OG-Planit

System Design Document

An overview of system design specifications such as technologies used, compatibility, architecture, and more.

Project OG-Planit
10-22-2019

Table of Contents

Components, Responsibilities, Collaborators

Front-End Relationships

App	
• Initialize Fonts & Redux configurations • Create navigation stacks & switches	• Main

Main	
• Render landing page	• Login

Login	
• Render Login Page • Authenticate user via pre-defined methods • Record “accessToken” and other user data into Redux store	• SignUp • Home

SignUp	
• Render Sign Up page • Validate new user information • Create new user via HTTP request to Server	• N/A

Home	
• Render Home Page • Allow navigation to viewing itineraries, or creating a rating • Display user profile information, recommendations, and other data	• Itinerary

Itinerary	
• Render Itinerary Page • Allow user to view events that are part of their itinerary • Provide detailed information for each event • Modify itinerary through HTTP request to Server • Allow user to rate an event	• CreateRatings • GMap

CreateRatings	
• Render Create Ratings Page • Allow user to review events they have been to • Send HTTP requests to Server to update recommendations for the user	• N/A

GMap	
• Render Google Maps • Show users all the events in their itinerary, as well as detailed information for each event • Allow navigation between events	• N/A

Back-End Relationships

Server (Express.js)	
• Initialize API routes • Initialize admin SDK for Firebase • Accept/Reject incoming HTTP requests	• Routes • Firebase-Admin

Routes	
• Handle executing on incoming request data • Modify Firestore DB with new data	• Firebase-Admin

System Interaction and Environment

The Planit application will be build using the following main technologies:

- Front-End: React Native, Native Base UI Library, Firebase Web SDK, Expo (for deployments)
- Back-End: Node.js, Express.js, Firebase Admin SDK
- Database: Google's Cloud Firestore (NoSQL Database)

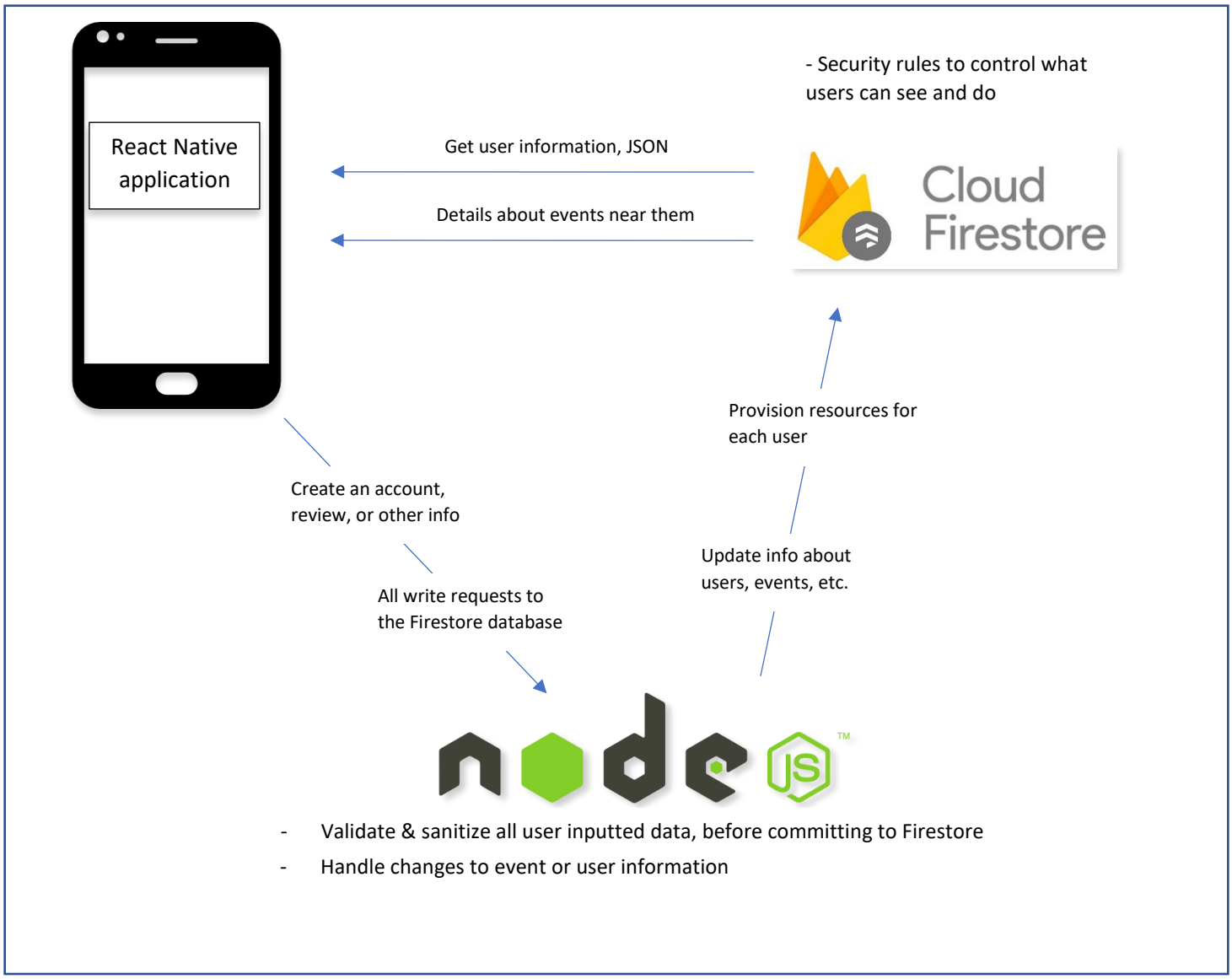
Through the Expo framework, the application will be built for both Android and iOS platforms, with the following restrictions (due to API requirements):

- Android: Requires Android 5 or greater
- iOS: Requires iOS 10 or greater

The front-end will be run via installing an APK file (for Android) or IPA file (for iOS). As for the Node.js server, it will be hosted by either Heroku or self-hosting on an exposed home-network through a Raspberry Pi. Finally, the database is hosted in the cloud by Google, and all connections to it will be through the Admin/Web SDK.

System Architecture

The application will follow a modified 3-tier architecture as shown:



Not all requests will be handled by the Node.js server. With the use of Security Rules in Firestore, we are able to authenticate users and restrict what data they can read with very granular control. However, for actions such as creating a new user, adding a review to an event, and any other write actions must be validated by a trusted source (Node.js server) before being committed to the database.

System Decomposition

Front-end

The React Native front-end will handle displaying all UI elements as we as handling logic for itinerary information, navigation between events, and more. All data will be cached on the client-side so that the user can exit the app and resume at any time. Caching is one of the major reasons as to why the user is able to query the Firestore NoSQL database directly; by utilizing the Firebase Web SDK, we are able to deliver high-performance and fast response times without having to proxy simple HTTP GET requests through a Node.js server.

By removing this entity in the middle (for GET requests only), we are able to load data much quicker and more reliably. Further, the use of the Firebase SDK grants offline-support with little to no setup required. So, a user can safely go offline for short periods of time and continue to enjoy the app.

Majority of the errors that can occur internally, for the front-end, are during HTTP requests. Such errors will be handled gracefully via well-placed catch statements and various call-back functions to ensure stability and responsiveness of the app. The user will also be notified of any issues that occur, such as HTTP requests failing to send due to the user being offline, authentication errors, and more. For adding data to the Firestore database, such as when signing up a new user or adding a review for an event, the requests will be sent to the Node.js server for validation before being committed. This is enforced at the database level with the use of security rules in Firestore.

Back-end

The Node.js server will be equipped with many tools to validate and sanitize user-inputted data. In the event of errors occurring with validation, the client will be notified immediately via response codes and messages. Further, in the event of an error when communicating with the Firestore Database the user will be notified immediately, although it is very limited as to what the Node.js server can do in this scenario (an administrator will likely have to go in manually to diagnose and fix the issue(s)). Another possible error that can occur is when the user is attempting to authenticate. In the event that a user's access token is invalid, their request will be promptly rejected with the appropriate response code and message.

Database

The database will validate all incoming GET requests from the Firebase SDK via security rules. These help us validate users without a dedicated server running. The database will store all user and event data with restrictions on what each user is able to see. Some errors that can occur with this is when a user requests to retrieve information that they are not provisioned to or that doesn't exit, which will promptly return a 4xx/5xx error (and be caught by the client-side). Another issue that can occur is when the project has reached its monthly quota on reads/writes/deletes for the Firestore database, in which

case all requests will likely be rejected until the next billing cycle (and the quick fix is to just buy a subscription, but we’re too broke for that).