on both Android and iOS emulator environments.

The primary objective of the app is to simplify budgeting for users who want a clear overview of their finances without the complexity found in traditional financial applications. To support this, the project uses Plaid’s Sandbox API to simulate secure bank account linking and retrieve mock account data. While this prototype does not connect to live financial institutions, the integration models how a production version would operate, providing a foundation for future expansion.

The Blue Budget team approached the project using a software development lifecycle (SDLC) structure, beginning with requirements gathering, followed by architectural design, development, and testing. Each phase contributed to building a working prototype that demonstrates core functionality such as user login, navigation between financial screens, adding budget items, and displaying account summaries.

This report summarizes the entire development process, from initial planning through final testing. The report also documents the tools, technologies, and source code structure used throughout the project. The goal is to provide a clear and complete overview of how Blue Budget was designed and implemented, as well as to highlight opportunities for future enhancements.

2.Requirements

2.1 Functional Requirements

Login and Account Creation

- Launch Page
  - The log in button redirects the user to the login page
  - Register the new account on the Create account page
- Login with username and password
- Password recovery through personal questions / email (optional)
- Create Account page
  - Phone
  - Last name
  - First name
  - Finish account setup

Login Page

- User logs in with Login button - enter credentials
- Password recovery via personal questions

Display Home Page

- Show financial dashboard overview
  - Display current account balances
  - Show recent transactions summary
- Navigation options to main features

Navigate to Profile Setup

- Complete user profile information
- Set financial preferences and categories
- Choose default settings
- Tutorial walkthrough for new users

Navigate to Bank Account Connection

- Connect bank accounts through Plaid integration
- Verify Plaid accounts
- Import initial transaction history

Navigate to Income Setup

- Add income sources and amounts
- Set recurring income schedules
- Categorize different income types (Active Income, investment, benefits, etc.)
- Edit or update income information

Navigate to Expense Tracking

- Add new expenses manually
- Categorize spending by type
- View and edit existing transactions

Navigate to Budget Creation

- Create spending limits by category
- Set monthly or weekly budget targets
- Allocate funds across different areas
- Save and activate budget plans

Navigate to Goal Setting

- Create new financial goals
  - Set target amounts and deadlines
  - Choose goal categories (savings, debt, etc.)
  - Track and monitor goal progress

Navigate to Visual Reports

- View spending charts and graphs
- Generate financial reports by time period specified
- Analyze spending patterns and trends

2.2 Non-Functional Requirements

Security

- Secure authentication system with password strength requirements
- Compliance with banking security standards and regulations
- Automatic session timeout after periods of inactivity
- Mandatory password complexity requirements

Capacity

- Handle storage of transaction history for up to 2 years
- Process data synchronization from banking sources
- Manage medium sized datasets for chart and graph generation
- Store user goals and progress tracking data efficiently

Usability

- Clear interface design suitable for users
- Clear visual navigation throughout the application
- Accessible design following mobile accessibility guidelines
- Simple onboarding process for new user setup

Other

- Cross platform compatibility for iOS devices
- Regular automatic backups of user financial information

2.3 External Interface Requirements

User Interface Requirements

- The user interface should be properly labeled to their respective headings. The charts will be color coordinated and easy to read.
- The headings will be broken down to weeks, months, and years. Customized categories are also available to users, alongside notification preferences.
- Will come in light and dark mode (can match the users OS)
- To create an app that is easy to use, the interface includes a search bar and is accessible to users.
- Provide customizable push notifications to remind users of upcoming bill payments, saving goals, or unusual spending activity.

Hardware Interface Requirements

- The application should work on multiple mobile devices that run on iOS version 10 and above and Android version 9 and above operating systems.
- Touch screen support.
- Accommodates different screen sizes and resolutions.

Software Interface Requirements

- To make sure the application is functionable, it will integrate with third-party APIs and libraries.
- Compatibility with cloud services are required to store private and confidential information, like bank information, passwords, emails, and even user goals. The cloud services will also store backup data.

Communication Interface Requirements

- For maintaining data and ensuring privacy, the application will use secure data exchange pathways between the device and the hosting server.
- TCP/IP protocols will be implemented to make sure the data from clients and servers are correct.

3.Analysis / Design

3.1 System Architecture Overview

Presentation layer

- Uses appState as a way to hold user data and to update the apps widgets in real time.
- Has different pages that hold the budget, goals, profile, and transactions

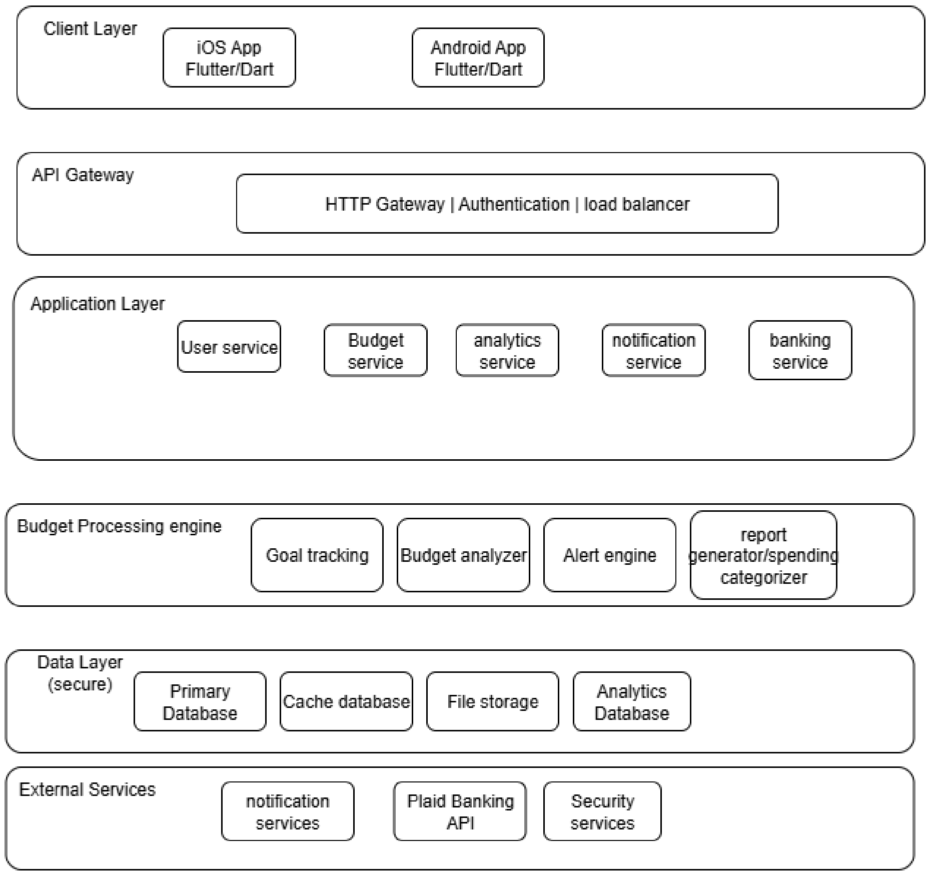
Application Logic Layer

- Processes API responses and maps JSON into Dart objects
- The budget calculations, navigation, and form validation is handled.

Backend Integration Layer

- Retrieves the checking and saving account information
- Create Plaid link tokens
- JSON responses are sent back to Flutter to display

Fig.1 System Architecture



3.2 UI/UX Design Overview

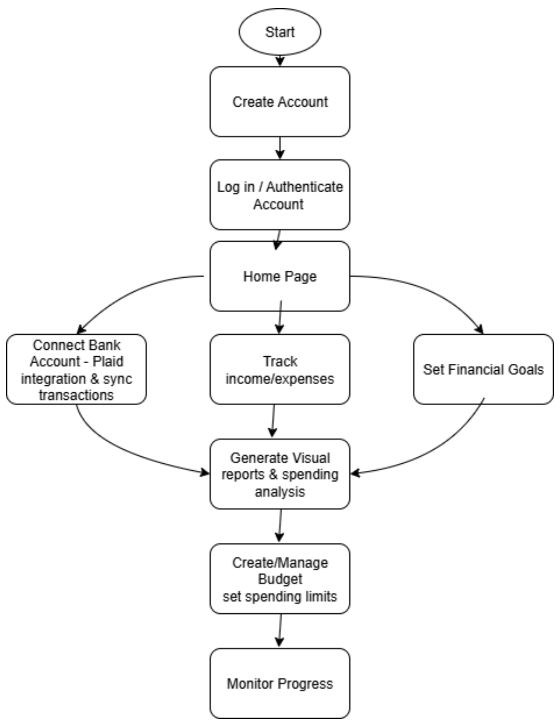
Interface

- Transaction: This display includes the date, category, and amounts
- Profile: Allows user to personalize their long-term financial objective
- Dashboard: Summary of the balance, spending, and activity
- Budget: Includes a progress bar for each category.
- The navigation is at the very bottom of the screen for easy access

3.3 Data Flow Design

1. The app is opened by users → Splash → Home Page
2. Users chooses to connect to their bank → Plaid link process
3. The server requests the accounts from Plaid
4. Plaid returns JSON → Node → Flutter
5. The dashboard and transaction screens are updated by Flutter
6. Add budget, add transaction (user actions) are updated by appState.

Detailed System Design


Start
Create Account
Log in / Authenticate Account
Home Page
Connect Bank Account - Plaid integration & sync transactions
Track income/expenses
Set Financial Goals
Generate Visual reports & spending analysis
Create/Manage Budget set spending limits
Monitor Progress

4.Development Summary

Blue Budget App is a financial mobile application tool that will help users take control of their finances regardless of whether or not they have any financial literacy. The app will present the users with intelligent tools such as expense tracking, customizable budgets, and actionable financial insights. The application thoroughly addresses the common challenge of financial management that may benefit any user over the age of 18.

4.1 Development Implementation

Main App – "Blue Budget"
Built with Flutter (Dart) for state management with five main screens: **Dashboard**, **Budget**, **Transactions**, **Goals**, and **Profile**.

Dashboard
Displays account balance, total monthly spending, and recent transactions. Uses 'monthlytotals' to calculate net income. Transactions update when Plaid Sandbox data is linked.

Budget Screen
Shows budgeted, spent, and remaining totals with progress bars for each category: Groceries, Dining, Rent, and Transfer. Indicates overspending in red and remaining funds in green. Updates in real time through appState.

Transactions Screen
Lists each transaction with merchant, date, category, and amount. Replaces seeded data with Plaid Sandbox transactions after linking.

Backend / Plaid Integration
Node/Express backend connects to Plaid Sandbox API using demo credentials Returns sample checking-account data to the Flutter app. Tested on macOS and Windows.

Website
A simple project website will display app details, screenshots, and team information. Planned host: **GitHub Pages**, with Google Sites as backup.

Ongoing Work
Add category charts and goal visuals, allow user-created categories, refine the UI to make it more visually engaging, and finish the website before the prototype presentation. Each of the transactions will need a description, amount, and date. The categories will be customizable with rent, income, groceries etc as default. The add button will be prominent and will have recurring expenses like rent and subscriptions. You can filter the app by dates and categories.

4.2 Database Connection

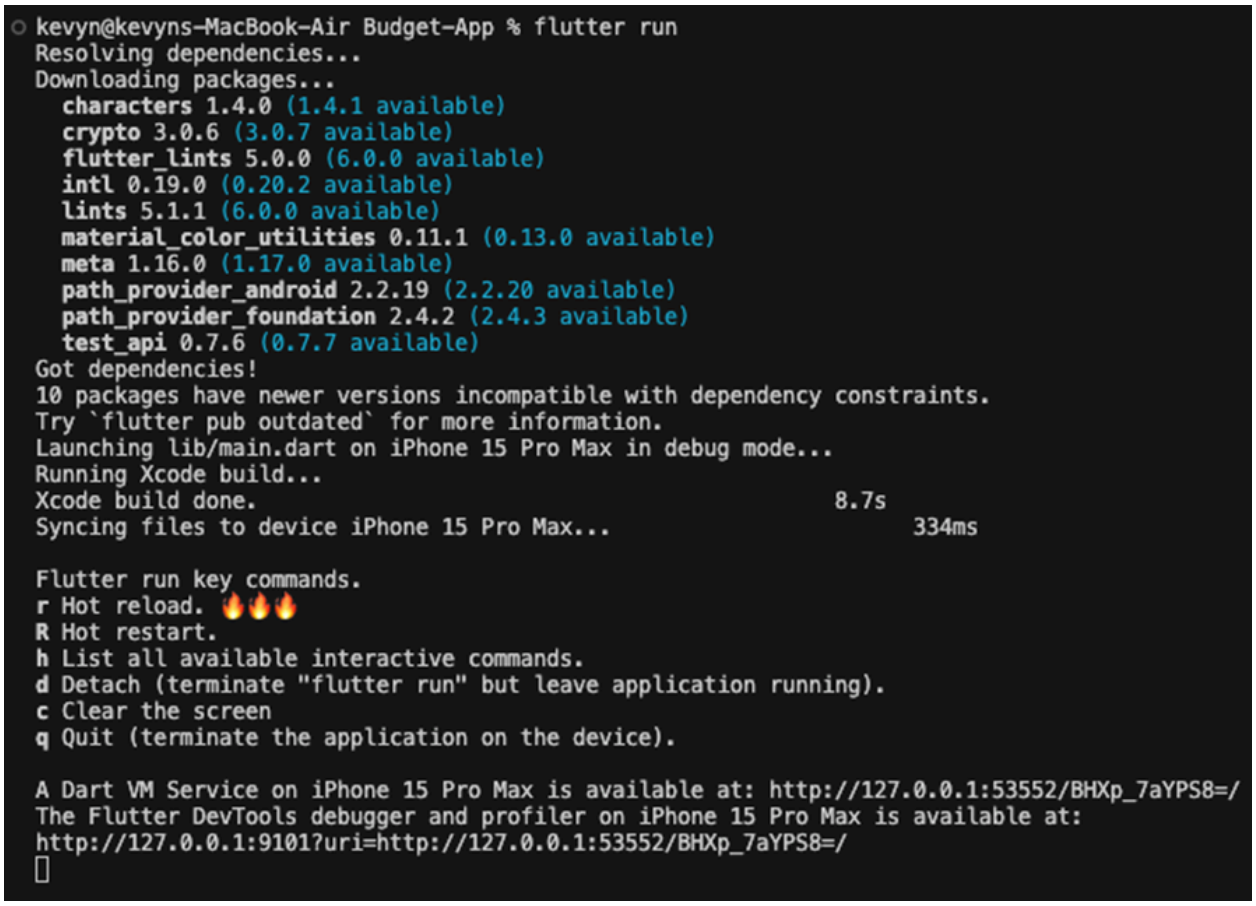
Plaid API / AppState
No database currently in use. Data flows from Plaid → Node/Express → Flutter

4.3. How to Set Up the Project

Overview
Mobile budgeting app for iOS, Android, and web. Uses Flutter for UI and Node/Express backend for Plaid data.

Requirements
Flutter SDK, Node.js (LTS), Xcode/iOS Simulator for macOS, or Android Studio for Android. Optional: VS Code with Flutter and Dart extensions.

Fig.3 Setup

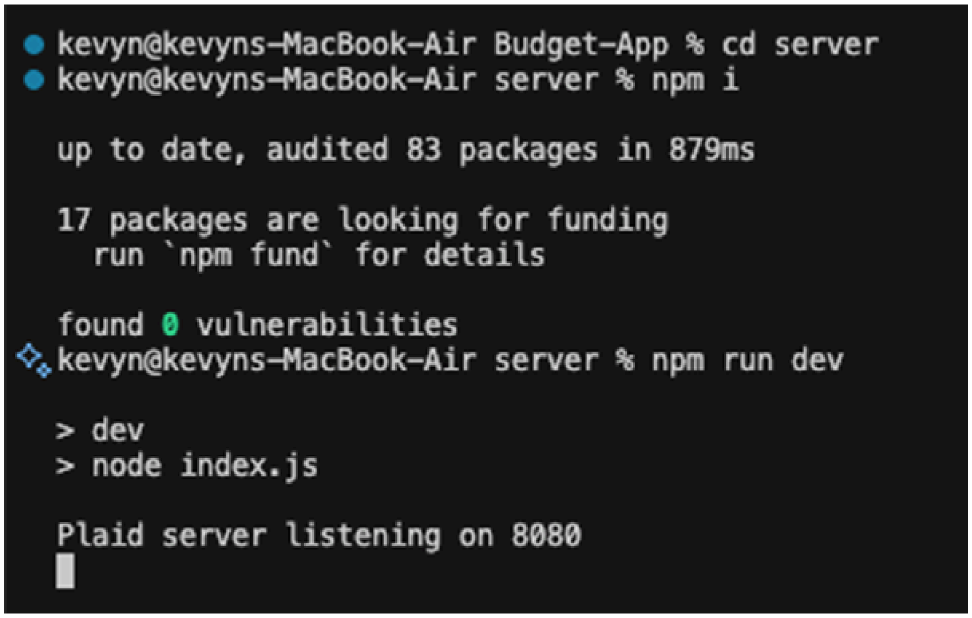


```
○ kevyn@kevyns-MacBook-Air Budget-App % flutter run
  Resolving dependencies...
  Downloading packages...
    characters 1.4.0 (1.4.1 available)
    crypto 3.0.6 (3.0.7 available)
    flutter_lints 5.0.0 (6.0.0 available)
    intl 0.19.0 (0.20.2 available)
    lints 5.1.1 (6.0.0 available)
    material_color_utilities 0.11.1 (0.13.0 available)
    meta 1.16.0 (1.17.0 available)
    path_provider_android 2.2.19 (2.2.20 available)
    path_provider_foundation 2.4.2 (2.4.3 available)
    test_api 0.7.6 (0.7.7 available)
  Got dependencies!
  10 packages have newer versions incompatible with dependency constraints.
  Try `flutter pub outdated` for more information.
  Launching lib/main.dart on iPhone 15 Pro Max in debug mode...
  Running Xcode build...
  Xcode build done.                                            8.7s
  Syncing files to device iPhone 15 Pro Max...                       334ms

  Flutter run key commands.
  r Hot reload. 🔥🔥🔥
  R Hot restart.
  h List all available interactive commands.
  d Detach (terminate "flutter run" but leave application running).
  c Clear the screen
  q Quit (terminate the application on the device).

  A Dart VM Service on iPhone 15 Pro Max is available at: http://127.0.0.1:53552/BHXp_7aYPS8=/
  The Flutter DevTools debugger and profiler on iPhone 15 Pro Max is available at:
  http://127.0.0.1:9101?uri=http://127.0.0.1:53552/BHXp_7aYPS8=/
```

Figure 4. Install & Run



```
● kevyn@kevyns-MacBook-Air Budget-App % cd server
● kevyn@kevyns-MacBook-Air server % npm i

  up to date, audited 83 packages in 879ms

  17 packages are looking for funding
    run `npm fund` for details

  found 0 vulnerabilities
  kevyn@kevyns-MacBook-Air server % npm run dev

  > dev
  > node index.js

  Plaid server listening on 8080
  █
```

Install and Run

git clone https://github.com/1e1ouch/Budget-App.git

cd Budget-App

flutter pub get

open -a Simulator

flutter devices

flutter run

Plaid Test Login

1. Open "Connect Accounts."
2. Log in with credentials.

3. Select Checking and complete MFA.
4. Dashboard and Transactions update with Plaid data.

4.4 Monetization Plan

- **Revenue Options**
  Free version with banner ads and optional ad-free premium version. Possible paid upgrades for advanced analytics or custom categories. Possible “Support Us” donation button option. Sponsored financial tips or affiliate links as future additions.

5.Test (plan and report)

5.1.Test Objectives

The objective of the Software Test Plan is to test and verify that major features of the mobile banking application function correctly and meet expected behavior.

Primary objectives:

- Validate that the user login screen accepts valid credentials and handles invalid input correctly.
- Verify successful navigation between main pages (Home, Accounts, Budget, Savings).
- Confirm that Plaid test-link integration initializes and returns mock account data.
- Validate basic budgeting calculations (expense totals, category handling).
- Ensure UI layout displays correctly on common mobile screen sizes.
- Identify and document bugs, severity, and pass/fail status.

5.2.Scope Testing

5.2.1.In-Scope:

- User login and authentication flow (mock credentials).
- Navigation between pages (Home, Accounts, Budgets, Savings).
  Plaid sandbox link token creation and mock account retrieval.

- Basic financial functionality:
    - Adding budget values
    - Viewing account summary data
- UI responsiveness and layout validation.
- Error-handling scenarios such as empty required fields.

5.2.2.Out-of-Scope:

- Real bank account connections (production Plaid).
- Performance benchmarking or load testing.
- Security penetration testing.
- Full database persistence (prototype uses mock/local data).
- Complete visual design testing (animations, advanced styling).

5.3.Test Cases:

5.3.1. Functional Scenarios:

Test Case ID	Description	Expected Result
TC-01	Login with valid credentials	User is taken to home screen
TC-02	Login with invalid credentials	Error message appears
TC-03	Navigation from Home to Accounts	Accounts page loads successfully
TC-04	Navigation Accounts to Budget	Budget page loads
TC-05	Create Plaid link token	Link token returns “sandbox” response
TC-06	Retrieve mock account list	Accounts populate on screen

TC-07	Add a new budget value	Budget updates without crash
TC-08	Attempt Action with empty fields	Displays a validation message
TC-09	Manually insert transaction	Transactions updated successfully

5.3.2. Non-Functional Scenarios:

Test Case ID	Description	Expected Result
NFTC-01	UI layout on iPhone 14 emulator	All components visible and accessible
NFTC-02	UI layout on Pixel 6 emulator	No overlapping or clipped UI
NFTC-03	App startup time under 3 seconds	Splash + Home load are within threshold
NFTC-04	System handles no internet during Plaid connection	Error message displayed

5.4. Test Procedures

TC-01: Login with valid credentials

Launch the Blue Budget app.
Enter username “testuser.”
Enter password “password123.”
Tap the Login button.
Expected Result: User is redirected to the Home screen.

TC-02: Login with invalid credentials
Steps:
Launch the app.
Enter an incorrect username or password.
Tap the Login button.
Expected Result: An error message appears; user remains on the login screen.

TC-03: Navigate from Home to Accounts page
Steps:
Log in to the app.
Tap the Accounts button on the Home screen.
Expected Result: The Accounts screen loads successfully.

TC-04: Navigate from Accounts to Budget page
Steps:
Open the Accounts screen.
Tap the Budget navigation option.
Expected Result: The Budget screen loads without errors.

TC-05: Generate Plaid link token
Steps:
Go to the Accounts screen.
Tap the Connect Bank button.
Wait for the app to request a link token.
Expected Result: The app receives a sandbox link token and begins the Plaid flow.

TC-06: Retrieve mock account list from Plaid
Steps:
Complete the Plaid test login using sandbox credentials.
Allow the app to retrieve mock account data.
Expected Result: Mock account names and balances appear on the Accounts screen.

TC-07: Add a new budget entry
Steps:
Navigate to the Budget screen.
Tap Add Budget Item.
Enter a category name.
Enter a budget amount.
Tap Save.
Expected Result: The new budget entry appears in the budget list.

TC-08: Validate empty required fields
Steps:
Leave one or more required fields blank on a form.

Tap Login or Save.
Expected Result: App displays a validation message and prevents the action.

NTC-01: UI layout on iPhone 14 simulator
Steps:
Launch the app on the iPhone 14 simulator.
Navigate through all screens (Home, Accounts, Budget, Savings).
Expected Result: All UI elements display correctly with no overlap.

NTC-02: UI layout on Pixel 6 emulator
Steps:
Launch the app on the Pixel 6 emulator.
Navigate through each main page.
Expected Result: Layout is responsive and readable on Android.

NTC-03: App startup time performance test
Steps:
Close the app completely.
Launch the app again.
Measure time until the Home screen appears.
Expected Result: App loads within 3 seconds.

NTC-04: No internet error handling for Plaid connection
Steps:
Disable internet on the emulator.
Open the Accounts screen.
Tap Connect Bank.
Expected Result: App displays an error message and does not freeze.

5.5 Test Environment

5.5.1. Hardware:

Android Emulator (Pixel 6, API 34)
iOS Simulator (iPhone 14, iOS 17)

5.5.2. Software:

Flutter 3.x

Dart SDK
Android Studio / Xcode
Plaid Sandbox API
Local mock JSON data for budgeting features

5.5.3. Network:

Wi-Fi for Plaid test API requests
Offline mode testing toggled manually

6. Test Data

- Login Credentials: testuser / password123
- Plaid User ID: demo-user-123
- Mock Account Data:
  - Checking: $1,200
  - Savings: $3,400
- Budget Categories: Food, Rent, Transportation
- Budget Values: $200, $900, $150

7. Software Test Report

Requirement	**Pass**	**Fail**	**Severity**
Create account	No	Yes	N/A (Not Implemented)
Login	No	Yes	N/A (Not Implemented)
Password recovery	No	Yes	N/A (Not Implemented)
Accounts Navigation	Yes	No	Minor
Mock Account Retrieval	Yes	No	Minor
Plaid Link Token Generation	Yes	No	Moderate

8.Version control

The Blue Budget uses GitHub as the repository and Git for version control. The team found this to be the easiest way because, easy to see the history and updates and who made them, can test out new features without affecting the main and revert to previous versions. The main branch in the GitHub repository stores the code that is used for testing. This approach allows the team to work simultaneously on different parts of the project. Using GitHub was beneficial in the sense that a backup was provided and that made the whole collaboration project smoother because the people were able to look back at the changes and minimized the risk of losing work.

9. Summary

In summary, the goal of our project is creating a budgeting app that is designed to assist users with managing their finances. The app’s interface provides the user with the ability to use visual aids such as charts and graphs to help users understand spending and decisions, setting goals and monitoring progress, by utilizing visual aids such as charts and graphs to help users understand decisions and spending. This will help the average user manage their finances better with little to no financial experience needed.