CS 4850 - Section 02 - Spring 2025

SP-26-Purple-Budget App
Final Report

June Kim, Chris Dinh, Caitlyn Alligood, Aldo Sanchez

Sharon Perry

04/27/2025

Website: https://chdinh25.github.io/chdinh.github.io/

Github: https://github.com/Purple-Budget-App/BudgetApp

Number of lines of code: 764

Number of project components/tools: 6 (Figma, React Native, Plaid API, Firebase Auth, Firebase Database, Railway)

Estimate man-hours: 230

Total man-hours: 260

Status: Project is 80% completed

Table of Contents

1.0 Introduction

The **Purple Budget App** is a cross-platform mobile application designed to promote financial literacy and responsible spending habits among young adults and college students. In an era where digital transactions and mobile banking have become the norm, managing one's finances through an integrated platform is becoming increasingly essential. This app addresses that need by offering users a secure and intuitive interface that provides real-time financial tracking, goal setting, and budget planning. Built using a combination of modern technologies, including React Native, Firebase, and the Plaid API, the application enables users to link their bank accounts and automatically track transactions securely. By visualizing expenses through dynamic charts and offering customizable budget categories, Purple Budget enables users to develop more effective money habits.

2.0 Requirements

Functional Requirements

Login

The main screen prompts users to log in or create an account if they don't already have one. The login screen includes a password recovery button for users who have forgotten their password. When creating an account, users are directed to a page where they can enter their first and last name, email address, and cell phone number, and create a password.

User Authentication

Users can log in using their email address and password, with the option to log in via OAuth using Google, Apple, or Facebook. Password recovery functionality is available via email. To create an account, users must enter their name, email, and password. Two-factor authentication (2FA) is implemented via email or SMS for added security.

Financial Account Integration

Users can link their bank accounts through the Plaid API, enabling the app to retrieve their transaction history, income, and expenses. Additionally, users can manually add cash transactions for complete tracking and record-keeping.

Budgeting & Expense Tracking

Users can set a monthly budget for categories such as food, rent, and shopping. The system tracks spending against the budget and sends alerts for any overspending.

Visual Data Representation

Users can view graphs comparing income and expenses, a category-wise spending distribution presented through pie charts, and trends over time using line graphs.

Notifications and Alerts

Users receive alerts when they exceed budget limits and receive monthly financial summary reports via email or in-app notifications.

Settings and Preferences

Users can customize budget categories, enable or disable notifications, and select from various UI customization options, including dark mode.

Display Home Page

Options available from the Home Page include checking your credit card balance, account balance, savings balance, and investments. From there, users can navigate to their profile, breakdown of their budget and spending, and to their savings.

Non-Functional Requirements

Security

Data encryption is implemented for stored financial data, utilizing secure OAuth authentication and two-factor authentication (2FA) for added protection.

Performance and Capacity

Fast API response times for real-time financial updates. The application will support thousands of concurrent users.

Usability

The UI/UX will be intuitive for easy **financial tracking**. There will be minimal setup required for **bank integration**.

External Interface Requirements

User Interface Requirements

Mobile-friendly UI optimized for both **iOS and Android**. There will be smooth, quick navigation between pages with minimal loading time.

Hardware Interface Requirements

The app is compatible with all smartphones running **iOS 13 or later or Android 8 or later**. Internet connectivity is required for real-time updates.

Software Interface Requirements

React Native is used for frontend integrations with a **Node** backend. **The Plaid API** is utilized for banking integration, while the **Firebase database** is employed for storing financial records.

Communication Interface Requirements

Communication between the frontend and backend ensures efficient data exchange, with secure transmission.

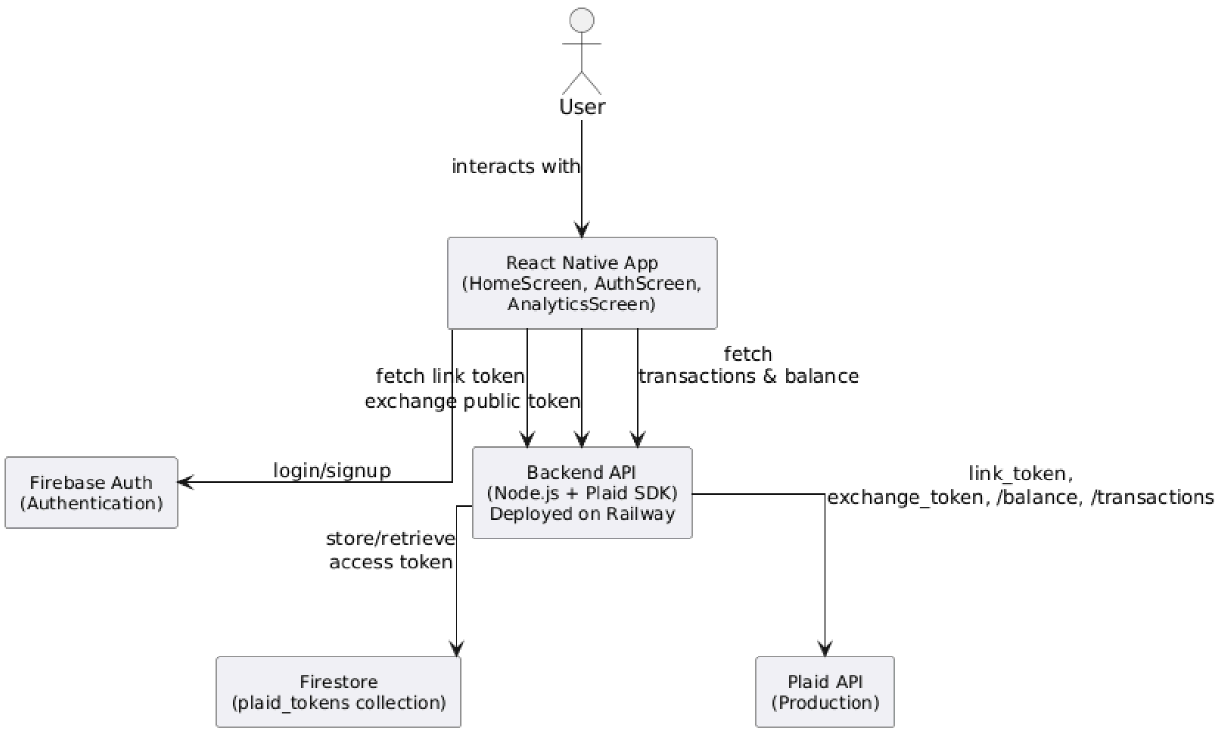
3.0 Analysis/Design

Related Software or Hardware

- The application is compatible with both Android and iOS devices.
- Plaid API will be used for secure financial data aggregation.
- The backend will be developed using Node.js and hosted on Railway.
- React Native will be used for front-end development.

End-User Characteristics

- The app is designed for college students and young adults who are starting to manage their finances.
- Users are expected to be mid-level tech-savvy and comfortable with mobile apps and online banking.
- The app should be intuitive, requiring minimal financial knowledge to get started.

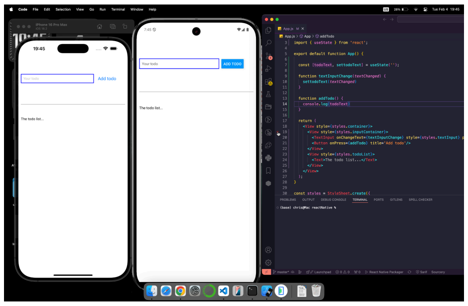

User
interacts with
React Native App (HomeScreen, AuthScreen, AnalyticsScreen)
fetch link token
exchange public token
fetch transactions & balance
login/signup
Firebase Auth (Authentication)
Backend API (Node.js + Plaid SDK) Deployed on Railway
link_token, exchange_token, /balance, /transactions
store/retrieve access token
Firestore (plaid_tokens collection)
Plaid API (Production)

4.0 Development

The development process followed agile principles, with tasks broken into sprints based on the project timeline. Initial planning was performed in Figma to outline UI flows. Firebase was set up early to manage user authentication and provide the database infrastructure. One of the key challenges was switching from Expo to React Native CLI to ensure compatibility with Plaid, a transition that required code refactoring and environment reconfiguration. Despite the setback, this pivot improved long-term stability and scalability of deployment.

Concepts

1. Front end



```
import { useState } from 'react';

export default function App() {

  const [todoText, settodoText] = useState('');

  function textInputChange(textChanged) {
    settodoText(textChanged)
  }

  function addTodo() {
    console.log(todoText)
  }

  return (
    <View style={styles.container}>
      <View style={styles.inputContainer}>
        <TextInput onChangeText={textInputChange} style={styles.textInput} p
        <Button onPress={addTodo} title='Add todo'/>
      </View>
      <View style={styles.todoList}>
        <Text>The todo list...</Text>
      </View>
    </View>
  );
}

const styles = StyleSheet.create({
```

```
(base) chris@Mac reactNative %
```

   React Native provides a solid foundation for the application, allowing for real-time phone emulation during development. Its cross-platform capabilities enable seamless deployment on both Android and iOS devices, ensuring a smooth user experience.

2. Back end

   Node.js API is utilized by the Railway hosting service to provide an isolation between client calls and token exchanges. Currently, the backend handles creating public tokens, exchanging the public token for an access token, saving the access token along with the user ID in the Firebase Database, and making transaction calls to retrieve transactions from Plaid servers.

3. Authentication
   Firebase Authentication is used to secure user authentication via Email/Password and Google Sign-In (pending), and stores the login data for our users.

4. Bank Link
   The Plaid API is integral to the app, enabling secure connections with financial institutions. Aggregating transaction data allows users to monitor their spending in real-time, ensuring accuracy and reliability.
   Plaid allows users to connect their bank accounts through Plaid and retrieve their balances and transactions. This enables us to analyze the data and identify trends over time, providing a comprehensive financial overview.

5. Display & Dashboard Module
   The screens will be created using JavaScript and/or TypeScript with React Native. At this time, we are still working on implementing previous concepts, so this part is not yet finalized.

   However, the team utilized Figma to create detailed screen mockups, ensuring a user-friendly and visually appealing interface. These mockups feature screens that showcase financial charts and transaction summaries, enabling users to interact more effectively with their data.

6. Tech Stack (Deployment & Hosting)
   Our current Tech Stack is as follows :

Component	Tech used
Frontend	React Native with React Native CLI
Backend	Node.js (Hosted on Railway)
Authentication	Firebase Auth (Email/password, Google sign-in)

Database	Firebase Cloud Firestore
Bank Integration	Plaid API
Deployment	Railway (Backend), Firebase (Firestore & Auth)

Database Connections

- Railway- used to host the backend server
- Cloud Firestore- used to store the access token when the Plaid API is linked.
- Firebase- used to create and authenticate users. It also stores the user account and password.
- Plaid is only contacted once the Link token has been created. The link token is exchanged for an access token once linked to a bank.

How to set up

- Set Up React Native Project
- Follow instructions from https://reactnative.dev/docs/getting-started-without-a-framework to set up a react-native project without Expo (Expo has trouble utilizing the Plaid API)
- Follow the instructions from https://reactnative.dev/docs/set-up-your-environment?os=windows to set up the environment for react native. This includes Node, the React Native command-line interface, a Java Development Kit (JDK), and Android Studio.
- Using npm, install dependencies.
- Firebase SDK for authentication
- Plaid SDK for financial data
- React Navigation for navigating through screens
- Implement Authentication Using Firebase
- Create an account with Firebase and create a new Firebase project
- Utilize the Firebase authentication module and enable the Email sign-in method
- Download the config files and implement them into the base code

- Use Cloud Firestore to store the access token given by the Plaid API
- Create a backend host to interact with the Plaid API
- Set up a server file that handles API calls to make tokens and connect with Plaid when required
- Use hosting such as Railway (one of our projects is using) to host the server
- Integrate Plaid API
- Use https://plaid.com/docs/link/react-native/ to be acquainted with calls, and reference the sample app https://github.com/plaid/tiny-quickstart/blob/main/react_native/README.md to create calls.
- Create Different Screens for users to interface with
- Create a backend to store data from the Plaid API to show users
- Manipulate Data to show graphs, balances, and transactions to users in a meaningful manner

5.0 Test Plan and Report

Software Test Plan

All four members of the group will perform tests. Tests will be evaluated against testing objectives and considered acceptable if all significant flows are stable, with no crashes, and the expected results are demonstrated.

The aspects that need to be tested are authentication (sign-up, login, and logout), token exchange, fetching balance and transactions, navigating between bottom tabs, and visualization of financial data.

Tests will be performed locally on an emulator with cloud deployment ready to go via Railway and Firebase hosting.

The 'Create Account' feature will be considered complete when a new user can use it and their account data is displayed on Firebase Auth.

Login is considered a pass when we verify that no random ID and password can be used to access the app, and the account created for the purpose is used to enter the app.

Logout is considered a pass when the logout button is pressed, and it sends the user back to the first screen, prompting them to log in again.

Navigation is considered passed when the user can click on different tabs at the bottom and can navigate through all pages without error.

Link to Plaid is considered a pass when a user can click the link to Plaid and successfully navigate through the pop-up to connect their account to a bank.

The show balance is considered passed when, after Plaid is linked, the account balance is displayed after clicking the Show Balance button.

Software Test Results

Requirement	Pass	Fail	Severity
Create account	✅		
Login	✅		
Log out	✅		
Link to Plaid	✅		
Show balance	✅		
Navigation	✅		

6.0 Version Control

Version control is managed using GitHub to track changes and facilitate efficient collaboration. Since we were a small team, we typically worked together when developing our app. One person would focus on the layout, while the other worked on the API. We would then commit our changes, test each iteration, and repeat the cycle.

7.0 Conclusion

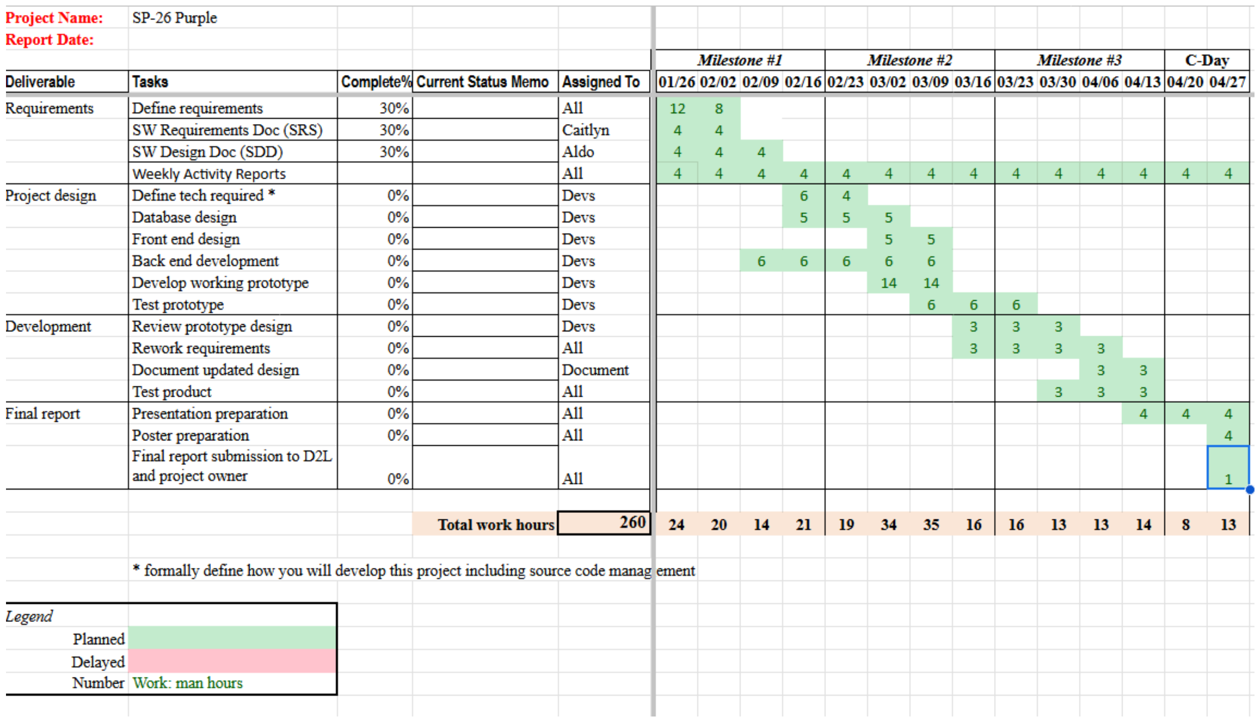
In summary, the Budgeting App provides a secure, intuitive, and effective solution for managing personal finances across multiple platforms. By integrating with Plaid.com, users can confidently connect their financial accounts to monitor income, expenses, and savings in one place. Enhanced by goal-setting tools, progress tracking, and visual aids like charts and graphs, this application empowers users to take control of their financial habits and make smarter, more informed decisions about their spending.

Lessons Learned

Through this project, we have learned the basics of creating a React Native app while utilizing multiple different tools and APIs to accomplish requirements. The biggest lessons were to choose the right tool for the right job and learn what that actually entails. For example, knowing that the EXPO doesn’t work well with the Plaid API saved us weeks of development, with almost no progress on understanding dependencies and finding and choosing to use the ones that aren’t deprecated. We learned about intercommunication within a small group, setting expectations, and how to utilize the SDLC to bring it all together in this project. By completing this project, we will gain more confidence in building apps and working collaboratively through the Software Development Life Cycle (SDLC) for future projects.

8.0 Appendix

Project Planning



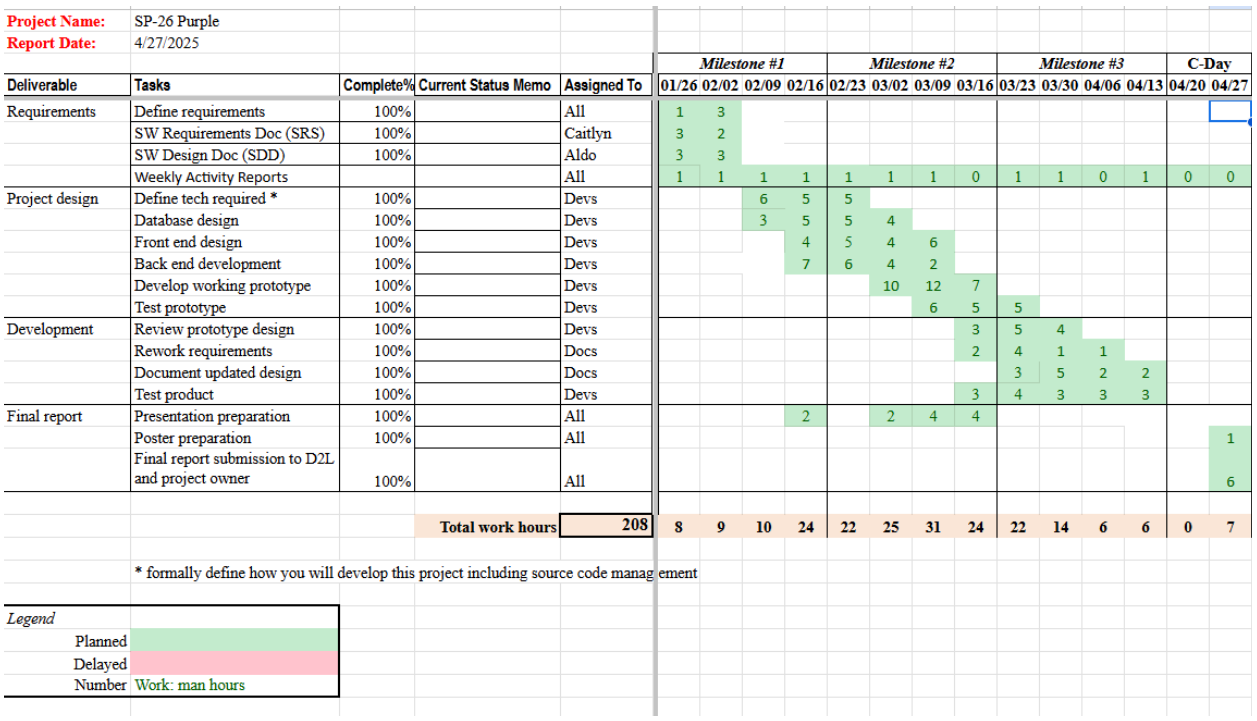
Project Name: SP-26 Purple
Report Date:

					Milestone #1				Milestone #2				Milestone #3				C-Day	
Deliverable	**Tasks**	**Complete%**	**Current Status Memo**	**Assigned To**	01/26	02/02	02/09	02/16	02/23	03/02	03/09	03/16	03/23	03/30	04/06	04/13	04/20	04/27
Requirements	Define requirements	30%		All	12	8												
	SW Requirements Doc (SRS)	30%		Caitlyn	4	4												
	SW Design Doc (SDD)	30%		Aldo	4	4	4											
	Weekly Activity Reports			All	4	4	4	4	4	4	4	4	4	4	4	4	4	4
Project design	Define tech required *	0%		Devs				6	4									
	Database design	0%		Devs				5	5	5								
	Front end design	0%		Devs						5	5							
	Back end development	0%		Devs			6	6	6	6	6							
	Develop working prototype	0%		Devs						14	14							
	Test prototype	0%		Devs							6	6	6					
Development	Review prototype design	0%		Devs								3	3	3				
	Rework requirements	0%		All								3	3	3	3			
	Document updated design	0%		Document											3	3		
	Test product	0%		All										3	3	3		
Final report	Presentation preparation	0%		All												4	4	4
	Poster preparation	0%		All														4
	Final report submission to D2L and project owner	0%		All														1
			Total work hours	**260**	**24**	**20**	**14**	**21**	**19**	**34**	**35**	**16**	**16**	**13**	**13**	**14**	**8**	**13**

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

Legend	
Planned	🟩
Delayed	🟥
Number	Work: man hours

Estimated Hours



Project Name: SP-26 Purple
Report Date: 4/27/2025

					Milestone #1				Milestone #2				Milestone #3				C-Day	
Deliverable	**Tasks**	**Complete%**	**Current Status Memo**	**Assigned To**	01/26	02/02	02/09	02/16	02/23	03/02	03/09	03/16	03/23	03/30	04/06	04/13	04/20	04/27
Requirements	Define requirements	100%		All	1	3												
	SW Requirements Doc (SRS)	100%		Caitlyn	3	2												
	SW Design Doc (SDD)	100%		Aldo	3	3												
	Weekly Activity Reports			All	1	1	1	1	1	1	1	0	1	1	0	1	0	0
Project design	Define tech required *	100%		Devs			6	5	5									
	Database design	100%		Devs			3	5	5	4								
	Front end design	100%		Devs				4	5	4	6							
	Back end development	100%		Devs				7	6	4	2							
	Develop working prototype	100%		Devs						10	12	7						
	Test prototype	100%		Devs							6	5	5					
Development	Review prototype design	100%		Devs								3	5	4				
	Rework requirements	100%		Docs								2	4	1	1			
	Document updated design	100%		Docs									3	5	2	2		
	Test product	100%		Devs								3	4	3	3	3		
Final report	Presentation preparation	100%		All				2		2	4	4						
	Poster preparation	100%		All														1
	Final report submission to D2L and project owner	100%		All														6
			Total work hours	**208**	**8**	**9**	**10**	**24**	**22**	**25**	**31**	**24**	**22**	**14**	**6**	**6**	**0**	**7**

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

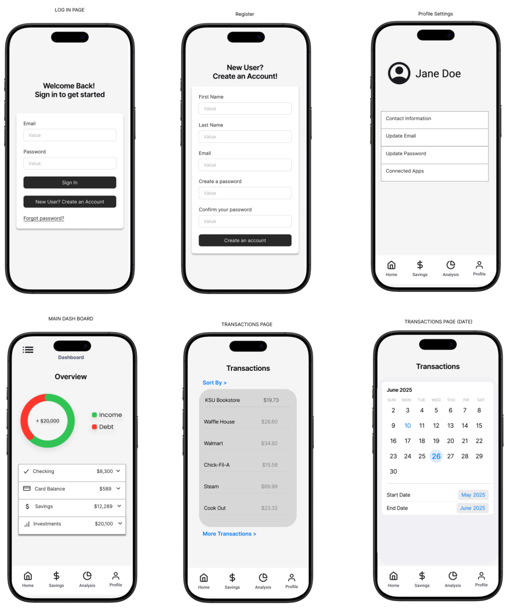
Legend	
Planned	🟩
Delayed	🟥
Number	Work: man hours

<u>Actual Hours</u>

The project adopted a more iterative approach, as some sections took significantly longer than others. For example, the setup of React Native took longer than expected, pushing us back a couple of weeks. Using Expo for testing the environment, and then realizing Plaid doesn't work well with Expo, led us to scrap the entire project and try the React Native CLI, which provided better results.

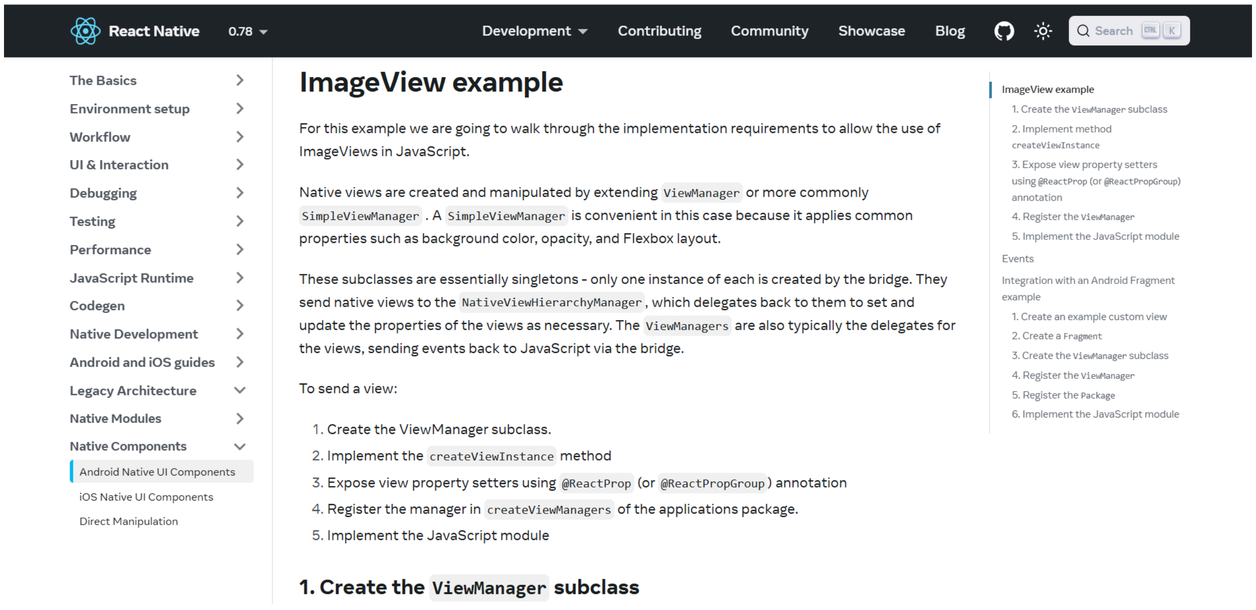
Mockups

- Login Page
- Account Creation
- Dashboard
- Profile/Setting
- Transactions



Team Member Tutorials

Caitlyn Alligood completed the react native tutorial.


React Native
0.78
Development
Contributing
Community
Showcase
Blog
Search
The Basics
Environment setup
Workflow
UI & Interaction
Debugging
Testing
Performance
JavaScript Runtime
Codegen
Native Development
Android and iOS guides
Legacy Architecture
Native Modules
Native Components
Android Native UI Components
iOS Native UI Components
Direct Manipulation
ImageView example
1. Create the ViewManager subclass
2. Implement method createViewInstance
3. Expose view property setters using @ReactProp (or @ReactPropGroup) annotation
4. Register the ViewManager
5. Implement the JavaScript module
Events
Integration with an Android Fragment example
1. Create an example custom view
2. Create a Fragment
3. Create the ViewManager subclass
4. Register the ViewManager
5. Register the Package
6. Implement the JavaScript module

ImageView example

For this example we are going to walk through the implementation requirements to allow the use of ImageViews in JavaScript.

Native views are created and manipulated by extending `ViewManager` or more commonly `SimpleViewManager` . A `SimpleViewManager` is convenient in this case because it applies common properties such as background color, opacity, and Flexbox layout.

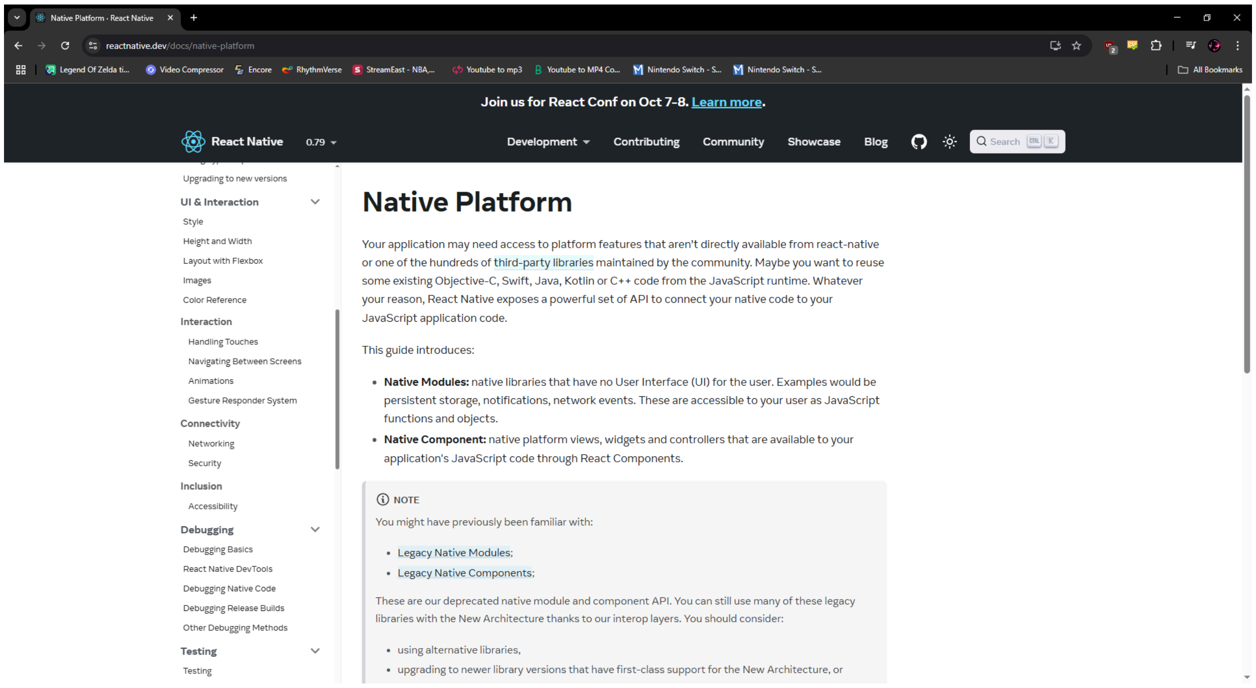
These subclasses are essentially singletons - only one instance of each is created by the bridge. They send native views to the `NativeViewHierarchyManager`, which delegates back to them to set and update the properties of the views as necessary. The `ViewManagers` are also typically the delegates for the views, sending events back to JavaScript via the bridge.

To send a view:

1. Create the ViewManager subclass.
2. Implement the `createViewInstance` method
3. Expose view property setters using `@ReactProp` (or `@ReactPropGroup`) annotation
4. Register the manager in `createViewManagers` of the applications package.
5. Implement the JavaScript module

1. Create the `ViewManager` subclass

Aldo Sanchez completed the react native tutoria.



Native Platform

Your application may need access to platform features that aren't directly available from react-native or one of the hundreds of third-party libraries maintained by the community. Maybe you want to reuse some existing Objective-C, Swift, Java, Kotlin or C++ code from the JavaScript runtime. Whatever your reason, React Native exposes a powerful set of API to connect your native code to your JavaScript application code.

This guide introduces:

- **Native Modules:** native libraries that have no User Interface (UI) for the user. Examples would be persistent storage, notifications, network events. These are accessible to your user as JavaScript functions and objects.
- **Native Component:** native platform views, widgets and controllers that are available to your application's JavaScript code through React Components.

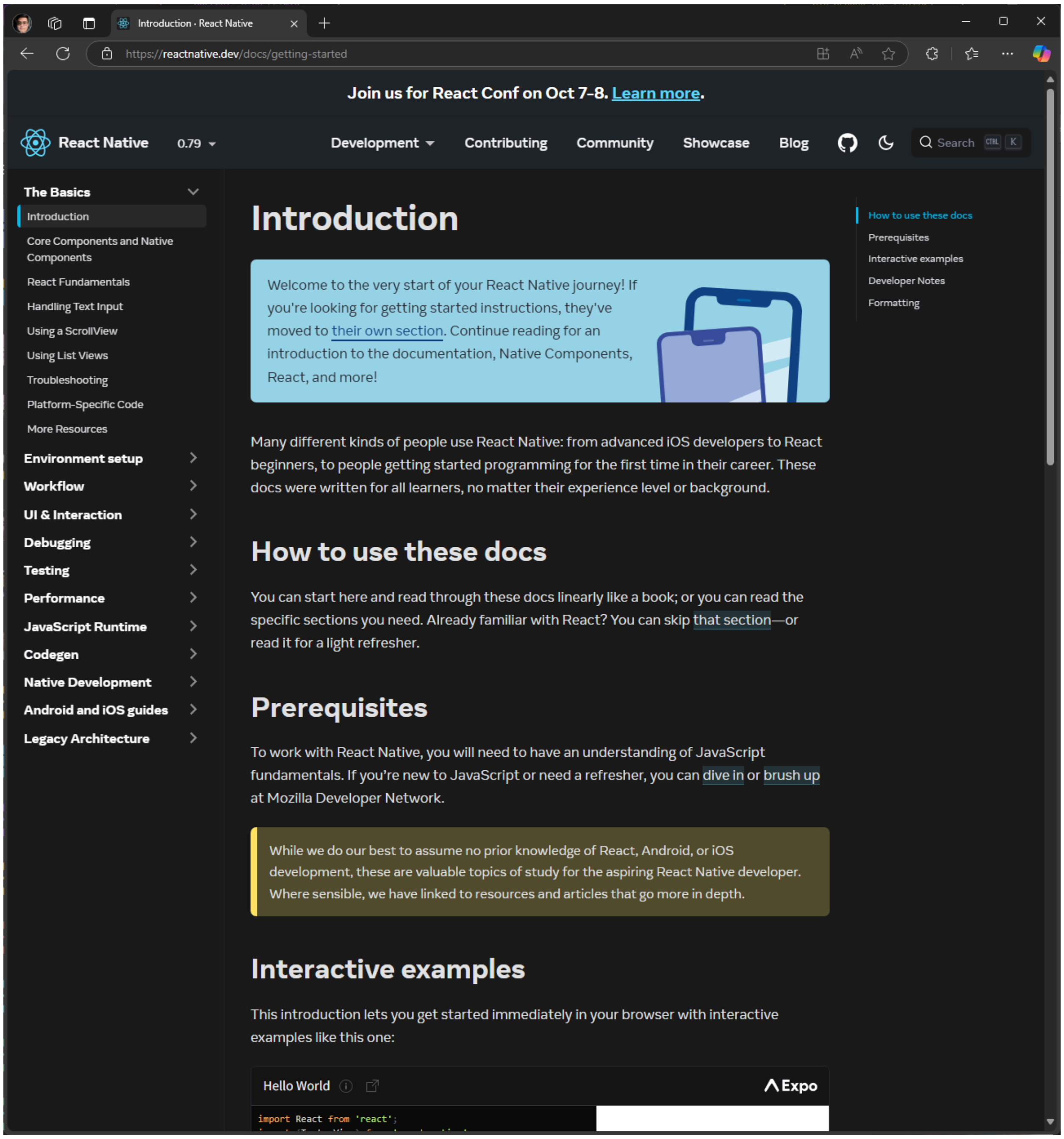
NOTE

You might have previously been familiar with:

- Legacy Native Modules;
- Legacy Native Components;

These are our deprecated native module and component API. You can still use many of these legacy libraries with the New Architecture thanks to our interop layers. You should consider:

- using alternative libraries,
- upgrading to newer library versions that have first-class support for the New Architecture, or



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

Introduction

Welcome to the very start of your React Native journey! If you're looking for getting started instructions, they've moved to their own section. Continue reading for an introduction to the documentation, Native Components, React, and more!

Many different kinds of people use React Native: from advanced iOS developers to React beginners, to people getting started programming for the first time in their career. These docs were written for all learners, no matter their experience level or background.

How to use these docs

You can start here and read through these docs linearly like a book; or you can read the specific sections you need. Already familiar with React? You can skip that section—or read it for a light refresher.

Prerequisites

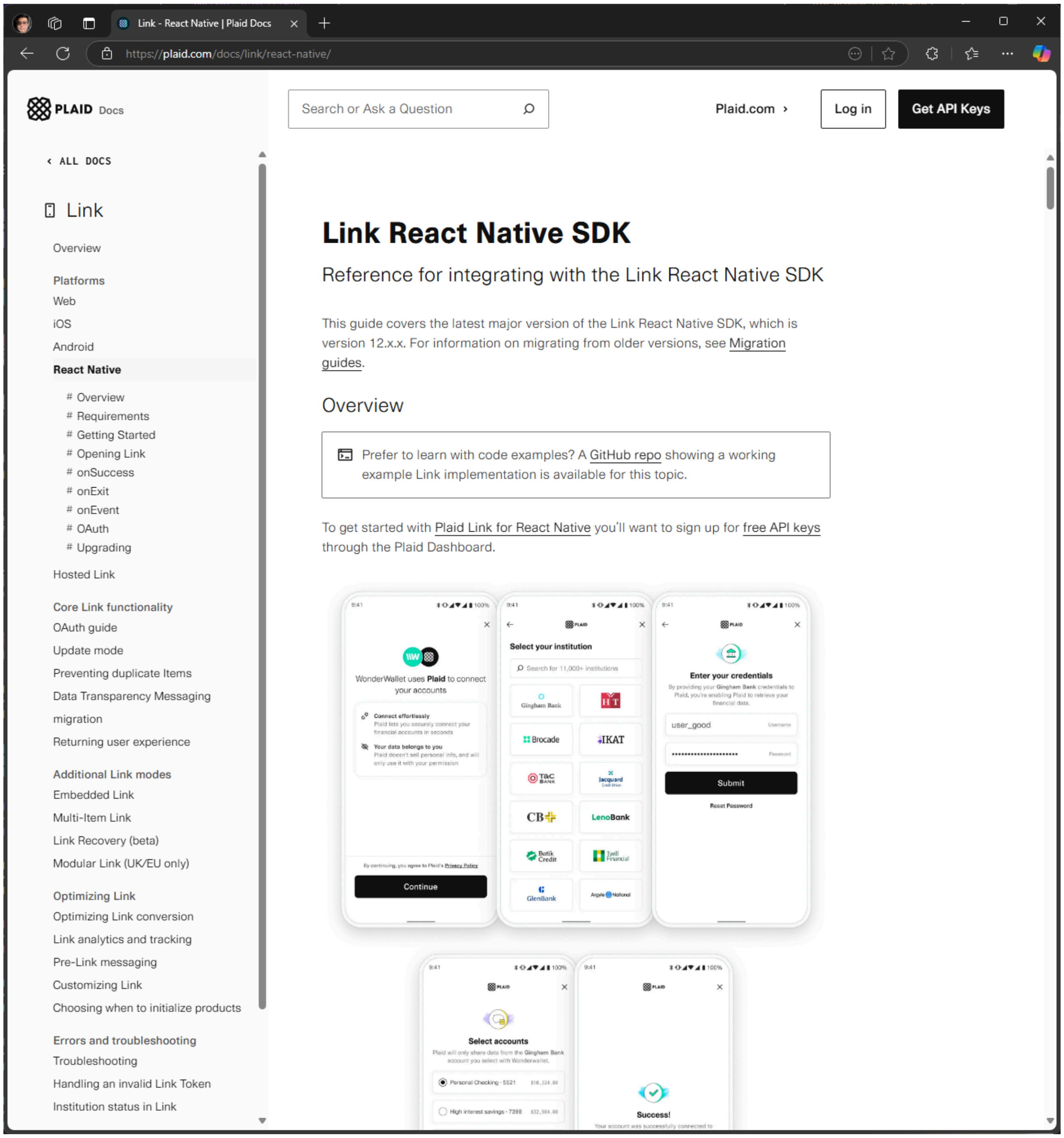
To work with React Native, you will need to have an understanding of JavaScript fundamentals. If you're new to JavaScript or need a refresher, you can dive in or brush up at Mozilla Developer Network.

While we do our best to assume no prior knowledge of React, Android, or iOS development, these are valuable topics of study for the aspiring React Native developer. Where sensible, we have linked to resources and articles that go more in depth.

Interactive examples

This introduction lets you get started immediately in your browser with interactive examples like this one:

```
import React from 'react';
```



Link - React Native | Plaid Docs

https://plaid.com/docs/link/react-native/

PLAID Docs

Search or Ask a Question

Plaid.com ›

Log in

Get API Keys

‹ ALL DOCS

Link

Overview

Platforms
Web
iOS
Android
React Native

Overview
Requirements
Getting Started
Opening Link
onSuccess
onExit
onEvent
OAuth
Upgrading

Hosted Link

Core Link functionality
OAuth guide
Update mode
Preventing duplicate Items
Data Transparency Messaging
migration
Returning user experience

Additional Link modes
Embedded Link
Multi-Item Link
Link Recovery (beta)
Modular Link (UK/EU only)

Optimizing Link
Optimizing Link conversion
Link analytics and tracking
Pre-Link messaging
Customizing Link
Choosing when to initialize products

Errors and troubleshooting
Troubleshooting
Handling an invalid Link Token
Institution status in Link

Link React Native SDK

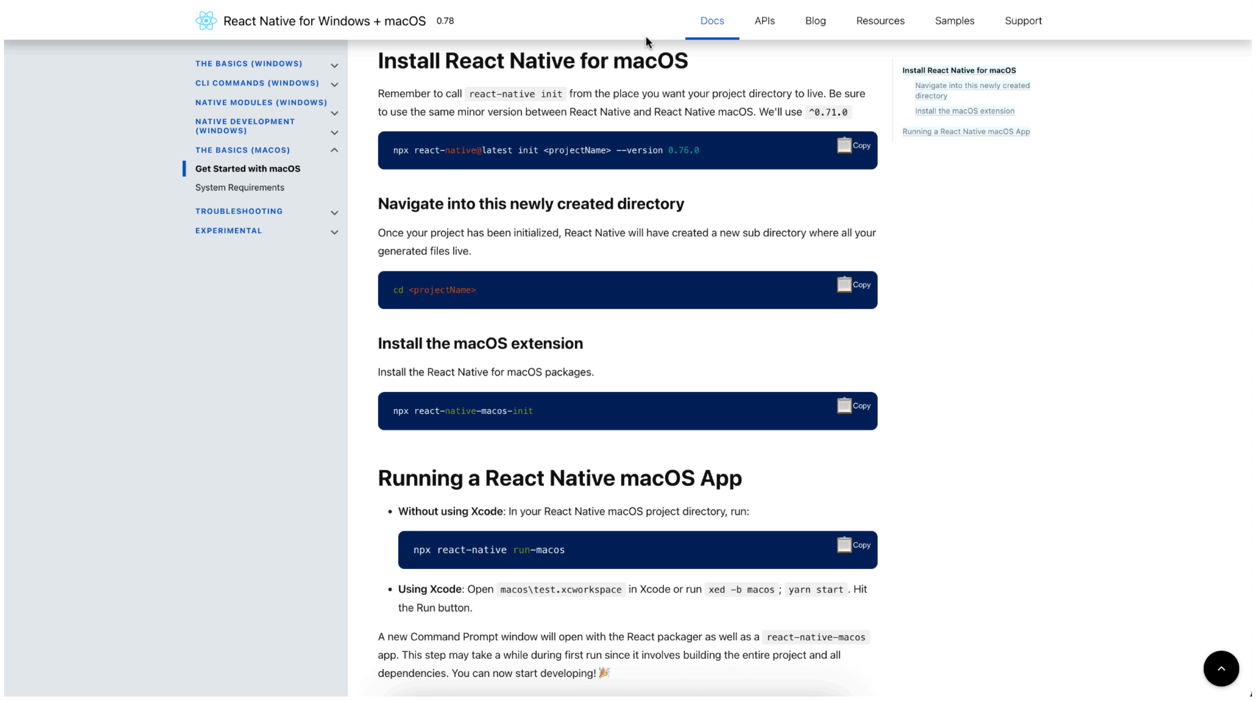
Reference for integrating with the Link React Native SDK

This guide covers the latest major version of the Link React Native SDK, which is version 12.x.x. For information on migrating from older versions, see Migration guides.

Overview

Prefer to learn with code examples? A GitHub repo showing a working example Link implementation is available for this topic.

To get started with Plaid Link for React Native you'll want to sign up for free API keys through the Plaid Dashboard.



React Native for Windows + macOS 0.78

Docs APIs Blog Resources Samples Support

THE BASICS (WINDOWS)
CLI COMMANDS (WINDOWS)
NATIVE MODULES (WINDOWS)
NATIVE DEVELOPMENT (WINDOWS)
THE BASICS (MACOS)
Get Started with macOS
System Requirements
TROUBLESHOOTING
EXPERIMENTAL

Install React Native for macOS

Remember to call `react-native init` from the place you want your project directory to live. Be sure to use the same minor version between React Native and React Native macOS. We'll use `^0.71.0`

```
npx react-native@latest init <projectName> --version 0.76.0
```

Navigate into this newly created directory

Once your project has been initialized, React Native will have created a new sub directory where all your generated files live.

```
cd <projectName>
```

Install the macOS extension

Install the React Native for macOS packages.

```
npx react-native-macos-init
```

Running a React Native macOS App

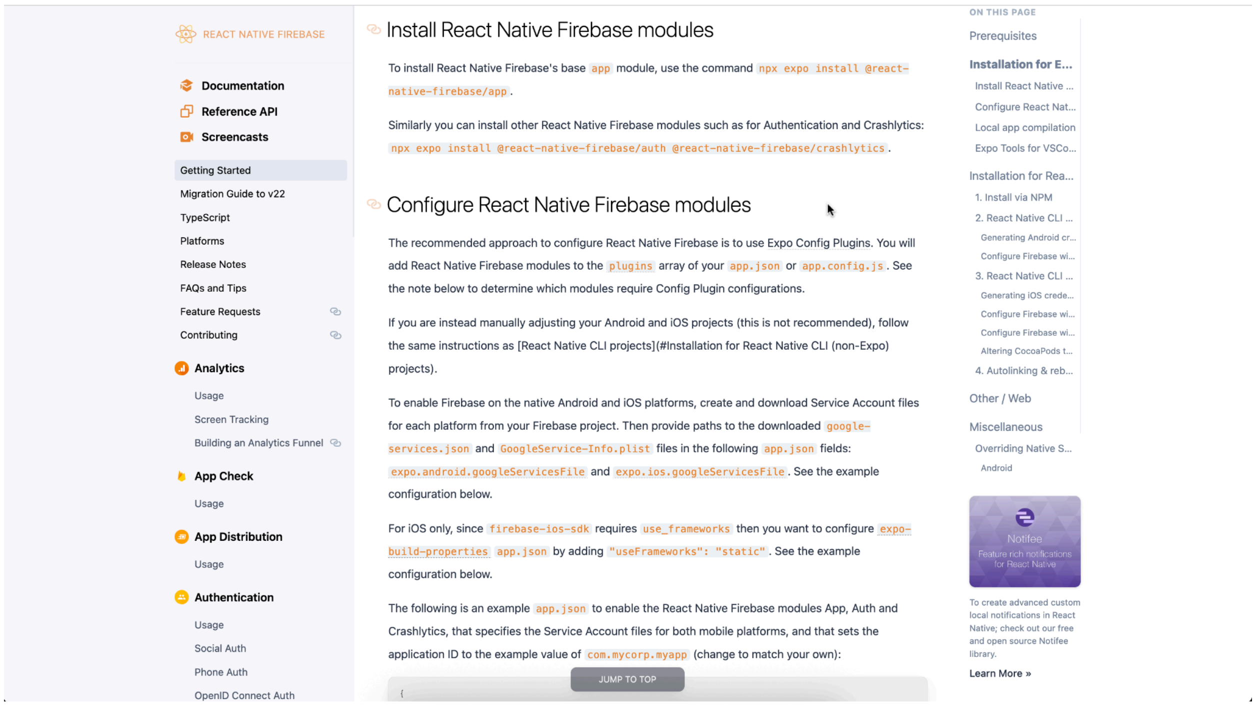
- **Without using Xcode**: In your React Native macOS project directory, run:

```
npx react-native run-macos
```

- **Using Xcode**: Open `macos\test.xcworkspace` in Xcode or run `xed -b macos`; `yarn start`. Hit the Run button.

A new Command Prompt window will open with the React packager as well as a `react-native-macos` app. This step may take a while during first run since it involves building the entire project and all dependencies. You can now start developing! 🎉

Install React Native for macOS
Navigate into this newly created directory
Install the macOS extension
Running a React Native macOS App



REACT NATIVE FIREBASE

Documentation
Reference API
Screencasts
Getting Started
Migration Guide to v22
TypeScript
Platforms
Release Notes
FAQs and Tips
Feature Requests
Contributing
Analytics
Usage
Screen Tracking
Building an Analytics Funnel
App Check
Usage
App Distribution
Usage
Authentication
Usage
Social Auth
Phone Auth
OpenID Connect Auth

Install React Native Firebase modules

To install React Native Firebase's base `app` module, use the command `npx expo install @react-native-firebase/app`.

Similarly you can install other React Native Firebase modules such as for Authentication and Crashlytics: `npx expo install @react-native-firebase/auth @react-native-firebase/crashlytics`.

Configure React Native Firebase modules

The recommended approach to configure React Native Firebase is to use Expo Config Plugins. You will add React Native Firebase modules to the `plugins` array of your `app.json` or `app.config.js`. See the note below to determine which modules require Config Plugin configurations.

If you are instead manually adjusting your Android and iOS projects (this is not recommended), follow the same instructions as [React Native CLI projects](#Installation for React Native CLI (non-Expo) projects).

To enable Firebase on the native Android and iOS platforms, create and download Service Account files for each platform from your Firebase project. Then provide paths to the downloaded `google-services.json` and `GoogleService-Info.plist` files in the following `app.json` fields: `expo.android.googleServicesFile` and `expo.ios.googleServicesFile`. See the example configuration below.

For iOS only, since `firebase-ios-sdk` requires `use_frameworks` then you want to configure `expo-build-properties` `app.json` by adding `"useFrameworks": "static"`. See the example configuration below.

The following is an example `app.json` to enable the React Native Firebase modules App, Auth and Crashlytics, that specifies the Service Account files for both mobile platforms, and that sets the application ID to the example value of `com.mycorp.myapp` (change to match your own):

```
{
```

JUMP TO TOP

ON THIS PAGE
Prerequisites
Installation for E...
Install React Native ...
Configure React Nat...
Local app compilation
Expo Tools for VSCo...
Installation for Rea...
1. Install via NPM
2. React Native CLI ...
Generating Android cr...
Configure Firebase wi...
3. React Native CLI ...
Generating iOS crede...
Configure Firebase wi...
Configure Firebase wi...
Altering CocoaPods t...
4. Autolinking & reb...
Other / Web
Miscellaneous
Overriding Native S...
Android

Notifee
Feature rich notifications for React Native

To create advanced custom local notifications in React Native; check out our free and open source Notifee library.

Learn More »