**The Sightz App Documentation Requirement** Strategy & Design Approach:

Our plan is to build a responsive application that can address the challenges of student with form of disability and with functional features to guide and navigate around the school environment easily.

Map navigation and also a reliable CRUD database connected to fire base API to handle state management on the application

|  |  |
| --- | --- |
| Target Release | Android 11, iOS 15 |
| Guide | Mobile Application to help disabled student group navigate and access school services easily |
| Documentation By | Adekunle Jimoh |
| Focus | Porting App to iOS |
| Target SDK | Version 33 |
| Database API | Firebase |

USERS REQUIREMENT

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| S/N | Features Title | Features Descriptions | Priority | Notes |
| 1 | Fragment Menu | A customized menu button below to ease navigation from major section | High | * This will enable user to switch within activity seamlessly on the application. * Make the layout of application direct users properly and navigate properly |
| 2 | Login & Registration | A vital part of CRUD which enables the create and reading of data | High | * Allow a proper connection to the firebase backend and generate a user identification |
| 3 | Maps navigation | Users will want to have access to get location and places with a reliable search view | High | * To ease direction and how to move within the area effectively |
| 4 | Update & delete | The later part of the CRUD function to update profile and delete when needed | High | * The update profile allow other attribute relating to text field, and a fresh UI to update also we integrate delete and function to erase data off completely |
| 5 | Read & write data Locally | Users want to have data saved locally and have it saved in the app | Medium | * To enable reading data from the device locally and making use of some of the device features |
| 6 | Firebase APi | Integration of the API as a backend to manage majority of the state management events and validations | Medium | * Allowing device to connect and get database resources virtually |

User interaction and design

|  |  |  |
| --- | --- | --- |
| Login Screen | Places Activity | Profile page |
| A screenshot of a login page  Description automatically generated with medium confidence | A picture containing text, screenshot, software, web page  Description automatically generated | A screenshot of a cell phone  Description automatically generated with low confidence |

Adaptation to other Mobile OS (Porting from Android to iOS)

|  |  |
| --- | --- |
| Adaptation/Changes |  |
| iOS Port | Design and user interface: iOS has its own user interface paradigms and design principles. Adjust the user interface's buttons, menus, and layouts to meet iOS requirements. To offer a seamless user experience, take into account employing native iOS controls and widgets.  Code Migration: Rewriting the application logic and adapting it to the iOS platform is required when transferring an app's code from Android (written in Java or Kotlin) to iOS (written in Swift or Objective-C). This entails rebuilding any platform-specific functionality, data management, networking, and UI-related code.  Examine the libraries and dependencies that are utilised in the Android app. Identify whether iOS has identical libraries or whether different solutions must be used. iOS manages dependencies using CocoaPods or Swift Package Manager.  Data Persistence: You'll need to find an equivalent solution for iOS if the Android app makes use of a local database or storage system. For local data persistence, iOS normally uses Core Data or SQLite.  Test and debug the ported app thoroughly on iOS devices and simulators. Keep an eye out for any potential screen size, performance, memory management, and device-specific behaviour issues. Debugging and troubleshooting can be done using Xcode and iOS development tools. |
|  |  |

Not Doings

* No Day/Night View integrated
* No dual screen orientation view