SOS App Documentation

Prepared by: Jordan and Amos
Date: 13/11/2025
Project: SOS App

Version: SOS V2.1

Table of Contents

1. Introduction

This document outlines the design, structure, and implementation of the **SOS app**, a cross-platform mobile application developed using **React Native**. The app integrates **Firebase** as its backend service provider for authentication, data storage, and cloud functions, while **Stripe** is implemented for handling secure online payments.

The SOS app is designed with a core focus on **user safety, reliability, and accessibility**. It allows users to send emergency alerts to trusted contacts, share their live location, and access a variety of features aimed at providing real-time assistance during emergencies. By integrating Firebase and Stripe, the app not only ensures dependable cloud-based operations but also supports scalable growth through a secure and user-friendly payment system.

This documentation serves as a complete reference for developers, testers, and stakeholders involved in the project. It provides detailed explanations of how the SOS app's backend, frontend, and payment systems interact, ensuring clarity in both the technical and operational design. It covers everything from app architecture, data handling, and authentication, to the integration of third-party services like Stripe for premium feature access.

The ultimate goal of the SOS app is to provide a **comprehensive emergency response platform** that is secure, fast, and dependable, while maintaining simplicity in design and functionality.

2. Objectives

- Build a **cross-platform mobile application** using React Native.
- Integrate **Firebase backend services** for authentication, database storage, and cloud functionality.
- Ensure **secure and efficient data handling**.
- Implement **Stripe payment integration** for premium features.
- Provide a **user-friendly interface** that meets usability standards.
- Establish a **maintainable architecture** for future scalability.

3. Project Overview

The SOS app allows users to send emergency alerts, manage emergency contacts, and access premium services that enhance user safety and responsiveness.

Key technologies used:

- **Frontend:** React Native
- **Backend/Cloud Services:** Firebase (Authentication, Firestore, Cloud Storage, Cloud Functions)
- **Payment System:** Stripe Integration
- **State Management:** Redux / Context API
- **Navigation:** React Navigation • **Other Tools:** Expo, Axios, etc.

All code components provided in the project are explained individually to clarify their purpose, functionality, and interaction with Firebase and Stripe.

4. System Requirements Hardware Requirements

- Smartphone or emulator with iOS/Android support
- Minimum RAM: 4GB
- Storage: 500MB free

Software Requirements

- Node.js (LTS version)
- React Native CLI / Expo CLI
- Firebase account
- Stripe developer account
- IDE: VS Code / Android Studio / Xcode
- Emulator or physical device for testing

5. Firebase Configuration Step 1: Create Firebase Project

Create a new project in the Firebase console and enable Authentication, Firestore, Storage, and Analytics.

Step 2: Configure Authentication

Enable preferred sign-in methods such as Email/Password or Google Sign-In.

Explanation: Authentication ensures secure account creation and login for users through Firebase's built-in security.

Step 3: Configure Firestore

Define collections such as users, alerts, and contacts for structured data storage.

Explanation: Each collection acts as a container for related data, providing organized and efficient real-time updates.

Step 4: Configure Storage

Set up Firebase Storage for uploading files securely with defined access rules.

Explanation: This allows the app to handle user-generated content, including images or documents, with appropriate access restrictions.

6. App Structure and Architecture

- **Component-Based Architecture:** The app is built using modular components that are reusable across screens.
- **Screens:** Each screen handles a specific function, such as profile management or emergency alerts.
- **State Management:** Global state is managed through Redux or Context API to synchronize data.
- **Navigation:** Stack and tab navigation ensure smooth movement across screens.

Explanation: The modular and layered structure ensures maintainability, scalability, and clean code separation between UI and logic.

7. Key Features

1. **User Authentication:** Secure user login, registration, and password reset via Firebase Auth.
2. **Emergency Alerts:** Allows users to send emergency notifications to saved contacts.
3. **Real-Time Location Tracking:** Shares the user's location during emergencies.
4. **Data Storage:** Stores user and alert data in Firestore.
5. **File Uploads:** Handles uploads to Firebase Storage.
6. **Offline Support:** Firestore persistence enables offline access to cached data.
7. **Premium Payments:** Users can upgrade to premium through Stripe integration.

8. Workflow / User Flow

1. The user opens the app.
2. Logs in or registers via Firebase Authentication.
3. Accesses the home screen to send alerts or manage contacts.
4. Sends SOS alerts which include real-time location data.
5. Upgrades to a premium plan through the Payment Gateway.
6. Views profile and manages settings.

Explanation: Each stage of this flow is supported by secure API calls and synchronized updates between Firebase and the frontend.

9. Data Handling and Storage

- **Firestore:** Contains collections for users, alerts, and contacts with proper access rules.
- **Security Rules:** Limit read/write access to authorized users.
- **Data Validation:** Prevents invalid data submission using client-side checks.
- **Storage:** Securely stores uploaded content in Firebase Storage.

Explanation: The system ensures that all user data remains secure, consistent, and retrievable in real time.

10. Authentication and Security

- Uses Firebase Authentication for user management.
- Enforces HTTPS and strong password policies.
- Implements Firebase Security Rules for database protection.

Explanation: Each interaction with Firebase services is authorized through secure tokens to prevent unauthorized access.

11. Error Handling and Logging

- Displays clear error messages for failed operations.
- Integrates Firebase Crashlytics to log runtime errors.
- Uses console logging during development for easier debugging.

Explanation: This ensures reliable app behavior and helps identify and resolve issues promptly.

12. Testing and Debugging

- **Unit Testing:** Ensures functional components behave as expected.
- **Integration Testing:** Validates Firebase and Stripe interactions.
- **End-to-End Testing:** Simulates user actions on both Android and iOS.

Explanation: Structured testing ensures that critical workflows, such as payments and authentication, remain functional under various conditions.

13. Payment Gateway Screen

The **Payment Gateway Screen** handles secure payment transactions within the SOS app. It integrates **Stripe** for online payments and **Firebase** for data updates postpayment.

Overview

This screen provides users with a safe and intuitive way to complete premium subscriptions using debit or credit cards. It validates user input, processes payments, and updates the user's premium status in Firebase. **Functional Breakdown**

1. **Imports and Dependencies**
   Includes essential libraries such as React Native components for the UI, Stripe for payment processing, and Firebase Firestore for database operations. It also uses custom hooks like useAuth to retrieve current user information.

Explanation: This setup enables smooth integration between frontend, backend, and payment systems.

2. **Component State Management**
   Manages form input (cardholderName) and payment state (isProcessing) to control UI behavior during payment.

Explanation: Prevents users from resubmitting or modifying data while a payment is being processed.

3. **Form Validation**
   Checks that the cardholder name field is not empty before processing.

Explanation: Ensures valid and complete data before communicating with Stripe APIs.

4. **Payment Handling Logic** ○ **Step 1:** Requests a PaymentIntent from a Firebase Cloud Function. ○ **Step 2:** Confirms the payment with Stripe using confirmPayment.
   - **Step 3:** Updates the user's document in Firestore to reflect premium status and records payment history.

Explanation: This structured flow guarantees secure, verifiable payments that immediately update backend data.

5. **User Interface Layout** ○ Header with navigation buttons.
   - Summary card displaying selected plan and amount. ○ Input fields for cardholder name and card details. ○ Payment and cancel buttons for user action.
   - Footer confirming transaction security.

Explanation: A clean layout ensures transparency and usability during payments.

6. **Styling and Design**
   Uses consistent colors and rounded edges for buttons, keeping the interface modern and aligned with the SOS app's theme.

Explanation: Visual design enhances user trust during sensitive actions like payments.

7. **Error Handling and Feedback**
   Displays appropriate alerts for invalid inputs, failed transactions, and successful payments.

Explanation: Ensures clarity and user confidence at each payment step.

14. Conclusion

The SOS app demonstrates a well-integrated mobile solution combining Firebase and Stripe. It provides secure authentication, real-time emergency communication, and a seamless premium subscription process. The modular architecture, robust backend, and user-friendly design ensure scalability and reliability for future expansion.