📱 Historical Places App

Prepared by Africk Teoh , ReactJS Developer, React Native Developer  
Date: 27 August 2025  
Linked Profile : https://www.linkedin.com/in/africkteoh/  
  
A mobile app built with React Native (https://reactnative.dev/) to explore historical places in Malaysia. It connects to a custom Node.js backend API for fetching data.

# ✨ Features

- 🌍 Cross-platform (iOS & Android)  
- 📦 State management with Redux Toolkit / Context API  
- 🧭 Smooth navigation with React Navigation (https://reactnavigation.org/)  
- 📑 Example screens & reusable components  
- 🔗 Node.js API integration for real-time data

# 🚀 Getting Started

## 1. Clone the Repository

git clone https://github.com/mrteoh/HistoricalPlacesApp.git  
cd HistoricalPlacesApp

## 2. Install Dependencies

npm install  
# or  
yarn install

## 3. Run the App

yarn ios  
yarn android  
npm start

## 4. Backend (Node.js API)

cd malaysia-places-api  
node server.js  
  
If successful, you should see:  
Server running at http://localhost:3001

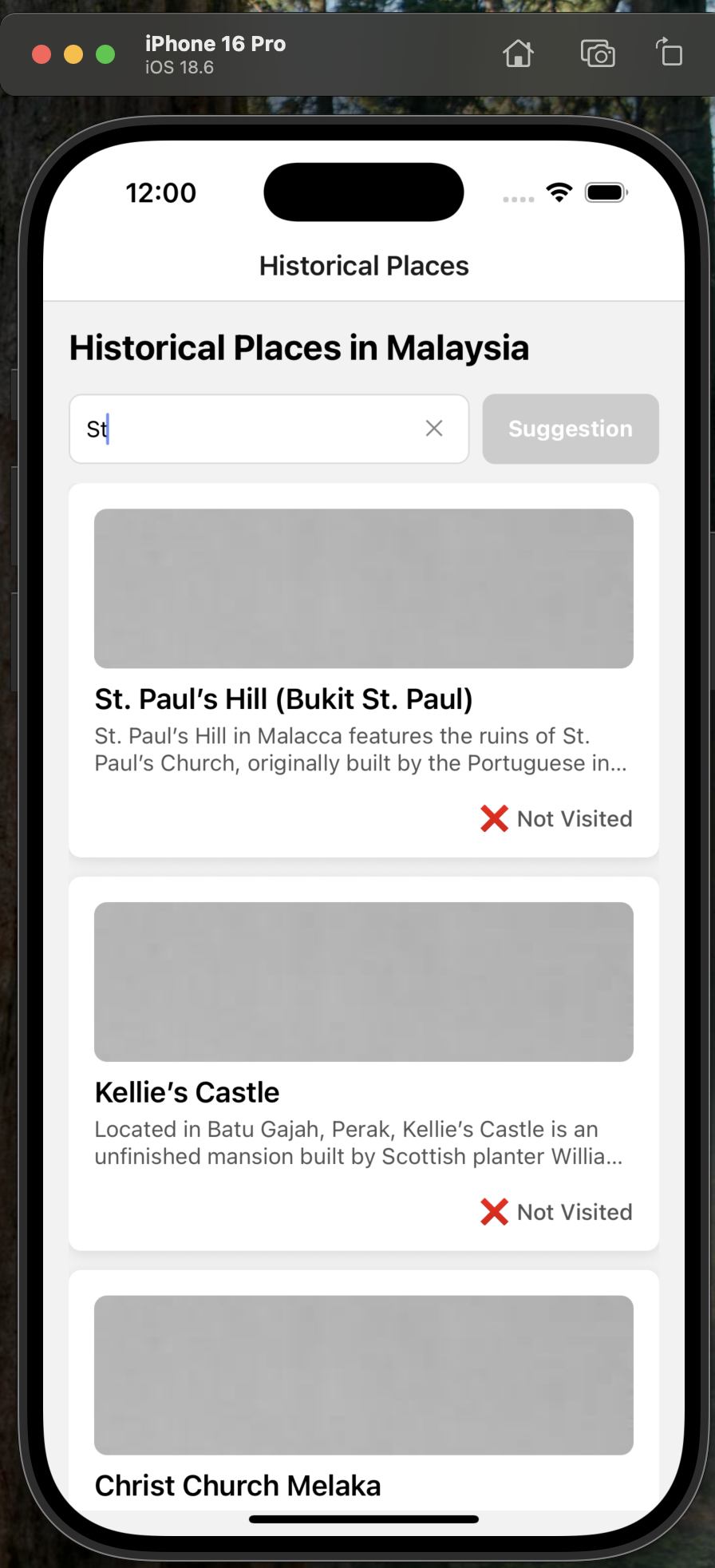
# 🧪 Mobile App Description

1. Introduction  
This document outlines the test plan for the Historical Places App, developed using React or React Native. The app displays a list of historical places, allows users to mark places as visited, includes navigation between screens, and features a fun interactive element.

2. Scope  
The testing will cover:  
∙ Data Management: Fetching and displaying historical places.  
∙ Visited Places: Marking and unmarking places as visited with immediate UI updates.  
∙ Fun Feature: Testing the interactive feature (e.g., random place suggestion).  
∙ UI/UX Consistency: Ensuring a consistent design.  
∙ State Management: Verification of Redux and Redux-Observable Epics.  
∙ Routes and Navigation: Testing screen transitions, route handling, and navigation.  
∙ Cross-Platform Consistency: For React Native, ensuring the app works on both iOS and Android.

3. Test Steps  
3.1 Data Management  
∙ Open the app and ensure that historical places are fetched from the data source.  
∙ Confirm that each place is displayed with its name, image, and description.  
3.2 Visited Places Functionality  
∙ Mark a place as visited from the list screen.  
∙ Verify that the UI updates immediately to reflect the place's visited status.  
∙ Unmark the place as visited.  
∙ Check that the UI updates accordingly to show the place as unvisited.  
∙ Confirm that the Redux store is updated in real-time using React Hooks.  
3.3 Fun Feature  
∙ Interact with the fun feature (e.g., press the random place suggestion button).  
∙ Verify that a random place is suggested.  
∙ Check that the suggested place is valid and appears correctly in the UI.  
3.4 Routes and Navigation  
∙ Navigation Between Screens:  
 o Tap on a historical place in the list to view its details.  
 o Navigate back to the list screen.  
 o Use the back button on the device or app to ensure correct navigation.  
 o Verify that screen transitions are smooth and that the correct data is displayed on each screen.  
∙ Direct Navigation (Deep Linking):  
 o Open the app using a specific route or deep link.  
 o Ensure that the app opens on the correct screen.  
 o Confirm that the navigation stack is handled correctly.  
∙ Navigation Behavior After Marking a Place:  
 o Mark a place as visited from the list screen.  
 o Navigate to the details screen of that place.  
 o Return to the list screen.  
 o Ensure that the visited status of the place is still correctly displayed in the list.

# 📸 Screenshots (iOS)





# 

# 📸 Screenshots (Android)

