

SP-26 Blue — Budget Mobile App

Final Report Draft





CS 4850 - Section 02 – Fall 2024

November 14, 2024 - Professor Perry

Website: <https://sp26bluebudgetapp.github.io/>

GitHub: <https://github.com/SP26BlueBudgetApp>

Lines of Code: **7098** Components: **8** Total man Hours: **189**

 Alvis Wheeler Team Leader	 Tyrell Jones Developer
 Jamieson Cannon Developer	 Andrew Taylor Documentation

Team Members:

Name	Role	Cell Phone / Alt Email
Alvis Wheeler	Team Leader	678.468.2411 Awheel41@students.kennesaw.edu
Tyrell Jones	Developer	678.830.3236 Tjljlv22@gmail.com
Andrew Taylor	Documentation	804.245.2545 Atayl336@students.kennesaw.edu
Jamieson Cannon	Developer	404.574.3672 Jcanno45@students.kennesaw.edu
Sharon Perry	Project Owner or Advisor	770.329.3895 Sperry46@kennesaw.edu

Table of Contents

1. Introduction	5
2. Requirements.....	5
2.1. Functional Requirements	5
2.1.1. Login	5
2.1.2. Register	5
2.1.3. Authentication	5
2.1.4. Display Home Page	5
2.1.5. Navigation Pane	5
2.1.6. Mobile Friendly	6
2.1.7. Database.....	6
2.1.8. Budgeting	6
2.2. Non-Functional Requirements	6
2.2.1. Security	6
2.2.2. Usability	6
2.2.3. Reliability.....	7
2.3. External Interface Requirements	7
2.2.1. User Interface Requirements.....	7
2.2.2. Software Interface Requirements	7
2.2.3. Communication Interface Requirements	7
3. Analysis	7
3.1. Login/Signup.....	7
3.2. Home page	7
3.3. Database.....	8
3.4. Budget.....	8
3.5. Report	8
3.6. Settings	8
4. Design	8

4.1.	Development Methods	8
4.2.	Assumptions and Dependencies	9
4.3.	System Architecture	9
4.4.	Architectural Strategies	11
4.5.	Detailed System Design	11
4.5.1.	Classification	11
4.5.2.	Resources	12
5.	Development	13
5.1.	Mobile Application	13
5.1.1.	Flutter and Dart	13
5.1.2.	Android/iOS	13
5.2.	Database	13
5.2.1.	Firebase Authenticator	13
5.2.2.	Firestore	13
5.3.	Interactions	14
5.3.1.	Budget Goals	14
5.4.	Plaid	14
5.4.1.	User Bank Linking	14
5.4.2.	Transaction Data	14
5.4.3.	Income Data	14
6.	Test Plan	16
7.	Version Control	17
8.	Summary	17
9.	Appendix	18
9.1.	Login Screen	18
9.2.	Signup Screen	18
9.3.	Home Screen	19
9.4.	Budget Screen	19
9.5.	Report Screen	20

9.6. Settings Screen 20

9.7. Tutorial Completion..... 21

1. Introduction

This document details our process that was used to build a budgeting app. We used Flutter and Dart, which empowers users to manage their finances seamlessly across platforms (Android and IOS). This app includes essential features for creating and managing budgets, tracking expenses, and monitoring financial goals, all within an intuitive, user-friendly interface. With real-time expense tracking, category-based budgets, and helpful notifications, the app is set to deliver practical tools for financial health in one cohesive package. By leveraging Figma for the design phase, we created detailed mockups to ensure we knew what the app should look like.

2. Requirements

2.1. Functional Requirements

2.1.1. Login

- Login with Username and Password

- Password Recovery

2.1.2. Register

- Register an account with:

 - Email

 - Password

 - First and Last Name

 - Phone Number

2.1.3. Authentication

- 2-factor Authentication via Email

2.1.4. Display Home Page

- Welcome screen with charts

- Transaction history

2.1.5. Navigation Pane

- Navigate to different pages

Recent Transaction

User Profile

Budgeting Feature

Settings Tab

2.1.6. Mobile Friendly

Usable on any iOS or Android device

2.1.7. Database

Firebase authentication

Firestore data storage

2.1.8. Budgeting

Total Monthly Expenses

Spending Habits

Purchase Breakdowns

Set budget goals

Graphs/Charts

Plaid Integration

2.2. Non-Functional Requirements

2.2.1. Security

The app ensures the security and privacy of user data by implementing industry-standard encryption for data at rest and in transit. It should provide two-factor authentication (2FA) options, such as email verification to enhance account security. Additionally, user sessions must have secure timeout mechanisms, and failed login attempts should trigger account lockouts and notify the user of suspicious activity.

2.2.2. Usability

The app provides an intuitive and user-friendly interface that allows users to navigate effortlessly, with clear and consistent design elements. It includes features such as guided onboarding for new users, and responsive layouts that

adapt seamlessly to various screen sizes and orientations. Common tasks, like adding transactions or adjusting budgets are achievable in no more than three taps or clicks.

2.2.3. Reliability

The app is instantly able to fetch a user's banking and set budgeting data at any moment, allowing them to see their progress towards their goals as well as recent transactions (if they have plaid integration setup). The software is embedded with code to make multiple attempts at HTTP/web requests in case a call fails, in order to increase reliability for users.

2.3. External Interface Requirements

2.2.1. User Interface Requirements

The app needs a clean UI that is minimalistic, allowing users to easily navigate and not overwhelm them. Distinct navigation bar with different pages all correlating to their own categories like home, budget, report, and settings.

2.2.2. Software Interface Requirements

The app receives necessary banking information using Plaid, which then sends the data to our servers for it to be displayed on the user's phone. This data is then stored in Firebase's database for each given user.

2.2.3. Communication Interface Requirements

Push notifications whenever budgeting goals have been met. Reminder of what budget goals have been set to follow them. Report notifications biweekly.

3. Analysis

3.1. Login/Signup

Firebase authentication is utilized by taking in user input and passing it through the database. The database has an email and userID attached and runs a match check to see if the user is in the system or not.

3.2. Home page

Integrating plaid user transactions are displayed. Everything is put in one quick access place with things like recent transactions, monthly breakdown, and spending goals.

3.3. Database

Firestore will function as the main database. Using dependencies and models in Flutter/Dart the information is passed to the database where it is stored. Real-time functionalities of Firestore allow seamless data transfer.

3.4. Budget

Users can create their own budget goals. This budget goal has the category of where they want the money spent and how much they want to limit it to. A start date is initialized at creation using current time. Goals can be added or removed at will or checked off for recognition later.

3.5. Report

Short breakdown of a user's income/expenses/savings. Users can edit these at will with a pie chart being generated showing the breakdown. Portions of the pie chart have their own icons and colors to identify what they are.

3.6. Settings

This page allows the user to change their personal settings. Items in this list include account, theme, notifications, and currency. Account enables the user to change things like their email address or first and last name. A sign out button also is positioned at the bottom allowing seamless account switching.

4. Design

4.1. Development Methods

Design Phase: Create wireframes and prototypes to define the user interface and user experience Utilize Flutter's widget library to design an intuitive and aesthetically pleasing interface.

Implementation Phase: Develop the app using Dart and Flutter, focusing on integrating core features and ensuring smooth performance across different platforms.

Testing Phase: Conduct thorough testing, including unit tests, integration tests, and user acceptance testing (UAT), to identify and resolve any issues.

Deployment: Prepare the app for release on the Apple App Store, Google Play Store, and web platforms.

Maintenance and Updates: Monitor user feedback and performance, provide regular updates and improvements based on user needs and technological advancements.

4.2. Assumptions and Dependencies

Building a Flutter app with Dart involves several essential dependencies and assumptions that guide the development process. Flutter supports multi-platform compatibility across iOS, Android, web, and desktop, assuming your app will benefit from a responsive design that works across various devices and screen sizes. Key tools include the Flutter SDK, Dart SDK, and IDE plugins, along with Android and iOS SDKs for mobile deployment.

Flutter has many core libraries that provide essential features, while services like Firebase enable easy backend integration. The app is written in Dart, which requires knowledge of null safety, asynchronous programming.

Flutter's testing framework, along with integration testing on emulators or real devices, is crucial for quality control. For deployment, CI/CD pipelines and adherence to app store guidelines help ensure a smooth release. Flutter's design also aims for efficient layouts, and strong error handling for good user experience.

4.3. System Architecture

The system is partitioned into several high-level systems, each responsible for distinct aspects of the application's functionality. This architectural composition allows for modular development, scalability, and ease of maintenance. The main subsystems and their interactions are outlined below:

User Interface (UI)

Application Layer

Data Management

Authentication and Authorization

Integration Layer

Analytics and Reporting

Administration and Maintenance

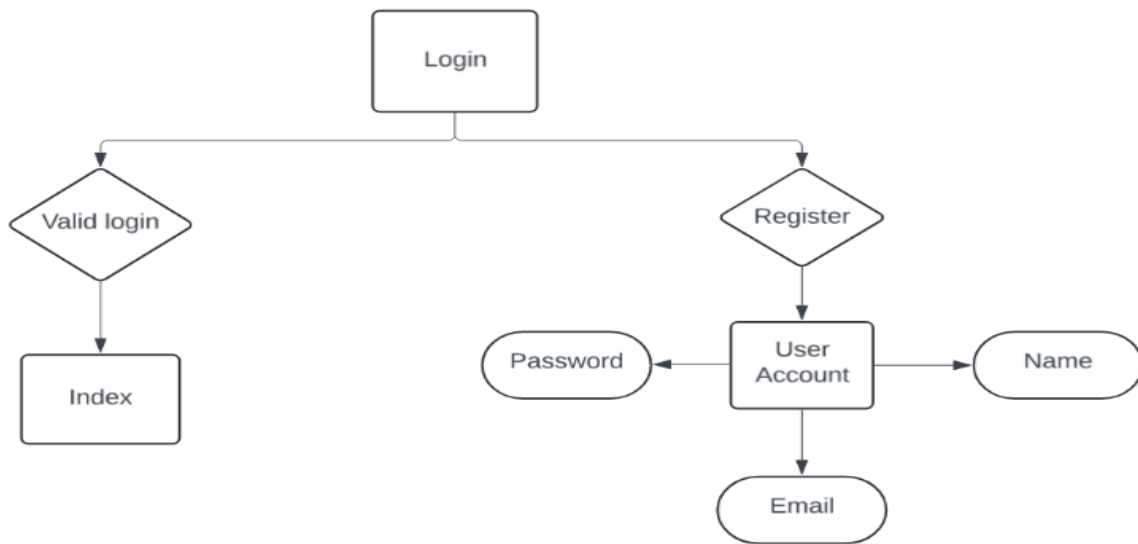


Figure 1: Login interaction

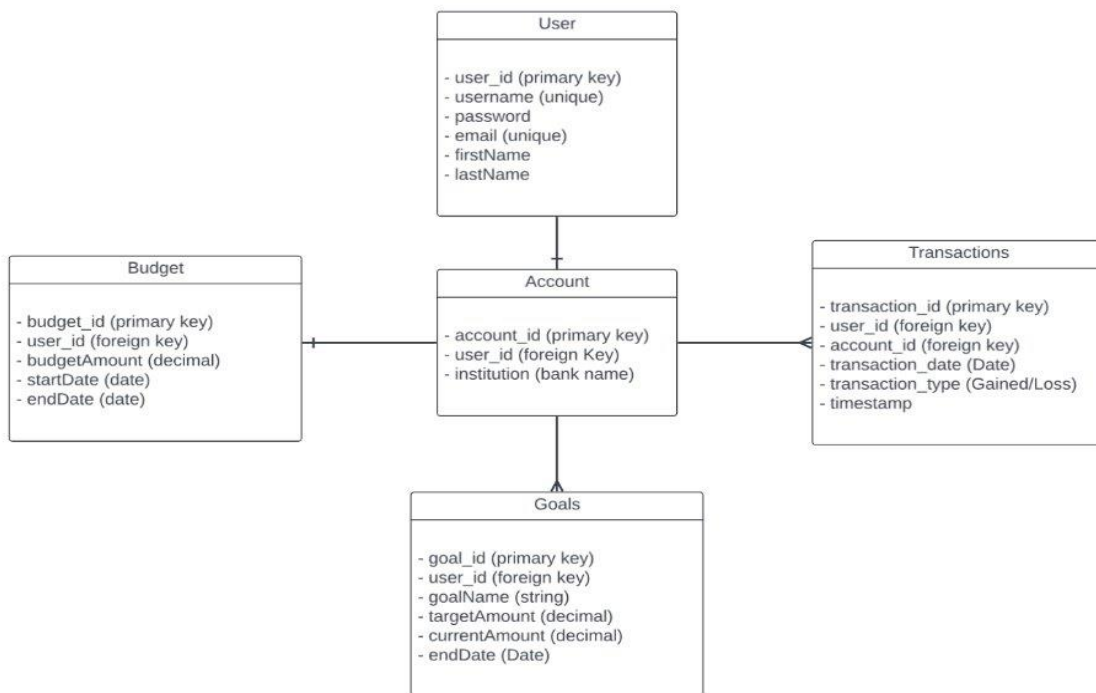


Figure 2: Database layout

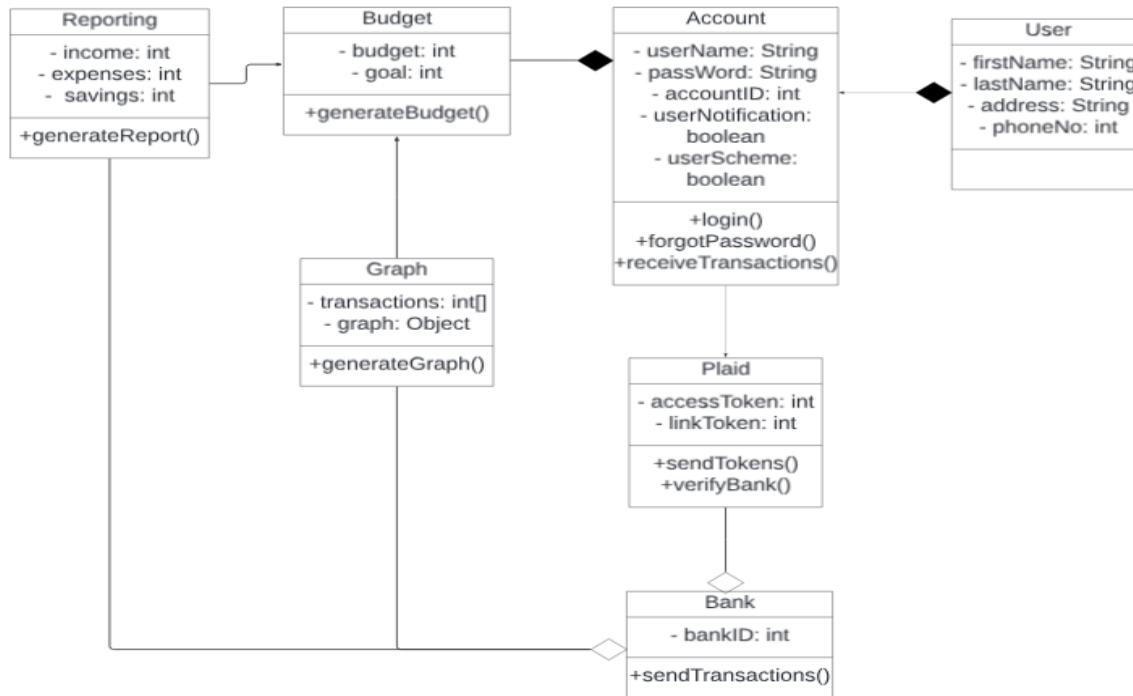


Figure 3: UML Class Diagram

4.4. Architectural Strategies

Dart and Flutter were used to design/make the prototype and finalize the app for our project. This will work as our programming language allowing us to use its libraries to make building the app simplistic.

Our database will be located on the Firebase servers. This means our app will communicate with Firebase for things like logging in and signing up for a new account. User data will also be stored on there, so any device can be used to view budgeting information for the account.

4.5. Detailed System Design

4.5.1. Classification

User Accounts and Authentication: Secure user accounts with login and registration functionalities.

Budget Creation and Management: Tools for setting up and adjusting budgets for various categories (e.g., groceries, entertainment).

Expense Tracking: Easy input and categorization of daily expenses, with options to attach receipts or notes.

Notifications and Reminders: Alerts for upcoming bills, overspending in categories, and budget adjustments.

Cross-platform Support: Seamless performance across iOS, Android, and web platforms.

4.5.2. Resources

Memory (RAM)

Description: Memory is used for storing application state, user data, and transient information during runtime.

Management: Dart helps manage memory automatically by reclaiming unused objects.

Potential Issues: Memory leaks can occur if references to unused objects are not properly removed. This can degrade performance and lead to crashes.

Processors (CPU)

Description: The CPU executes Dart code and handles concurrent tasks.

Management: Flutter applications are designed to be responsive and handle UI updates efficiently.

Potential Issues: CPU-bound tasks can lead to performance bottlenecks if not managed correctly.

Database

Description: Used to store user data, including transactions, budgets, and preferences with Firebase.

Management: Data access is managed through database libraries and APIs.

Network Resources

Description: Includes APIs for fetching financial data, bank integration, and backend services.

Management: Network requests are handled using HTTP or other network libraries. Error handling and retry mechanisms are implemented to manage network failures or slow responses.

5. Development

5.1. Mobile Application

5.1.1. Flutter and Dart

Flutter and Dart was utilized to build our mobile application. Dart acts more like a programming language with syntax like Java/JavaScript. Flutter acts as the framework allowing widgets and states that Dart then utilizes. Both work in unison for the development of the app.

5.1.2. Android/iOS

The app must be cross-compatible, so Flutter and Dart were chosen to build the app. Inherently, both versions can be built simultaneously with only the Firebase application being different.

5.2. Database

5.2.1. Firebase Authenticator

This works as our verification for users to enter the application. A controller with a text field allows the user to enter their email address and password when logging in or signing up. Firebase has a built-in dependency on Flutter which utilizes streams making a final credit for users. These entered values are passed into a class that checks or creates a user. If signing up, the userID is stored in the authenticator page for logging in later. When credentials do not match the user does not gain access to the app.

5.2.2. Firestore

All user information is stored in this database. In Firestore, a collection was created for the app to place entered data. A model was created that takes in required fields and copies them into a Json which Firestore uses to store data. A service tab then pings the database, using a collection reference, into its correct location. Additional functions like add/update/delete also are found here.

Inside of the app, users press buttons that call these functions editing the database in real-time.

5.3. Interactions

5.3.1. Budget Goals

Users can create their own budget goals for categories of their choosing. They indicate how much they would like to spend and can mark them as complete as well. When completed goals still stay, but by using a long press users can remove goals that are finished or keep them there.

5.4. Plaid

5.4.1. User Bank Linking

Allows a user to sign in and link their bank account. This will allow for later discussed functionality, like transaction and income data.

5.4.2. Transaction Data

Retrieves data from the Plaid API to a user's bank account on the recent transactions of a user. This directly will feed into a user's budgeting goals to track if they are on pace for meeting their set allowances.

5.4.3. Income Data

Retrieves data from the Plaid API to a user's bank account on the monthly income data of a user. This in turn is what the users will be able to edit in the app, setting categories for the different things a user's money is spent on and setting goals on spending limits for those categories.

6. Test Plan Overview

For the test plan, user testing was completed running the app for each of the requirements. A spreadsheet was made that listed our requirements and populated during user testing. Passes and fails were marked with an X accompanying the requirements respective function. In the case of a failure, a severity was added indicating how important that requirement was and needed fixing. High severity means that the requirement is groundbreaking with low being not a key issue.

Requirement	Pass	Fail	Severity
Login			
Sign Up			
Forgot Password			
2-Factor Login			
SignUp error message			
FireBase Integration			
Sign Out			
Navigation Pane			
Plaid Integration			
Bank Transactions			
Create Budget Goals			
Remove Budget Goals			
Update Budget Goals			
Breakdown of Purchases			
Budget			
Graphs/Charts			
Create income/expenses/ savings			
Change Theme			
Change First and Last name			
Change Notification Settings			
Android and IOS Compatibility			

7. Test Plan Report

Requirement	Pass	Fail	Severity
Login	X		
Sign Up	X		
Forgot Password		X	Medium
2-Factor Login		X	Medium
SignUp error message		X	Medium
FireBase Integration	X		
Sign Out	X		
Navigation Pane	X		
Plaid Integration		X	Highest Level
Bank Transactions		X	High
Create Budget Goals	X		
Remove Budget Goals	X		
Update Budget Goals	X		
Breakdown of Purchases	X		
Budget Graphs/Charts		X	High
Create income/expenses/ savings		X	High
Change Theme		X	Low
Change First and Last name		X	Low
Change Notification Settings		X	Low
Android and IOS Compatibility	X		

8. Version Control

To manage the various features being worked on simultaneously by our team, we utilized Git and GitHub as our version control. All team members were added to a GitHub team, installed Git into their IDE's, and set up their components as a Git repository. This allowed us to all work on different features, then merge our progress together along the way. Also, by using Git and syncing our work to GitHub, there is access to all previous versions of each of our contributions, meaning versions can be skipped back to if new issues arise, and need to go back to a known working state.

Link to team repositories: [SP26BlueBudgetApp repositories](#)

9. Summary

This report outlines the scope of the senior project. It explains the requirements, analysis, design, how it was managed and built using an SDLC methodology, including conducting a test and publishing the results. Overall, using a plethora of tools, widgets, and dependencies, we learned how to make an app utilizing Dart and Flutter learning the framework as a whole.

10. Appendix

10.1. Login Screen



Login

Already have an account? [Signup](#)

10.2. Signup Screen



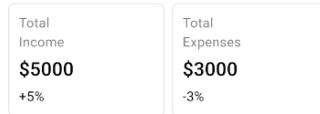
Signup

Already have an account? [Login](#)

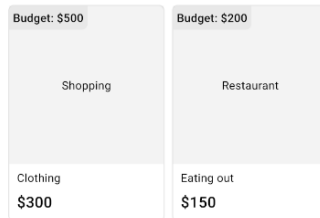
10.3. Home Screen



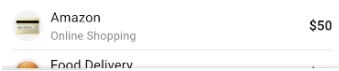
Monthly Breakdown



Spending Goals



Recent Transactions



10.4. Budget Screen

← Budget



Add a new BudgetGoal first



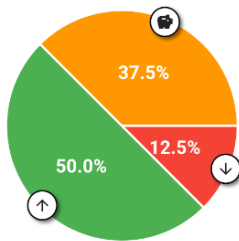
10.5. Report Screen

← Report

Income: \$400

Expense: \$100

Savings: \$300



10.6. Settings Screen

Settings

⌵ Account Settings

🔒 Privacy Settings

📄 Theme Settings

🔔 Notification Settings

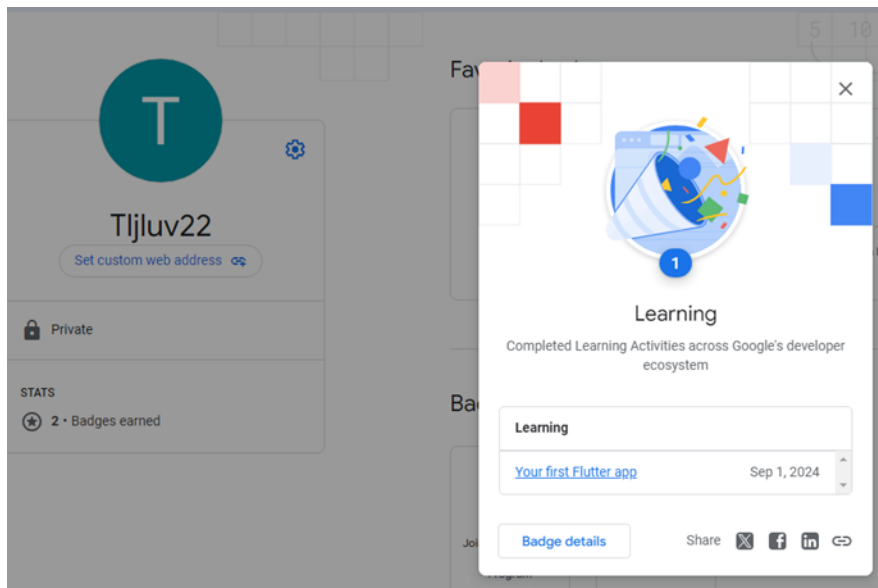
💵 Currency

USD EUR GBP ✓ CAD

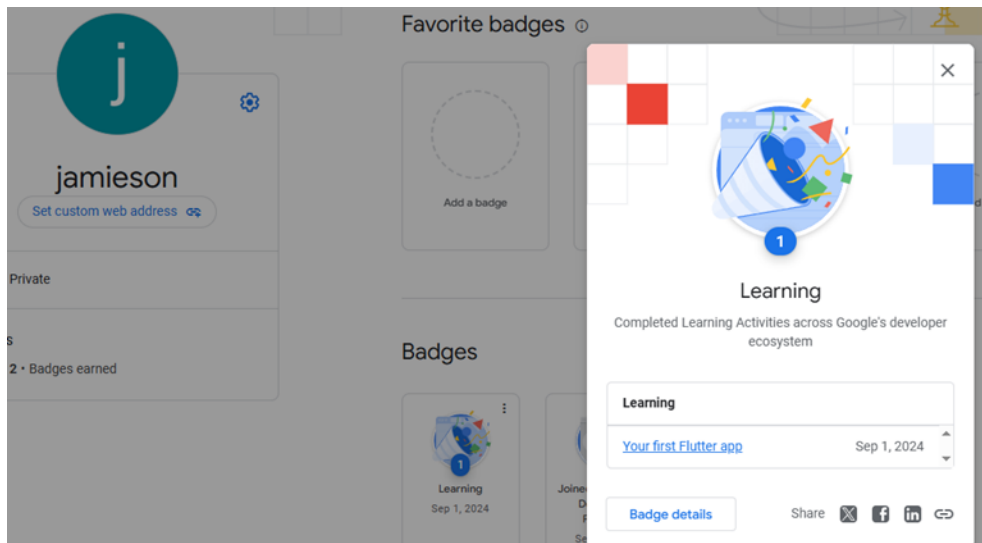
Sign Out



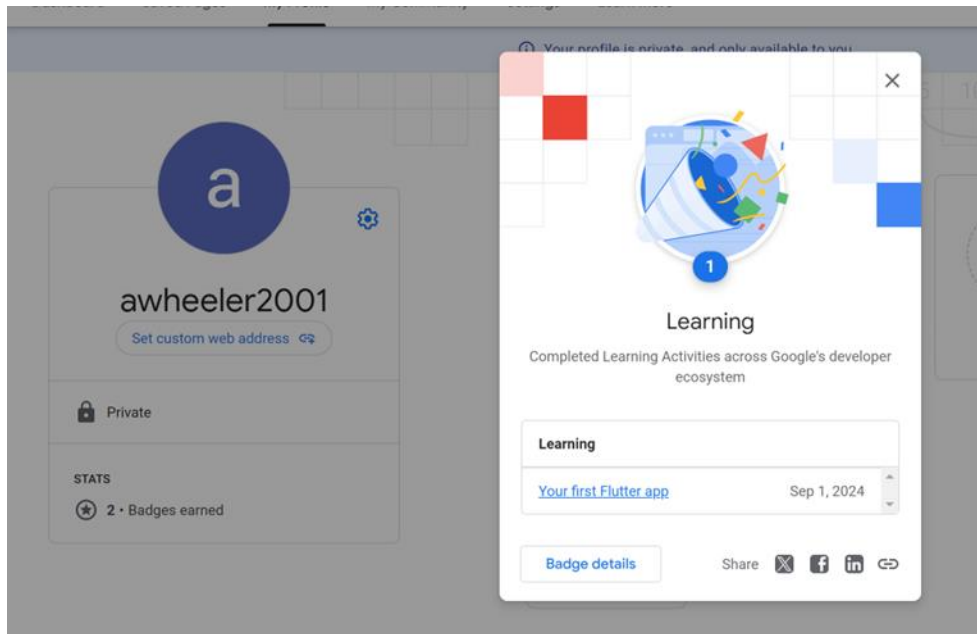
10.7. Tutorial Completion



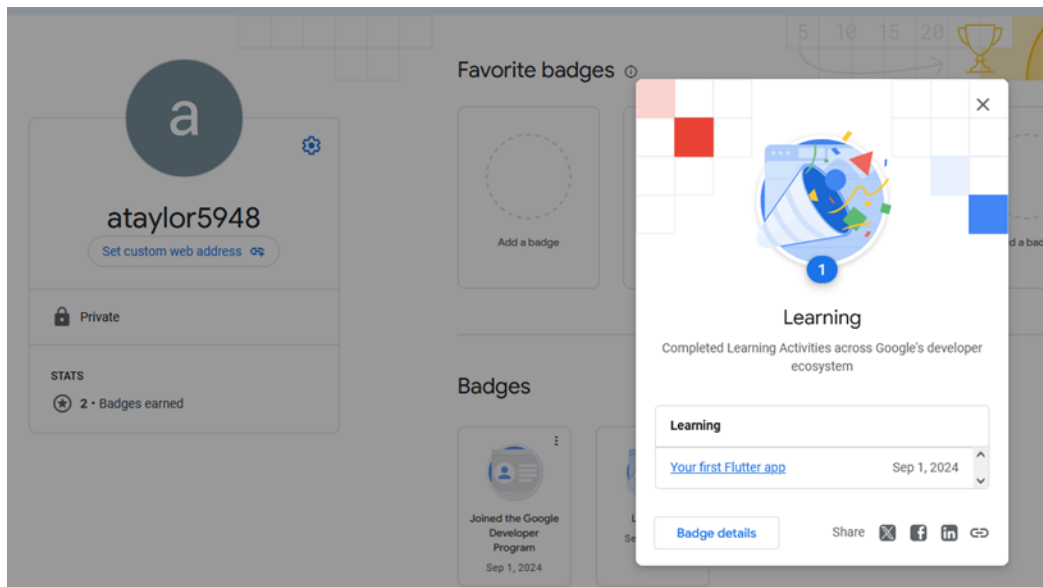
Tyrell Jones



Jamieson Cannon



Alvis Wheeler



Andrew Taylor