SP-26 Red Budget - Final Report

CS 4580 - 04 Fall 2025
Sharon Perry - 12/8/2025
Authors: John Nguyen, Reynaldo Lechuga, Sameer Khan, Taylor Thompson, Yasmeen Issa

https://yasmeeni-0.github.io/BudgetWise/

https://github.com/SP-26-Red-Budget/BudgetWiseApp

Stats and Status	
Lines of Code	4022
Number of Components/Tools	28 Components / 9 Tools (Complete List in the Appendix)
Total Man Hours Estimate	615 hours
Total Man Hours Actual	600 hours
Status	85% complete

Table of Contents

Introduction

The primary goal of BudgetWise, a cross-platform mobile budgeting app, is to assist college students in better understanding, managing, and improving their personal finances. Due to inexperience, a lack of financial education, and the difficulty of juggling income with ongoing costs like rent, food, transportation, and subscriptions, many students have trouble creating a budget. These issues are addressed by BudgetWise, which gives users a straightforward, well-organized, and aesthetically pleasing method of monitoring their earnings, outlays, and general financial practices.

Although the app is designed for college students, anyone looking for a simple budgeting solution can use it. Users can set budget goals, keep an eye on spending, and view financial data in an easy-to-understand manner with this application. In one location, users can manage credit card, loan, and banking information, track expenditures in several categories, and view summaries of their spending. BudgetWise facilitates better financial decision-making and increases users' awareness of where their money is going by arranging financial data into readable charts and breakdowns. BudgetWise was developed with React Native and Expo, it can function flawlessly on both iOS and Android devices from a single codebase. This strategy guarantees responsive design, consistent performance, and simpler updates in the future. Plaid is integrated to securely connect users' bank accounts and retrieve transaction data, while Firebase is used for analytics, real-time database storage, and secure user authentication.

Together, these technologies make BudgetWise dependable, quick, and secure when managing sensitive financial data. With its user-friendly interface, safe system architecture, and real-time data tracking, BudgetWise seeks to empower users by providing them with greater financial control. The project's objective is to give users of all backgrounds a useful, approachable budgeting tool that promotes long-term planning, financial confidence, and financial responsibility.

Requirements

1.0 Introduction

1.1 Project Scope

The scope includes budget management, transaction categories, visualization of data, and financial literacy resources. The secure, user-friendly system will allow the user to easily begin spending and saving in no time.

Our extended scope as a 5 person team will be to create an accessibility feature so that everyone can be comfortable using our app.

1.2 Assumptions

With this budgeting app we assume that it will be mainly used by college but hope many others will use it as well.

We also assume that those who use this app will have a steady income and multiple expenses per month.

We assume the user should have a basic understanding of a smartphone i.e be able to navigate through the app, manage notification settings, and to be able to input their transactions if needed

We assume the user will have a basic understanding of budget. They must be able to categorize their transactions appropriately and be able to budget (set spending limits) in each of these categories if needed

2.0 Constraints

2.1 Environment

Some environmental constraints include the user's mobile device overheating or losing power causing them to not be able to access the app.

A user’s network connectivity could be an issue by limiting the real time data synchronization of transactions

2.2 System

- The application must not drain user’s phone battery excessively
- The application must be able to run on iOS and Android devices
- The application must be able to run for only single-users only
- The application must have multi-factor authentication (via SMS text code or Face ID)

3.0 Functional Requirements

3.1 Login Page

Users will be able to log in/Create an account, login with username and password. Password recovery, and sign up with Google if they choose so.

3.2 Authentication

In order to confirm the user's identification, an email code will be sent to the email that the user signed up with in order to authenticate the user account.

3.3 Display Home Page

Options from Home Page

- Home page
- User portfolio tab
- Spending/budgeting tab

3.4 Navigate to X Page

We have included some mock ups of what these pages will look like.

3.4.1 Launch Page

3.4.1.1 Log in

Must have a login button that redirects the users who have an account to the login page.

3.4.1.2 Sign up

Must have a Sign up button for users that do not have an account that redirects the user to the registration page.

3.4.2 Log In Page

3.4.2.1 Username or Email Blank

There must be a blank where the user can input the username or email they used to create the account.

3.4.2.2 Password Blank

There must be a blank for the user to input the password they created when signing up for the app.

3.4.2.3 Forgot Password

There must be a place where the user can select that they forgot their password. This button will take the user to the page to either recover their password or reset it.

3.4.3 Forgot Password Page

3.4.3.1 Enter Email Address

A blank to enter the email associated with the account must be created. Once the email is typed into the blank the next step button will send an email to the users email with a security code.

There will also be a button to create an account.

3.4.3.2 Security Pin

The security pin page will include a space to input the numbers emailed to the user. After typing the in the security pin clicking accept will either take the user to the next page or tell them the security code is incorrect. If the user did not receive the email they can click the send again button to have the security pin emailed to them again.

There will also be a button to create an account.

3.4.3.3 New Password

There will be a blank to enter the new password and a blank to confirm the new password. After typing them in the and pressing the change password button the users password will be updated and they will be taken to the main page.

3.4.4 Create Account Page

3.4.4.1 Full Name Blank

A blank will be created for the user to enter their first and last name.

3.4.4.2 Email Blank

The next blank listed will be the blank to enter the users email address. This email address will be used as the username for the app.

3.4.4.3 Phone Number Blank

The phone number will be entered by the user next. This will be saved to the user's profile.

3.4.4.4 Date of Birth Blank

The user’s date of birth will also be entered on this page. This information will be saved to the user’s profile.

3.4.4.5 Password Blank

The user will now enter a password that will be used to log into their account. The password will be hidden from view when typed but there will be a button so that the user can see what they are typing.

3.4.4.6 Confirm Password Blank

The user will retype in their password to confirm that it is imputed correctly. The password will be hidden from view when typed but there will be a button so that the user can see what they are typing.

3.4.4.7 Sign Up Button

Once all the information has been input by the user they can press the sign up button that will create their account and take them to the main page.

3.4.4.8 Sign In Button

If a user who already has an account accidentally clicks sign up, they can click the sign in button to be redirected to the sign in page.

3.4.5 Budget Page

This page will allow users to set budget goals. Additionally, the user will be able to enter their salary and how soon they would like to meet the budget goal, in order for the app to calculate how long it will take to meet their goal

3.4.6 Spending Page

This page will allow the user to get a clear view, with visuals, where their spending is going.

3.4.7 Profile Management Page

This page will allow the user to manage/customize their profile:

- Edit Profile Picture
- Edit profile
    - Switch from dark/light mode
- Settings
- Log out

3.4.8 Loan Management Page

Users will be able to enter their student loan amount and the app will be able to help manage the payments to help the user pay off their loans with ease.

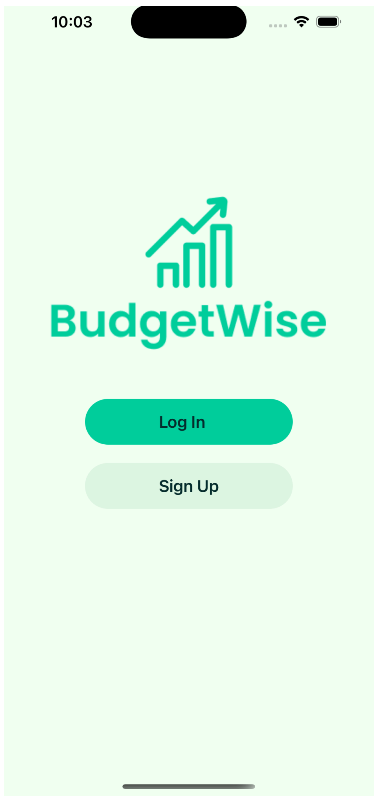
3.4.9 Credit Card/Banking Information Home Page

Users Will Land here first upon signing in in which they can add an account in this page it will remind you of any upcoming payments that will be due Red signifies Urgent, Yellow is a warning and Green means it’s not due for a couple of weeks. Here you can either navigate to the Credit Cards Page or Check on your Cash such as checking or savings accounts.

3.9.1 Credit Card Details Pages

When either pressing on the Orange discover tab or When pressing on Credit Cards you will then be redirected to this page in which your card details and information will be displayed. Information Includes Balances, Payment Due Dates, and the minimum monthly Payment. As well as any recent Transactions. Similar details will then be displayed when selecting the Amex tab, where it will then show you those Card details and their information. As well as any other cards you may add.

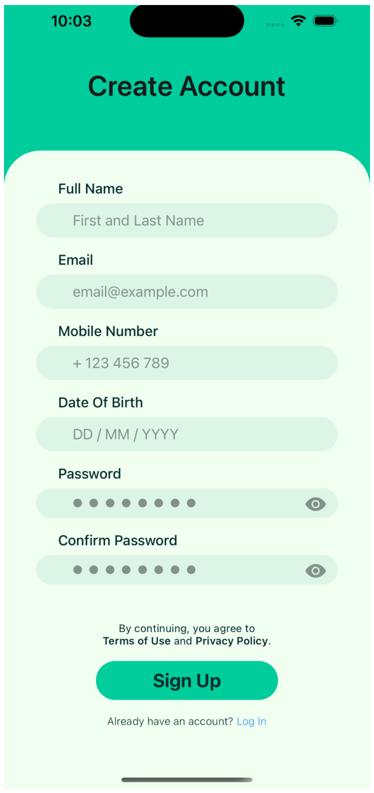
3.5 Mock Up Designs


10:03

BudgetWise

Log In

Sign Up


10:03

Create Account

Full Name

First and Last Name

Email

email@example.com

Mobile Number

+ 123 456 789

Date Of Birth

DD / MM / YYYY

Password

● ● ● ● ● ● ● ●

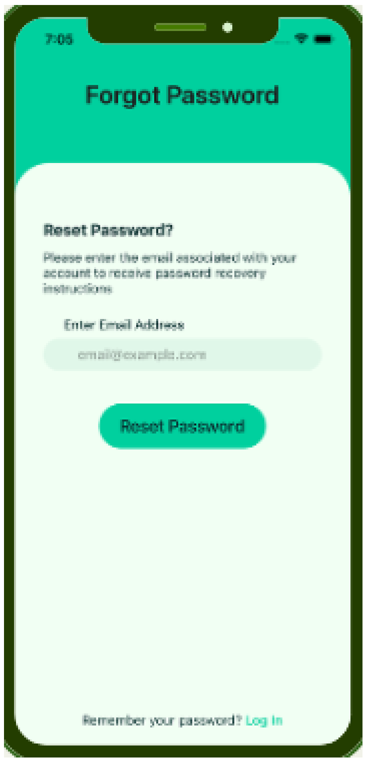
Confirm Password

● ● ● ● ● ● ● ●

By continuing, you agree to
Terms of Use and **Privacy Policy**.

Sign Up

Already have an account? Log In


7:05

Forgot Password

Reset Password?

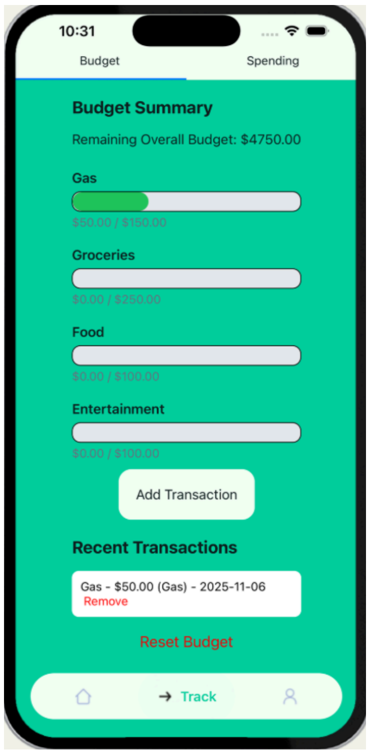
Please enter the email associated with your account to receive password recovery instructions

Enter Email Address

email@example.com

Reset Password

Remember your password? Log In


10:31

Budget Spending

Budget Summary

Remaining Overall Budget: $4750.00

Gas

$50.00 / $150.00

Groceries

$0.00 / $250.00

Food

$0.00 / $100.00

Entertainment

$0.00 / $100.00

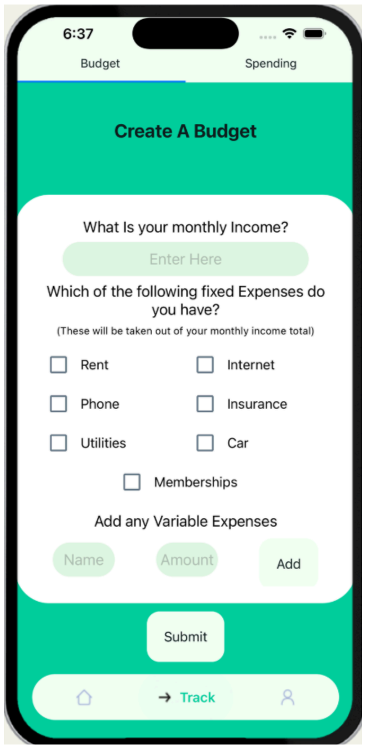
Add Transaction

Recent Transactions

Gas - $50.00 (Gas) - 2025-11-06
Remove

Reset Budget

Track


6:37

Budget | Spending

Create A Budget

What Is your monthly Income?

Enter Here

Which of the following fixed Expenses do you have?

(These will be taken out of your monthly income total)

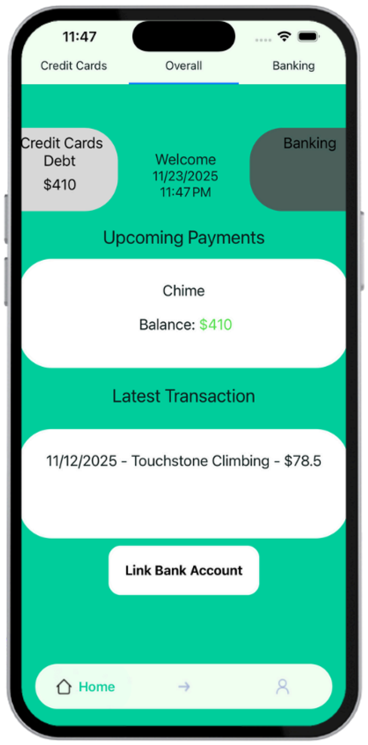
- ☐ Rent
- ☐ Internet
- ☐ Phone
- ☐ Insurance
- ☐ Utilities
- ☐ Car
- ☐ Memberships

Add any Variable Expenses

Name | Amount | Add

Submit

Track


11:47

Credit Cards | Overall | Banking

Credit Cards Debt
$410

Welcome
11/23/2025
11:47 PM

Banking

Upcoming Payments

Chime

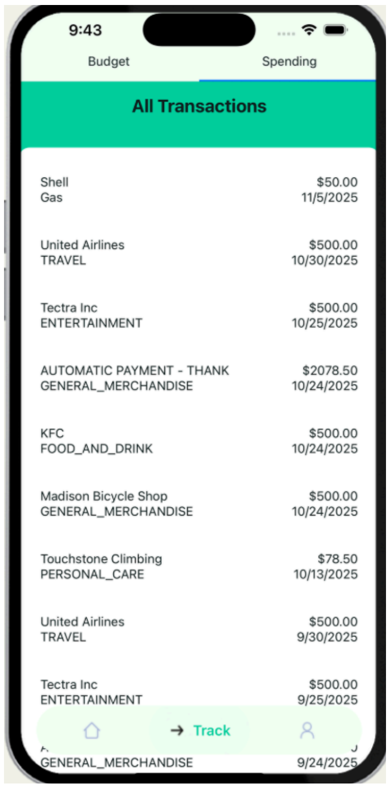
Balance: $410

Latest Transaction

11/12/2025 - Touchstone Climbing - $78.5

Link Bank Account

Home

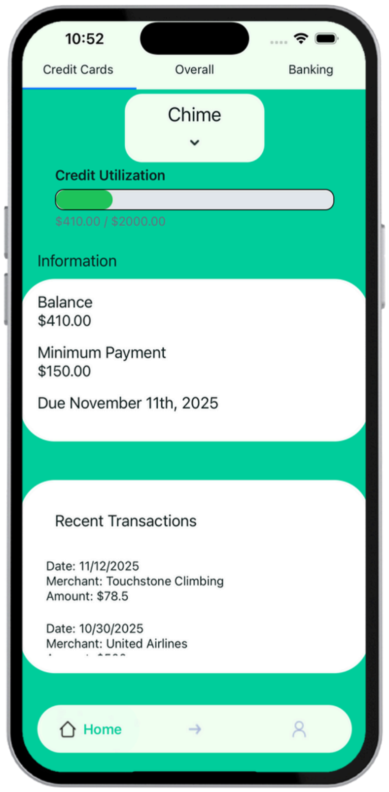

9:43

Budget | Spending

All Transactions

Shell Gas	$50.00 11/5/2025
United Airlines TRAVEL	$500.00 10/30/2025
Tectra Inc ENTERTAINMENT	$500.00 10/25/2025
AUTOMATIC PAYMENT - THANK GENERAL_MERCHANDISE	$2078.50 10/24/2025
KFC FOOD_AND_DRINK	$500.00 10/24/2025
Madison Bicycle Shop GENERAL_MERCHANDISE	$500.00 10/24/2025
Touchstone Climbing PERSONAL_CARE	$78.50 10/13/2025
United Airlines TRAVEL	$500.00 9/30/2025
Tectra Inc ENTERTAINMENT	$500.00 9/25/2025
GENERAL_MERCHANDISE	9/24/2025

Track


10:52

Credit Cards | Overall | Banking

Chime

Credit Utilization

$410.00 / $2000.00

Information

Balance
$410.00

Minimum Payment
$150.00

Due November 11th, 2025

Recent Transactions

Date: 11/12/2025
Merchant: Touchstone Climbing
Amount: $78.5

Date: 10/30/2025
Merchant: United Airlines

Home

4.0 Non-Functional Requirements

4.1 Security

This app will need to prioritize the security of the account numbers as it is dealing with bank account connections. The data will have to be encrypted and kept secure. Whether it is in use or just being stored.

4.2 Capacity

For capacity, it would be best if we could manage a few hundred, as it really isn't too taxing since it is just data being displayed in different forms. It’s not anything that's constantly changing every second.

4.3 Usability

Usability. As for usability, we need the program to be quick and responsive. Allowing people from all over the country to be able to use this program and respond in a snap. Being available to people, whether they are using IOS or Android. And if time allows, we will see how compatible it is on a web browser.

4.4 Other

We also need data to be extracted from Firebase, depending on the user's login data. Being able to store it accordingly and accurately.

5.0 External Interface Requirements

5.1 User Interface Requirements

5.1.1 Buttons

Buttons will be used to move the app to different screens and to save the imputed information.

5.1.2 Blanks

Blanks will be given for the user to input their information to create their account and to enter their financial information.

5.1.3 Graphics

Graphics such as pie charts and other graphs will be used to help explain budgeting to the users.

5.1.4 Error Messages

Should the user enter incorrect financial information an error message will be displayed on their screen.

5.2 Hardware Interface Requirements

BudgetWise will be supported on both Android and iOS devices.

5.3 Software Interface Requirements

BudgetWise will use Plaid to bring users financial information from their bank.
We will also be connecting Firebase to our application in order to store the users' information, such as their login information. This will also give us analytics on the application, such as user activity over time, Active Users, retention, Engagement, and Users by country. And if we ever monetize, it'll also keep track of Revenue and average revenue per active user.

5.4 Communication Interface Requirements

Email communication will be used to send a security pin to users to reset their password

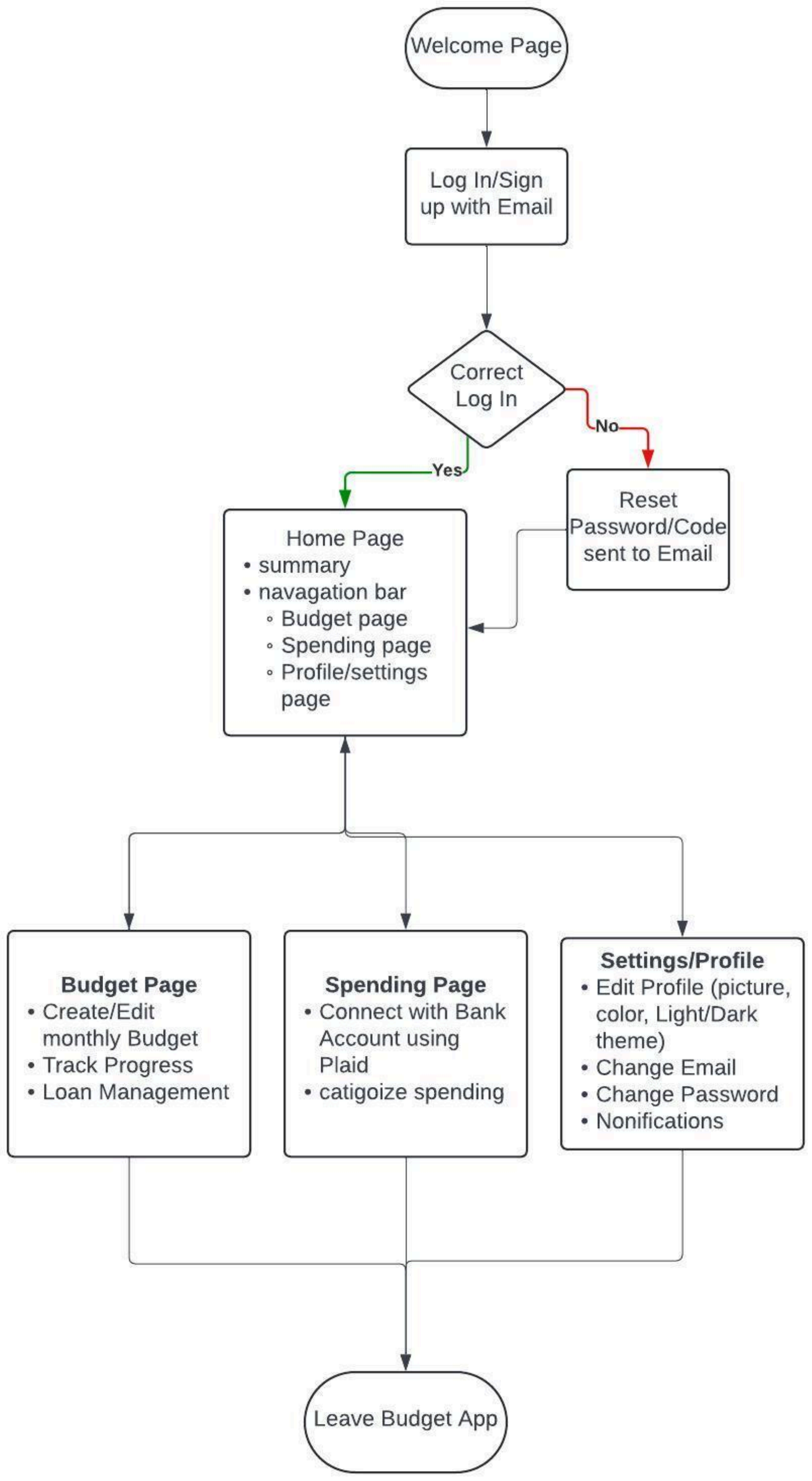
6.0 Analysis

6.1 Use Cases

Some use cases for BudgetWise include

- Expense tracking
- Budget planning
- Alerts for over spending
- Loan tracking

6.2 Data Flow Chart



Design

1. Design Considerations

1.1. Assumptions and Dependencies

We assume that:

- All users should have access to any iOS/Android device with sufficient internet connection
- All users should have a basic understanding of budgeting i.e knowing what income/expenses are, being able to set limits on their spending, etc
- All users should be responsible enough to set their own authentication securely (Hard to guess PIN/Passwords)

Dependencies:

- Android/iOS
- API: Plaid
- Database: Firebase auth

1.2. General Constraints

As this is a mobile budget app, there will be mobile constraints that include cross-platform between iOS and Android devices. There will be security constraints that deal with the user's personal information and account information that require a certain level of authentication to prevent others from gaining access.

1. Mobile Requirements
    - The app must be able to function properly on iOS and Android devices

2. Security Requirements
    - Every user's data must be encrypted when at rest and in transit to prevent stealing of data
    - This application must have secure user authentication, such as multi-factor authentication, when logging into an account and/or bank account. Other authentication includes Face ID (iOS) and/or a one-time passcode via text message.
3. Other constraints
    - Instant transaction processing
    - Able to handle a vast amount of transaction history (memory management)
    - Responsive app with none to minimal delays (user experience)
    - Push notifications for budget alerts (user experience)
    - The application will only support English as the preferred language and USD as the preferred currency at the moment

1.3. Development Methods

For our project, we are taking an agile approach to development. Since we have a limited amount of time to deliver and test, we need to work as quickly as possible in order to make sure our product is working as it should and to ensure functionality. We are leaning more towards a scrum approach in which we break down the whole thing into manageable parts and then continue on to the rest. This will allow us to give full attention to all the individual parts, ensuring quality instead of tackling it head-on all at once.

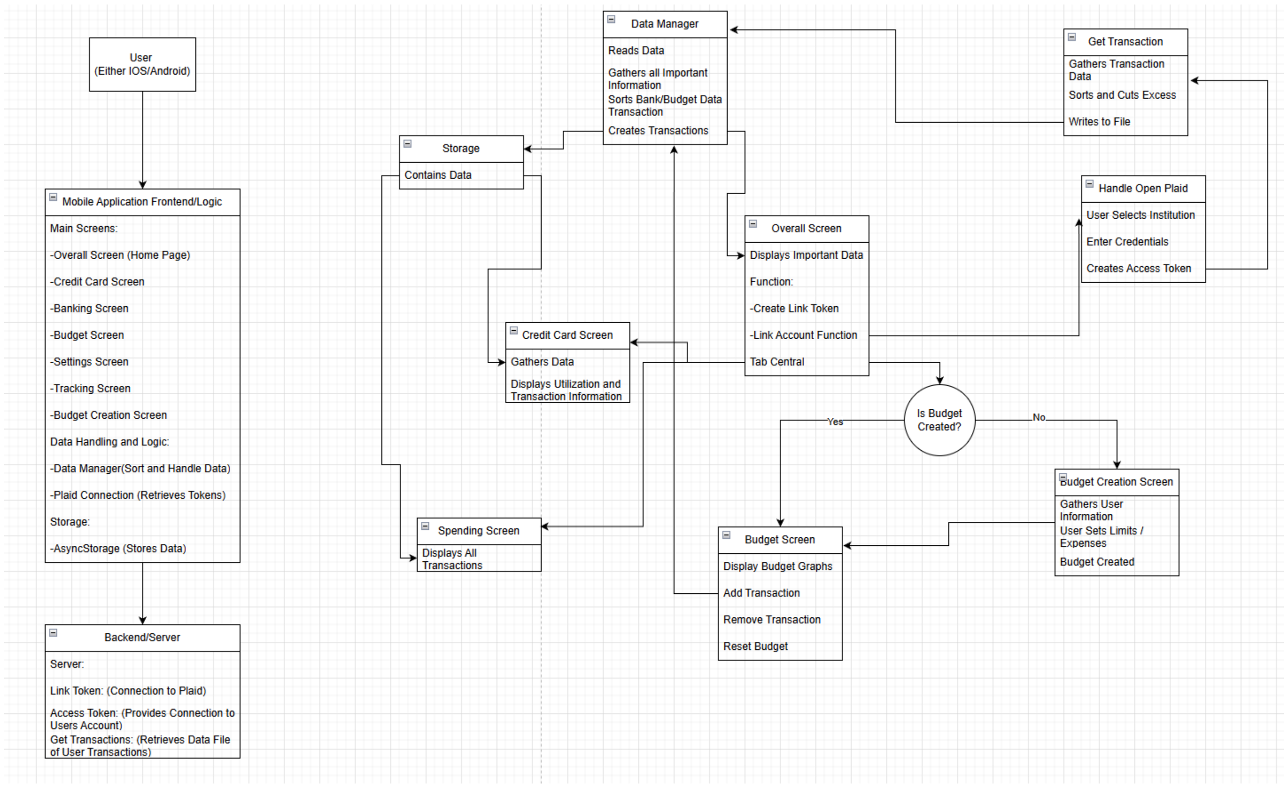
2. Architectural Strategies

As for the use of certain products, we have decided to go with React Native for our language, Firebase for storing all of our data, and Plaid in order to gather the information for our product. Most of these decisions were made because they were what was most known to us and what we had experience with. There were other alternatives, such as Flutter or a different way to gather the financial data, but they were all foreign to us. As for the design decisions, we want the user to be able to understand our product easily. And for them not to feel overwhelmed by our product. It needs to be accessible to a vast user base, and making it simple and easy to understand is key to making it happen. We can't make it too complex, or else users will get overwhelmed and will not want to take the time to learn how to use the product. We need to make a good first impression on the product; they need to see it as simple and easy to consume. As for handling the data, it would be best for us to just display data and not allow us to make any changes to the users' accounts. Such as withdrawing money, since it could pose a security risk. It'd be best to display their data in a big picture format and help them manage their finances in a way that is easy to manage. And for storing all our information, we will use Firebase and link it to our application to handle all our data.

3. System Architecture

- BudgetWise is divided into a lightweight cloud backend and a thin, cross-platform front end (React Native). The main elements of the runtime are:
- React Native mobile client: presentation logic, input validation, local cache, and user interface screens. operates on iOS and Android and uses TLS to connect to the backend.
- Plaid integration, authentication, input validation, business rules (budget calculation), and server-side notifications are all coordinated by the Firebase Functions API/Backend.
- User profiles, budgets, transaction history snapshots, user settings, and preferences are all stored in the data store (Firebase Firestore + Firebase Auth); Firebase Auth is utilized for credential management.
- The backend trades tokens with Plaid and normalizes incoming transactions using the bank connector (Plaid), which provides safe, read-only access to user bank accounts and transaction feeds.
- Real-time push notifications and scheduled budget alerts are managed by the notification service (Firebase Cloud Messaging or platform push).

- Analytics & Monitoring: Firebase Analytics and Crashlytics provide lightweight event logging for use and error monitoring.
- User → mobile client → backend (auth + business logic) → Firestore is the high-level data flow. The backend routinely collects transaction batches from Plaid for bank data (or upon user request), normalizes and saves them, and then initiates notifications and a budget re-evaluation. The data layer (Firestore rules and encryption at rest), authentication layer (Firebase Auth + optional MFA), and transport layer (HTTPS/TLS) all enforce security boundaries. Security (no direct client-side Plaid keys and role-based Firestore rules), developer productivity (React Native + Firebase), and responsiveness (client caching + offline read) are given top priority in this design.



4. Detailed System Design

The main elements of the BudgetWise system are explained in this part along with their classification, function, limitations, and interfaces. Every component is described, along with its limitations and how it interacts with other system elements.

4.1 Classification

- Subsystem/Service (Firebase Auth supported) is the classification.
- It manages user registration, login, session management, password resets, and optional multi-factor authentication. It securely handles OAuth and Plaid tokens on the backend and associates an authenticated identity with Firestore data.
- Limitations:
- restricted to Firebase Auth features (password, Google, Apple, and email).
- All flows must use TLS, and tokens need to be safely managed and updated.
- Plaid access tokens are never stored on the device; they are only kept in backend secrets.
- Rate-limiting and account lockouts must be implemented to prevent brute-force attacks.
- Interfaces/Exports: Sign-up, login, and refresh Firebase SDK endpoints.
- Server routines for session revocation, UID → user record mapping, and token validation.

4.2 Definition

- Classification: Plaid integration with a backend service.
- Definition: Stores canonical transactions in Firestore, normalizes merchant names and categories, deduplicates records, and connects to Plaid to collect transaction feeds. allows for automatic budget classification using MCC tags.
- Limitations:
- Plaid incremental sync must be used, and rate limitations must be observed.
- Idempotency keys are necessary to avoid duplicates.
- eliminates sensitive account numbers and only keeps the fields that are required for budgeting.
- The client needs to show sync timestamps; it can be asynchronous.
- Exports and Interfaces:
- To pull transactions, use syncTransactions(userId, startDate, endDate).
- To obtain paginated results, use getTransactions(userId, query).
- Webhook handlers to react to the new activity alerts from Plaid.

4.3 Constraints

- Module/library (stateless functions) is the classification.
- Definition: Classifies transactions, computes spend versus targets, and sets off alarms to assess budgets against transaction streams. Both manual overrides and auto-classification are supported.
- Limitations: The computations need to be idempotent and deterministic.
- Load is decreased via batch recalculations every night; some recalculations take place instantly.
- While raw transactions continue to be the source of truth, aggregates are stored for efficiency.
- Interfaces/Exports: return per-category totals via calculatingBudgetStatus(userId, period).
- classifyTransaction(transaction) for the categories that have been suggested.
- User rule management functions (merchant → category mappings).

4.4 Resources

One resource we will be using is Plaid. Plaid will provide us with an API that we allow us to connect to the users bank account securely. We will be able to connect the users credit card and transaction history in order to help them to create budgets. Some race conditions that may occur with the resource is that there might be multiple users attempting to update data at the same exact time. A way for us to prevent this we will make sure to keep in mind the rate limits that Plain offers in order to prevent bottlenecks from occurring.

4.5 Interface/Exports

- Client modules (React Native) are the classification.
- Definition: Offers dashboard, budget, and transaction interfaces that are compatible with several platforms. enables optimistic updates and offline caching for a seamless user experience.
- Limitations: It must adhere to platform norms and run on both iOS and Android.
- Only recent transactions and preferences are stored locally.
- Conflicts must be resolved deterministically or through user input in offline modifications.
- Interfaces/Exports: Firebase SDK-based UI flows, including fetchDashboard(), submitCategoryOverride(), and requestManualSync().

<u>Development</u>

1.0 Technical Details

1.1 Launch Page

When the user opens up the BudgetWise app, the launch page displays a splash screen with our logo. After a brief couple of seconds, it automatically transitions to the welcome screen, which offers the user to either “Log in” (existing users) or “Sign Up” (new users).

Built with React Native

1.2 Log in Page

If the user selects “Log in,” they’re then moved over to the Login page, where they must provide their email address and password. If the user forgets their password, they can click “Forgot password?” and they are then navigated to that page, or they can click “Sign Up” if they haven’t done so.

Used Firebase Authentication. Firebase is able to handle password encryption and is able to manage sessions when a user is logged in. Used React Native to transition to different pages

1.3 Create Account Page

If the user selects “Sign Up,” they’re then moved over to the Create Account page where they must enter their full name, email address, mobile number, date of birth, and password. Once this is all complete, they can click Sign Up and create their account.

Used Firebase Authentication. When a user creates their account, the data is stored in Firebase Firestore.

1.4 Forgot Password Page

If the existing user forgets their password, they can click “Forgot Password?” and then they’re navigated to this page where they are asked to enter their email address. If the email address is correct, they will receive an email to reset their password.

Used Firebase Authentication. Firebase generates a token and sends the reset email automatically if the email address is correct.

1.4.1 Password Reset Email Page

The user is sent a link in their email to reset their password. Once clicked, it’ll take them to the Reset Password Page.

Used Firebase Authentication. The token that Firebase generates expires at a certain time and can only be used once.

1.4.2 Reset Password Page

The user is able to create a new password after clicking the reset link in their email.

Used Firebase Authentication. It validates the token and updates the password for that account. The user must enter their new password twice for confirmation.

1.5 Budget Page

The user can set monthly limits, build category-based budgets, and view real-time updates to remaining amounts when transactions are posted on the Budget page. A header dropdown allows users to change months and add, edit, or remove categories. Quick feedback is provided via a progress bar and the "amount remaining / over" label touching a category brings up its detail view with recently completed associated transactions.

Used React Native for the navigation and user interface. Each budget (category, limit, period, thresholds, and timestamps) is stored under the logged-in user using Firebase Firestore. The client's most recent linked-account information, obtained through Plaid, is used to calculate the transaction totals for each category.

1.6 Spending Page

This will be the users date range, and category filters, the Spending page displays the user's most recent transactions across all linked accounts. The merchant, date, amount, and category are displayed in each transaction row; pressing brings up a details drawer with notes, re-categorization, and report hiding options. A summary header displays the top categories and the overall amount spent during the chosen window.

Pull-to-refresh and React Native FlatList were used to create a fluid feed. Plaid is used to retrieve transactions, while Firebase Firestore is used to store user category overrides and notes. "Total Spent" and "Top Categories" are the results of straightforward client-side aggregate.

1.7 Profile Management Page

Users can access and edit their app preferences and basic profile information (name, mobile number, and DOB) in Profile. Password changes, email verification status checks, and sign-out are examples of security measures. The user can relink or remove financial institutions from a Data & Accounts area that displays associated financial institutions. Users can also request the deletion of their accounts through the Privacy & Deletion route.

Firebase Authentication was used for session management and secure account operations (re-authentication for password/email changes). App preferences and profile information are kept in Firebase Firestore under the user document. The Plaid Link flow begins when financial institutions are linked or unlinked.

1.8 Home Page

Quick-glance cards are displayed on the Home (dashboard) after logging in. These include "Budgets" (total remaining + top 1-2 budgets), "Spending" (total for current period), and "Credit Cards" (balance snapshot). Each card has a "See all" button that takes you to the entire page (Budget, Spending, Credit Card Details). You may quickly access Home, Track (Budget/Spending), and Profile via the bottom menu.

React Native was used in its construction. The most recent Plaid pull (transactions summary) and Firestore (budgets, user preferences) provide the data displayed on the dashboard. In between refreshes, light client-side memoization avoids needless recalculations.

1.9 Credit Card Details Page

The user's associated credit cards are listed on this page along with their username, disguised last four, current balance, and recent activity. When choosing a card, information is displayed, including recent transactions, a breakdown of categories, and, if available, statement information (balance and closing date) and payment instructions (due date and minimum amount). The page shows a utilization estimate if limits are known.

Uses React Navigation for stack transitions and React Native for the user interface. Plaid provides the card/transaction data, whereas Firestore stores any user annotations (such as a unique card nickname). Utilization and category breakdown calculations are carried out client-side using retrieved data.

2.0 Database Connection

Firebase:

- Firebase Authentication: Handles the user registration, login, and password management
- Firebase Firestore: Stores user profiles

Plaid:

- Connects to user bank accounts and pulls in transaction data.

3.0 How To Set Project Up – steps

To set up the BudgetWise mobile app:

1. Install Node.js, Expo, and Git onto your computer
2. Clone the repository from Github
3. Install the dependencies by typing ‘npm install’ in terminal
4. Start up the app by typing ‘npx expo start’ in terminal
    a. You can either use any mobile device of your choosing or use the emulator

The app utilizes Firebase and Plaid. Firebase is used for authentication and real time data storage, whereas Plaid is for secure financial data integration. Firebase is automatically initialized when the app starts. When the user is either signing up or logging in, their data is stored within our database. Plaid is used more on the backend, where users can link their bank account and retrieve their data from there.

Once the user launches the app:

1. They can choose to either log in or sign up
2. After a successful login, they are directed to the dashboard
    a. The dashboard displays an overview of your credit card page, overall page, and banking page
2. The user is able to select the ‘Track’ tab at the bottom
    a. The track tab allows the user to track their budget and spending
3. The user is also able to select the ‘Profile’ tab at the bottom
    a. The profile tab allows the user to customize their profile, app settings, and logging out.

<u>Software Testing Plan and Report</u>

1.0 Overview

1.1 Purpose

The goal of this test plan is to make sure our app, BudgetWise, works the way we intended it to. We want to make sure that users of our app can create an account, log in successfully, create budgets (add/remove income and expenses), and be able to see a summary of their transactions. We also want to validate that our error handling and input validation works correctly as intended.

2.0 Testing Summary

2.1 Scope of Testing

2.1.1 In scope

- User Account Management
    - Creating an account
    - Logging in and out
- Budgets
    - Able to make a new budget each month
    - Add/edit/delete budget categories
- Transactions
    - Adding income
    - Adding expenses

    - Editing transactions
- Summary
    - Display total income, expenses, and remaining budget
    - View categorized spending
- Validation
    - Handle invalid credentials on login
    - Required fields validation

2.1.2 Out of scope

- Performance testing utilizing real-world large data sets
- Security test passed basic login
- Backup/restore procedures
- Integration with other external APIs besides the one currently used

3.0 Analysis of Scope and Test Focus Areas

3.1 Release Content

With the release of BudgetWise, it will include all of the features needed for personal budgeting. The content of the app allows users to track their money monthly in a simple and organized way. The main features in this release includes:

- Users are able to create an account, log in, and log out
- Users are able to create a budget each month and set limits for their spending categories
- Users can add, remove, and edit their transactions
- Users are able to view the summary of their total income, expenses, and remaining budget

With the help of external API integrations such as Firebase Authentication and Plaid, we're able to implement these features safely and securely with ease of use

3.2 Regression Testing

Regression testing was done whenever there were any changes to existing features or when new features were added. The purpose of doing this is that we wanted our core features to still behave correctly whenever changes were made. We focused mainly on retesting to make sure no new bugs were introduced. This was throughout the project on iOS and Android devices.

3.3 Test Environment

The testing platform for our application includes:

- Operating Systems (iOS/Android OS)
    - Testing on Android 12/13 and iOS 16/17
    - Runs on various device models
- Hardware
    - Verify the device has sufficient amount of RAM
- Software

    - Frameworks includes React Native, Expo Go
    - API includes Firebase Authentication and Plaid
    - Development includes VS Code, Xcode, Android Studio, and GitHub
- Network
    - WiFi needed

4.0 Test Cases

Test ID	Function	Test Objective	Evaluation Criteria	Priority
1	Create Account	Verify that new users are able to create an account successfully	User is added into the system via Firebase	High
2	Logging In And Out	Ensure users are able to log in and log out securely	Login and Logout succeeds	High
3	Password Recovery	If a user forgets their password all of the necessary steps to create a new on works	A new passwords is able to be created	
4	Creation Of Budget	Confirm users are able to create a new budget monthly with a total amount	Budget appears with the correct values	High
5	Connect Bank Account	The bank account of the user is successfully contented to BudgetWise through Plaid	The account is successful connected and loaded into the app	
6	Add Income	Ensure users are able to enter their income for any given month	Income update correctly	High
7	Add Expenses	Confirm users are able to add expenses based on the categories	Expenses and balances update correctly	High
8	Delete Transactions	Ensure users can modify/remove their transactions	Changes to transaction reflects in the total and the dashboard	Medium
9	Input Validation	Ensure the required fields cannot be left blank	Error messages appear	High

5.0 Other Testing

5.1 Unit Testing

Unit testing was done manually by each member of the group during development. We ensured that each function within BudgetWise works as expected and intended. Some of these functions include calculating the income, expenses and budget. We also tested to see if users are able to update categories and as well as edit transactions.

5.2 Integration Testing

Integration testing consisted of making sure the various parts of the app worked correctly together. This includes making sure budgets are able to load properly after logging in, users are able to navigate to each screen smoothly, and data is able to flow correctly from our backend (Firebase Authentication/Plaid) into the UI.

6.0 Test Strategy

6.1 Test Type & Approach

Test Type	Objectives
Functional Testing	● Perform tests on monthly budget ● Perform tests on editing existing data ● Perform tests on adding/removing income, expenses, and transactions ● Confirm the dashboard displays the totals accurately
Unit Testing	● Test each functions to make sure it's working accordingly as intended and there's no bugs ● Make sure each input validation functions works correctly ● Verify that transaction functions are working correctly (core feature of BudgetWise)
Integration Testing	● Make sure that each components within the app works together smoothly ● Verify that our backend works accordingly with the app's UI ● Verify that as each new transaction is inputted, it's updated in the dashboard

6.2 Test Data

6.2.1 Sample Test Accounts

	Correct Test Account 1	**Correct Test Account 2**	**Incorrect Test Account**
Email Address	test1@gmail.com	test2@gmail.com	test3@gmail.com
Password	BudgetWise100!!	BudgetWise100!!	BudgetWise100!!
Outcome	Success	Success	Fail

These accounts are created and used to test the login feature, as well as the main features, of BudgetWise.

6.2.2 Sample Data

Data Types	Description	Values for Testing
Budget	This represents the monthly budget. It includes the selected month and the amount allocated for that specific month	Month: January 2026 Budget: $2000
Category	This represents how users are able to create their own category if needed and allocate a specific amount of money to that category. Allows users to track their spending and view the remaining balance for that specific category	Rent: $1000 Gas: $60 Groceries: $100 Dining Out: $80 Shopping: $150 Savings: $100
Transaction	This represents the income and expense input into BudgetWise. As each transaction is inputted, the budget is updated accordingly, and the dashboard reflects the financial summary for that month	Income: $2,500 Expenses: ○ Rent: $1000 ○ Date Night: $70 ○ Vending Machine: $2 ○ …

6.3 Test Procedures

	Create Account	**Login with correct credentials**	**Login with incorrect credentials**	**Create Monthly Budget**	**Connect Account**
Step 1:	Open app	Open app	Open app	Open app	Open App
Step 2:	Click ‘Register’	Enter valid credentials (username, password)	Enter invalid credentials (username, password)	Head to Track Tab (Middle tab on bottom of screen)	Click Link Bank Account Button on Home Screen
Step 3:	Enter correct information for the required fields	Click ‘Log In’	Click ‘Log In’	Fill out your details	Fill in your details and select your bank
Step 4:	Tap ‘Create Account’			Hit Submit	Refresh Application
Expected:	Account is created and no error message is displayed	User is logged in	Error message is displayed and user isn’t able to log in	Budget is then created	App is now connected to institution and will populate expected tabs

	Add Income	**Add Expense**	**Adding Fixed Expense**	**Delete Transaction**	**View Summary**
Step 1:	Head to Track Tab On the bottom Tabs (Middle Tab)	Once budget is created head to Track Tab and Then Budget	Head to track Tab Before Budget has been created	After Budget is Created go to your Track Tab Then Budget	Add Budget or Connect Card to application (Add Expense/ Connect Account)
Step 2:	Once on the Create a Budget Tab fill out Your monthly Income	Press Add Transaction Button	On this screen check the checkbox next to your fixed expense	Look under Recent Transactions	Head to Track Tab

Step 3:	Submit At bottom once other fields are completed	Fill in the Transaction Name, Amount and select your Category	Type in your desired amount	Select the Red Remove to desired Transaction	Select the Spending Tab at the top of the screen
Step 4:		Submit with the add button	Submit once other fields have been filled out		
Expected:	Income will then be set and adjusted based on your variables and fixed income expenses	Budget Summary Tab will be updated and your progress bar should now be filled accordingly.	Fixed expenses will then be automatically removed from your monthly income	Transaction will then be deleted and Bar will be updated accordingly	Here you will see all of your transactions Added through budget or By Institution Connection

7.0 Entry and Exit Criteria

7.1 Entry Criteria

Testing will commence once the application is considered stable with minimal to no errors on Android and iOS devices. The environment for testing will be set up with sample accounts created, as well as

sample data. All of the test cases must be reviewed so that testers are able to understand and give the expected results. Once conditions have been met, the team can start executing the test.

7.2 Exit Criteria

Testing for BudgetWise will be considered complete when all test cases have been executed and all major defects have been properly addressed or resolved. The main core features of the application (login, budgeting, etc) must function 100% without any issues such as crashes or errors. The Software Test Report must include any pass or fail results, along with relevant notes. Once all of the criteria have been met and the app is stable, the testing phase will be completed.

8.0 Software Test Report

Test ID	Requirement	Pass	Fail	Severity
1	Create Account	🟩		
2	Login	🟩		
3	Password Recovery		🟥	High
4	Create Monthly Budget	🟩		
5	Connect Account	🟩		
6	Add Income	🟩		
7	Add Expenses	🟩		
8	Fixed Expense	🟩		
9	Edit Transaction	🟩		

8.1 Notes and Observations

- Each function of BudgetWise works accordingly as it should.
- The only function that fails is ‘Password Recovery’. The issue here is that users aren’t able to receive emails upon clicking the ‘Reset Password’ button. This problem is most likely from our API, Firebase Authentication. We need to further review what we did incorrectly in the process and fix it accordingly.

Version Control

A group account Github will be created for our budget app. We will utilize their version control software. All code will be worked on Github where multiple users can work on the code at the same time and be able to see the recent history should we need to go back to a previous version. GitHub facilitates effective team collaboration in addition to basic version tracking by allowing multiple members to work on different features concurrently without impeding each other's progress. While new features are being developed and tested, each team member can work in a separate branch, which helps avoid breaking the main codebase. A feature can be reviewed and integrated into the main branch once it is finished. Additionally, GitHub offers thorough commit histories that include timestamps and descriptions for each modification made to the project. This makes it simpler to determine when bugs were introduced and swiftly roll back to a stable version when needed. GitHub Pull requests and issues are also used for task management, bug tracking, and code review prior to merging. This guarantees that every code contribution is examined for accuracy, security, and quality. Additionally, version control protects the team from data loss in the event of a local system failure by acting as a backup system for the entire project. Our team is able to collaborate on BudgetWise while maintaining organization, accountability, and stability by using GitHub consistently throughout the development process.

Conclusion

Working on BudgetWise as a team allowed us to take an idea from the planning stage all the way to a working mobile app. The goal from the beginning was to create a budgeting tool that would help college students understand their financial habits. Throughout this semester, we designed features that made budgeting easier. Some of the features include creating monthly budgets, tracking spending, and viewing categories. We also put a strong emphasis on making the app simple to use so that any one of all ages could pick it up without feeling overwhelmed.

During development, we learned how important it is to balance functionality with security. Since BudgetWise deals with sensitive financial information, we made sure to use secure tools like Firebase for authentication and Plaid for bank connections. These choices helped us protect user data while still allowing the app to access real financial information. Testing showed that most features worked the way we expected them to. Our app performed well across different devices and operating systems.

Overall, this project taught us a lot about working as a team, communicating, and creating software step by step. Although we had a large group of four, we managed to split our work into manageable parts so that everyone could contribute. Even when we had issues to confront, we adapted to them and kept working. Eventually, we ended up creating BudgetWise, a system that helps users see their finances and plan for goals.

In the future, we hope to continue improving the app by adding more features, making the interface even more accessible, and fixing the areas that still need work. BudgetWise is a strong starting point, and we are proud of what we accomplished together.

Appendix A - Definitions, Acronyms, and References

1.0 Definitions

Budget: A financial plan that estimates income and expenses over a given period
Transaction: Any recorded financial activity (income or expenses)
Authentication: A process of verifying the identity of the user
Encryption: A security technique used to protect sensitive data from unauthorized access

2.0 Acronyms

OS: Operating System (iOS, Android)
API: Application Programming Interface
TLS = Transport Layer Security
SDK = Software Development Kit
UID = User Interface Design
API = Application Programming Interface
UI = User Interface

Appendix B - References

The MockUp interfaces referenced the budget toolkit titled “Finance Management Mobile App UI UX Kit” by Vector Fair in Figma

Appendix C - Components and Tools Used

1.0 Components

- components/Typo.tsx
- components/BackButton.tsx
- components/ScreenWrapper.tsx
- Contexts / Providers
- contexts/AuthContext.tsx
- contexts/ThemeContext.tsx

2.0 Root Layout

- app/_layout.tsx

3.0 Tabs

- app/(tabs)/_layout.tsx
- app/(tabs)/tabControl.tsx
- app/(tabs)/tracking.tsx

4.0 Profile Stack

- app/(tabs)/profile/_layout.tsx
- app/(tabs)/profile/index.tsx
- app/(tabs)/profile/settings.tsx
- app/(tabs)/profile/edit.tsx

5.0 Auth Screens

- app/index.tsx
- app/auth/welcome.tsx
- app/auth/login.tsx
- app/auth/register.tsx
- app/auth/forgot-password.tsx
- app/auth/forgot-password-success.tsx
- app/auth/reset-password.tsx

6.0 Track Screen Tabs

- app/TrackScreenTabs/BudgetSummary.tsx
- app/TrackScreenTabs/BudgetCreation.tsx
- app/TrackScreenTabs/Graph.tsx
- app/TrackScreenTabs/Spending.tsx
- app/TrackScreenTabs/Storage.tsx

7.0 Home Screen Tabs

- app/HomeScreenTabs/timeDisplayer.tsx
- app/HomeScreenTabs/CreditCardScreen.tsx
- app/HomeScreenTabs/OverallScreen.tsx

- app/HomeScreenTabs/BankingScreen.tsx
- app/HomeScreenTabs/DataManager.ts
- app/HomeScreenTabs/data.json

8.0 Tools

- hooks/use-theme-color.ts
- hooks/use-color-scheme.ts & hooks/use-color-scheme.web.ts
- contexts/AuthContext.tsx
- contexts/ThemeContext.tsx
- utils/styling.ts
- auth/firebase.ts
- app/TrackScreenTabs/Storage.tsx
- app/HomeScreenTabs/DataManager.ts
- app/HomeScreenTabs/data.json

Appendix D - Project Plan

1.0 Deliverables - Specific To Your Project

- Budget tracking page
- Spending Tracker page
- Investing Tracker page
- Student Loan management page
- Goals/savings page (make it like keeping a streak)
- Graphs/visual representation of data
- authentication / security
- Subscription tracker (spotify, netflix, chegg, etc)
- Challenges (no spend week, no cafe month, no doordash)
- Customizable UI with accessibility features

2.0 Group Meeting Schedule Date/Time

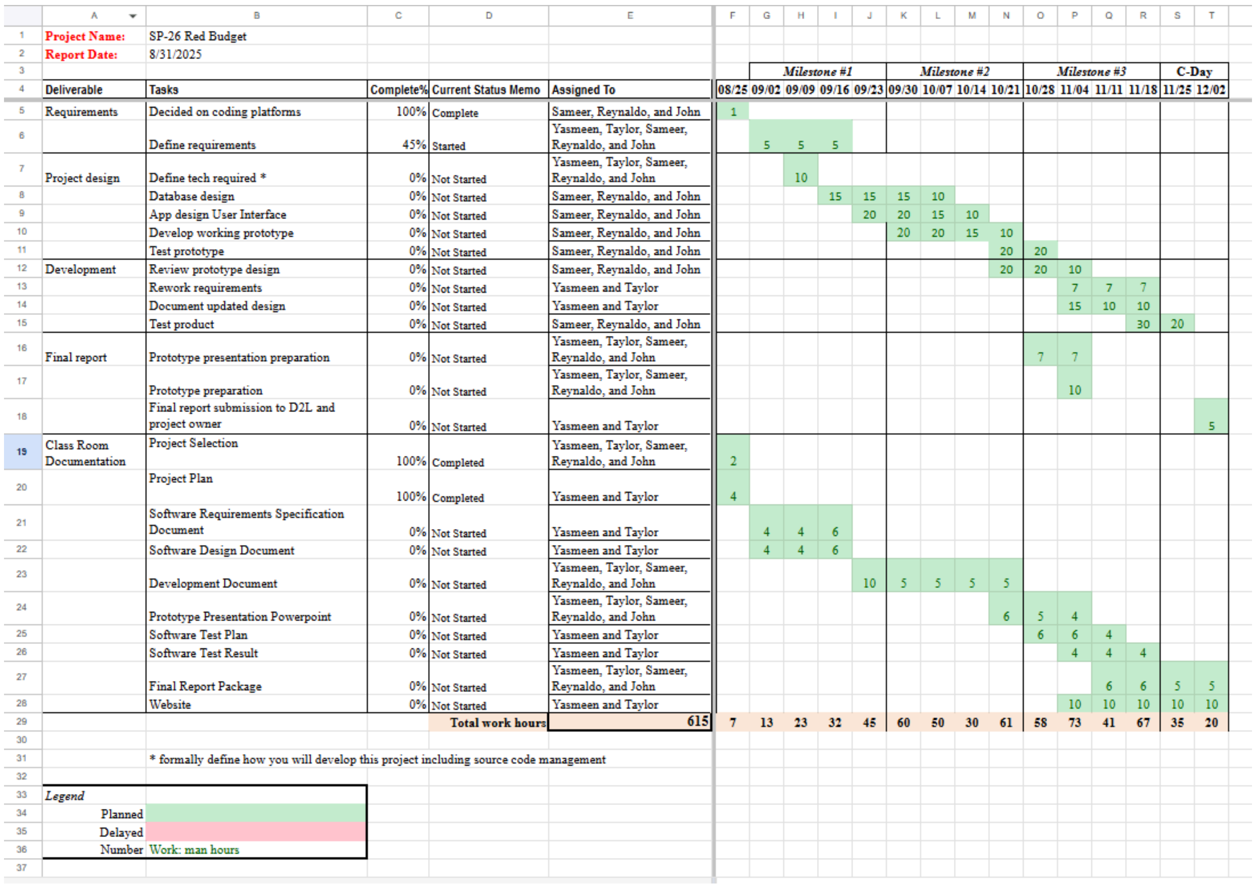
Tuesdays and Thursdays from 11:00 to 12:15 (in class).
More meetings will be added as needed.

3.0 Collaboration and Communication Plan

Communication	—	Teams, Cellphones (Call/Text)
Collaboration	—	iMessage

How Often? At least once every week or when blockers arise.

4.0 Project Schedule and Task Planning



Project Name: SP-26 Red Budget
Report Date: 8/31/2025

						Milestone #1				Milestone #2				Milestone #3				C-Day	
Deliverable	Tasks	Complete%	Current Status Memo	Assigned To	08/25	09/02	09/09	09/16	09/23	09/30	10/07	10/14	10/21	10/28	11/04	11/11	11/18	11/25	12/02
Requirements	Decided on coding platforms	100%	Complete	Sameer, Reynaldo, and John	1														
	Define requirements	45%	Started	Yasmeen, Taylor, Sameer, Reynaldo, and John		5	5	5											
Project design	Define tech required *	0%	Not Started	Yasmeen, Taylor, Sameer, Reynaldo, and John			10												
	Database design	0%	Not Started	Sameer, Reynaldo, and John				15	15	15	10								
	App design User Interface	0%	Not Started	Sameer, Reynaldo, and John					20	20	15	10							
	Develop working prototype	0%	Not Started	Sameer, Reynaldo, and John						20	20	15	10						
	Test prototype	0%	Not Started	Sameer, Reynaldo, and John									20	20					
Development	Review prototype design	0%	Not Started	Sameer, Reynaldo, and John									20	20	10				
	Rework requirements	0%	Not Started	Yasmeen and Taylor											7	7	7		
	Document updated design	0%	Not Started	Yasmeen and Taylor											15	10	10		
	Test product	0%	Not Started	Sameer, Reynaldo, and John													30	20	
Final report	Prototype presentation preparation	0%	Not Started	Yasmeen, Taylor, Sameer, Reynaldo, and John										7	7				
	Prototype preparation	0%	Not Started	Yasmeen, Taylor, Sameer, Reynaldo, and John											10				
	Final report submission to D2L and project owner	0%	Not Started	Yasmeen and Taylor															5
Class Room Documentation	Project Selection	100%	Completed	Yasmeen, Taylor, Sameer, Reynaldo, and John	2														
	Project Plan	100%	Completed	Yasmeen and Taylor	4														
	Software Requirements Specification Document	0%	Not Started	Yasmeen and Taylor		4	4	6											
	Software Design Document	0%	Not Started	Yasmeen and Taylor		4	4	6											
	Development Document	0%	Not Started	Yasmeen, Taylor, Sameer, Reynaldo, and John					10	5	5	5	5						
	Prototype Presentation Powerpoint	0%	Not Started	Yasmeen, Taylor, Sameer, Reynaldo, and John									6	5	4				
	Software Test Plan	0%	Not Started	Yasmeen and Taylor										6	6	4			
	Software Test Result	0%	Not Started	Yasmeen and Taylor											4	4	4		
	Final Report Package	0%	Not Started	Yasmeen, Taylor, Sameer, Reynaldo, and John												6	6	5	5
	Website	0%	Not Started	Yasmeen and Taylor											10	10	10	10	10
			Total work hours	615	7	13	23	32	45	60	50	30	61	58	73	41	67	35	20

* formally define how you will develop this project including source code management

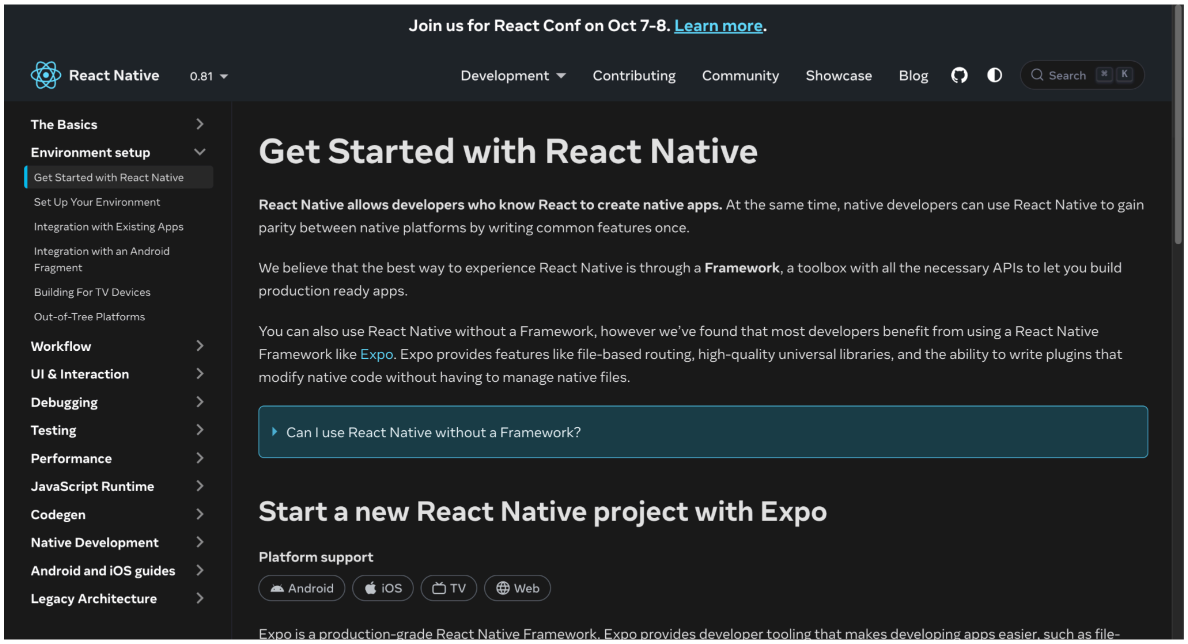
Legend	
Planned	🟩
Delayed	🟥
Number	Work: man hours

5.0 Risk Assessment

After looking at the kind of data our budget app would be collecting we knew that there could be a risk in the collecting and storing of users' financial data. If a user's password for our app gets stolen. There is a risk that their financial data can be stolen and used in identity fraud.

Appendix E - Tutorial Screenshoots for React Native and Flutter

Reynaldo's Screenshot:



Join us for React Conf on Oct 7-8. Learn more.

React Native 0.81 | Development | Contributing | Community | Showcase | Blog | Search

The Basics
Environment setup
- Get Started with React Native
- Set Up Your Environment
- Integration with Existing Apps
- Integration with an Android Fragment
- Building For TV Devices
- Out-of-Tree Platforms

Workflow
UI & Interaction
Debugging
Testing
Performance
JavaScript Runtime
Codegen
Native Development
Android and iOS guides
Legacy Architecture

Get Started with React Native

React Native allows developers who know React to create native apps. At the same time, native developers can use React Native to gain parity between native platforms by writing common features once.

We believe that the best way to experience React Native is through a **Framework**, a toolbox with all the necessary APIs to let you build production ready apps.

You can also use React Native without a Framework, however we've found that most developers benefit from using a React Native Framework like Expo. Expo provides features like file-based routing, high-quality universal libraries, and the ability to write plugins that modify native code without having to manage native files.

Can I use React Native without a Framework?

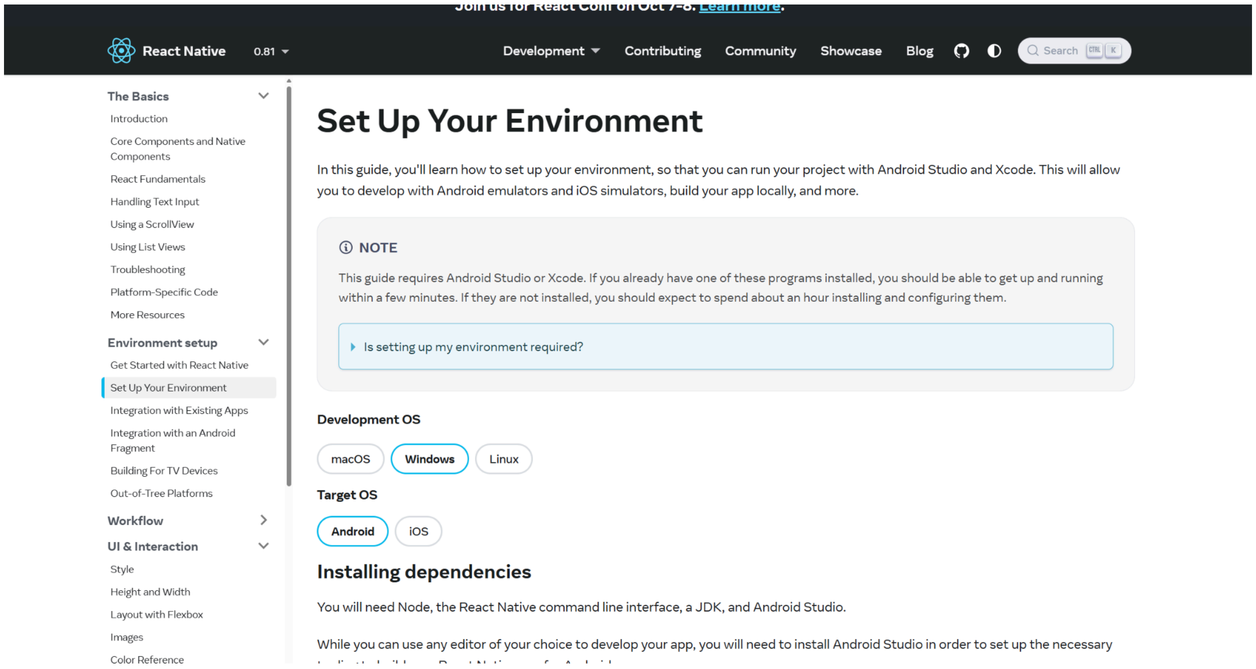
Start a new React Native project with Expo

Platform support

Android | iOS | TV | Web

Expo is a production-grade React Native Framework. Expo provides developer tooling that makes developing apps easier, such as file-

John's screenshot:



Join us for React Conf on Oct 7-8. Learn more.

React Native 0.81 | Development | Contributing | Community | Showcase | Blog | Search

The Basics
- Introduction
- Core Components and Native Components
- React Fundamentals
- Handling Text Input
- Using a ScrollView
- Using List Views
- Troubleshooting
- Platform-Specific Code
- More Resources

Environment setup
- Get Started with React Native
- Set Up Your Environment
- Integration with Existing Apps
- Integration with an Android Fragment
- Building For TV Devices
- Out-of-Tree Platforms

Workflow
UI & Interaction
- Style
- Height and Width
- Layout with Flexbox
- Images
- Color Reference

Set Up Your Environment

In this guide, you'll learn how to set up your environment, so that you can run your project with Android Studio and Xcode. This will allow you to develop with Android emulators and iOS simulators, build your app locally, and more.

NOTE

This guide requires Android Studio or Xcode. If you already have one of these programs installed, you should be able to get up and running within a few minutes. If they are not installed, you should expect to spend about an hour installing and configuring them.

Is setting up my environment required?

Development OS

macOS | [✓ Windows] | Linux

Target OS

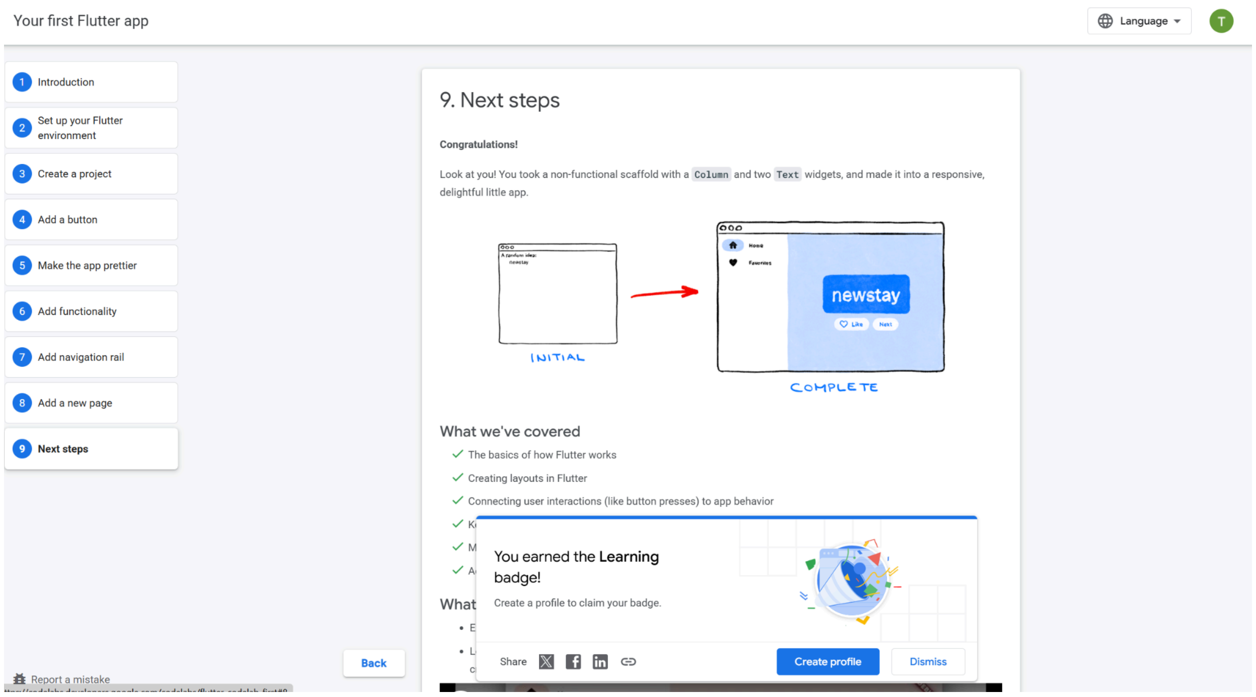
[✓ Android] | iOS

Installing dependencies

You will need Node, the React Native command line interface, a JDK, and Android Studio.

While you can use any editor of your choice to develop your app, you will need to install Android Studio in order to set up the necessary

Taylor's Screenshot:



9. Next steps

Congratulations!

Look at you! You took a non-functional scaffold with a `Column` and two `Text` widgets, and made it into a responsive, delightful little app.

What we've covered

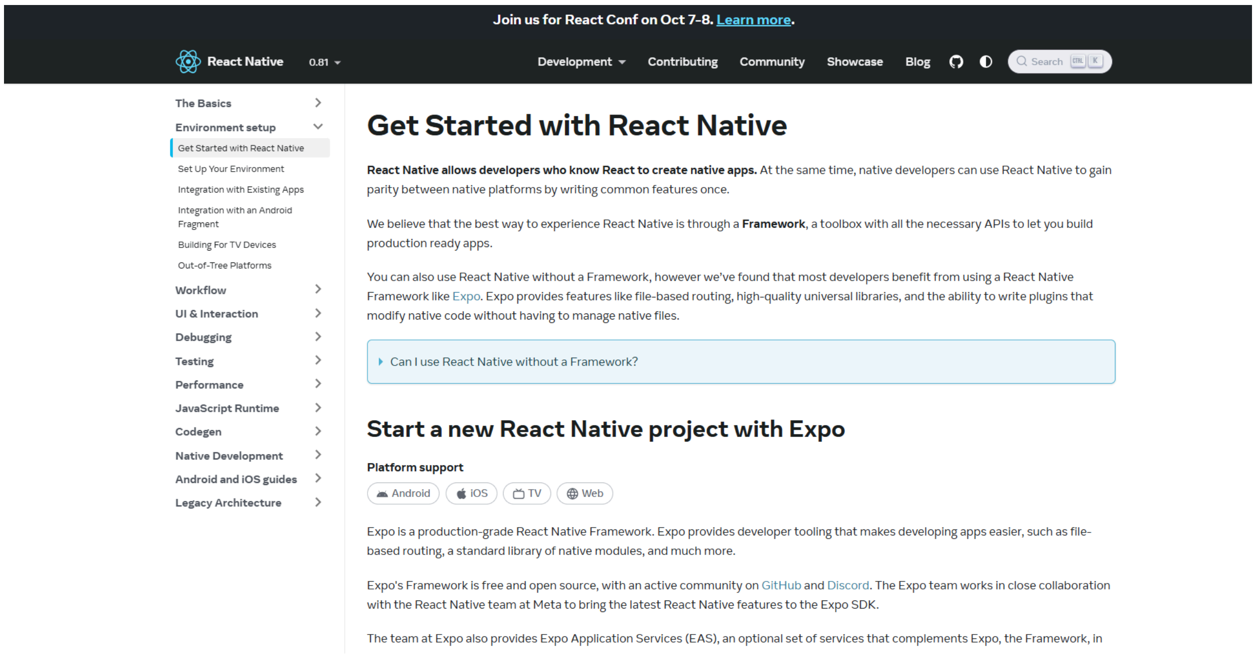
- ✓ The basics of how Flutter works
- ✓ Creating layouts in Flutter
- ✓ Connecting user interactions (like button presses) to app behavior
- ✓ K
- ✓ M
- ✓ A

What

- E
- L

You earned the **Learning** badge!

Create a profile to claim your badge.


Join us for React Conf on Oct 7-8. Learn more.

Get Started with React Native

React Native allows developers who know React to create native apps. At the same time, native developers can use React Native to gain parity between native platforms by writing common features once.

We believe that the best way to experience React Native is through a **Framework**, a toolbox with all the necessary APIs to let you build production ready apps.

You can also use React Native without a Framework, however we've found that most developers benefit from using a React Native Framework like Expo. Expo provides features like file-based routing, high-quality universal libraries, and the ability to write plugins that modify native code without having to manage native files.

▸ Can I use React Native without a Framework?

Start a new React Native project with Expo

Platform support

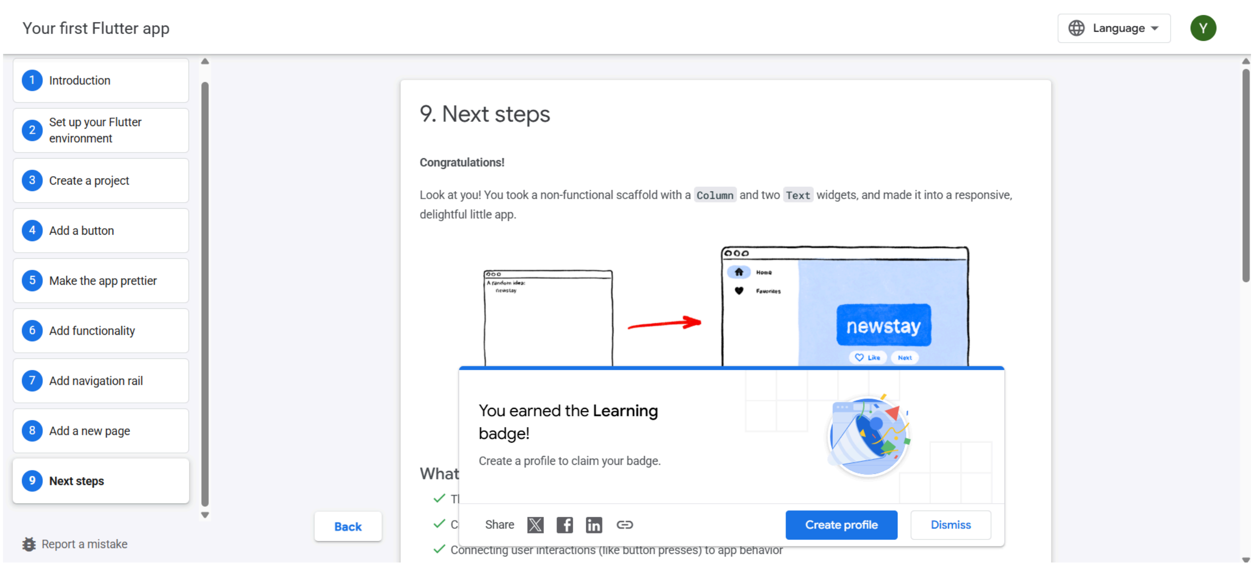
Android iOS TV Web

Expo is a production-grade React Native Framework. Expo provides developer tooling that makes developing apps easier, such as file-based routing, a standard library of native modules, and much more.

Expo's Framework is free and open source, with an active community on GitHub and Discord. The Expo team works in close collaboration with the React Native team at Meta to bring the latest React Native features to the Expo SDK.

The team at Expo also provides Expo Application Services (EAS), an optional set of services that complements Expo, the Framework, in

Yasmeen's Screenshot


Your first Flutter app

Language

1 Introduction
2 Set up your Flutter environment
3 Create a project
4 Add a button
5 Make the app prettier
6 Add functionality
7 Add navigation rail
8 Add a new page
9 **Next steps**

Report a mistake

Back

9. Next steps

Congratulations!

Look at you! You took a non-functional scaffold with a `Column` and two `Text` widgets, and made it into a responsive, delightful little app.

You earned the **Learning** badge!

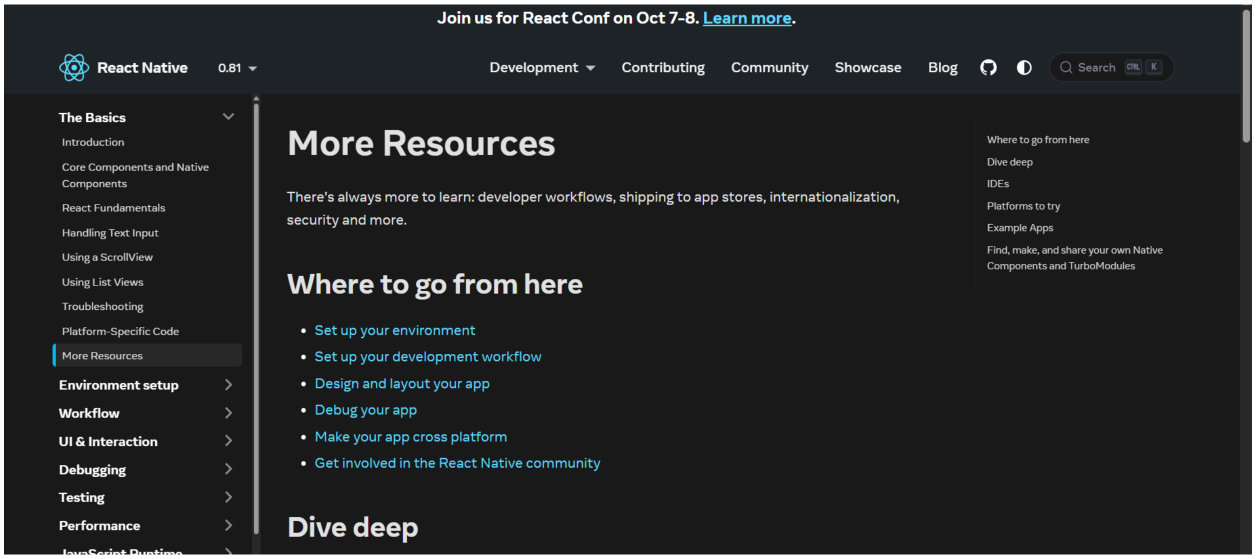
Create a profile to claim your badge.

Share

Create profile
Dismiss

What

✓ Connecting user interactions (like button presses) to app behavior


Join us for React Conf on Oct 7-8. Learn more.

React Native 0.81 Development Contributing Community Showcase Blog Search

The Basics
Introduction
Core Components and Native Components
React Fundamentals
Handling Text Input
Using a ScrollView
Using List Views
Troubleshooting
Platform-Specific Code
More Resources
Environment setup
Workflow
UI & Interaction
Debugging
Testing
Performance

More Resources

There's always more to learn: developer workflows, shipping to app stores, internationalization, security and more.

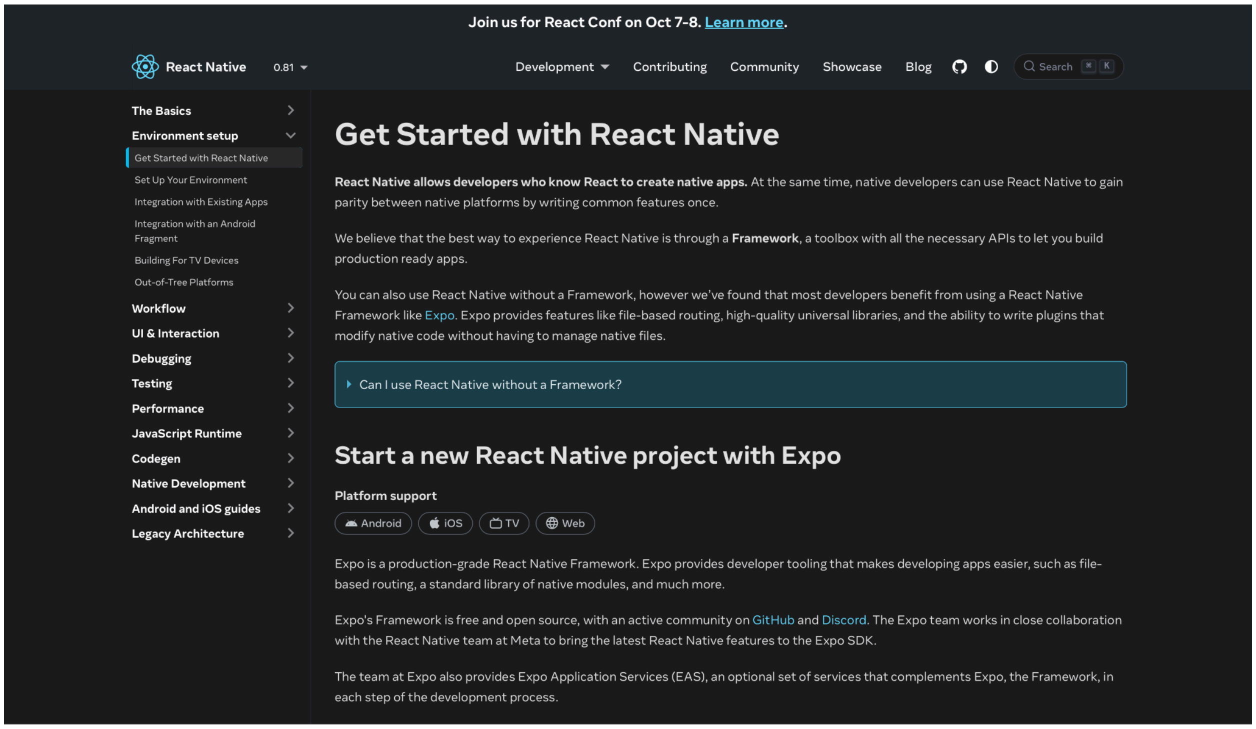
Where to go from here

- Set up your environment
- Set up your development workflow
- Design and layout your app
- Debug your app
- Make your app cross platform
- Get involved in the React Native community

Dive deep

Where to go from here
Dive deep
IDEs
Platforms to try
Example Apps
Find, make, and share your own Native Components and TurboModules

Sameer’s Screenshots:


Join us for React Conf on Oct 7-8. Learn more.

Get Started with React Native

React Native allows developers who know React to create native apps. At the same time, native developers can use React Native to gain parity between native platforms by writing common features once.

We believe that the best way to experience React Native is through a **Framework**, a toolbox with all the necessary APIs to let you build production ready apps.

You can also use React Native without a Framework, however we've found that most developers benefit from using a React Native Framework like Expo. Expo provides features like file-based routing, high-quality universal libraries, and the ability to write plugins that modify native code without having to manage native files.

Can I use React Native without a Framework?

Start a new React Native project with Expo

Platform support

Android iOS TV Web

Expo is a production-grade React Native Framework. Expo provides developer tooling that makes developing apps easier, such as file-based routing, a standard library of native modules, and much more.

Expo's Framework is free and open source, with an active community on GitHub and Discord. The Expo team works in close collaboration with the React Native team at Meta to bring the latest React Native features to the Expo SDK.

The team at Expo also provides Expo Application Services (EAS), an optional set of services that complements Expo, the Framework, in each step of the development process.