|  |  |  |
| --- | --- | --- |
| SplashActivity (Shown to all users) | | |
|  | | * SplashActivity trigger SUTD\_TTS Singleton to check on the validity of the saved credentials (if any), and redirect user to the correct activity accordingly   + 1st Time User launching app: **WelcomeActivity**   + User has never logged in before/Login credentials are incorrect: **LoginPageActivity**   + Login Credentials Are correct: **MainActivity** |
| WelcomeActivity (For first time users) | | |
|  | * Upon pressing the “Get Started” button, we will check for BLE permissions and request for it accordingly * For users running Android 10and above, for BLE ranging to run in the background, special permissions (Allow location all the time) is required * However, this permission cannot be requested via the traditional requestPermissions(CONTEXT, ...) * Instead, implicit Intent will need to be fired to bring the user to their Settings app, where they grant BluEntry permissions to access location in the background * startActivity(Intent(android.provider.Settings.ACTION\_LOCATION\_SOURCE\_SETTINGS)) | |
| LoginPageActivity: This activity enables the user to login. | | |
|  | It performs the login via the SUTD\_TTS Singleton Instance. | |
| MainActivity: This activity handles most of the logic of the app, and has 3 fragments: | | |
|  | 1. Home fragment  * Shows bluetooth status * Shows frequently visited location   Bluetooth Status   * The “Bluetooth Status” circle will update when there is an error with the BLE system. * When bluetooth is off, clicking on the “Turn on bluetooth” button brings the user to the bluetooth settings page.   Frequently Visited Location   * Horizontal scroll view to show the frequently visited locations. * Shows the location * Shows the number of times visited   Menu Drawer   * Help page contains the answers to frequently asked questions * About page allow users to find out more about BluEntry * Logout button to log the user out of BluEntry | |
|  | 1. HistoryFragment  * Records of Check In and Check Out (Date, Duration, Place) * RecyclerView with CardView connected to the Firebase backend   Spinner   * To give date filter options   Two tabs   * Period: Categorised according to date. * Place: Categorised according to Place visited.   Help Button   * Provides details of the page | |
|  | 3. DeclarationFragment   * ScrollRefreshView with 3 selections for user to select if they wish to perform a ***Daily Declaration*** or ***Temperature Declaration*** * Selections are integrated into the SUTD\_TTS Singleton, which will return a Boolean on whether the declaration has been done * For declarations which **have been completed**, users will see a *100%* and will **not be able to click on the selection**   DeclarationActivity   * This activity handles WebView interaction with the tts.sutd.edu.sg portal when the user would like to perform a temperature/daily declaration. * An intent will be fired from DeclarationFragment & specifies the type of declaration would like to be performed. * DeclarationActivity is launched, and it will obtain the login cookies from SUTD\_TTS Singleton. * **The login cookies are then injected into the WebView,** providing a seamless (“single-sign-on”) experience * **JavaScript Tricks** *(via ChromeWebClient)*   **Navigation Bar Issue**   * As the tts.sutd.edu.sg web portal has a navigation bar which enables user to logout of the system, this might invalidate the login cookies stored in the SUTD\_TTS Singleton and used throughout the app * To resolve this, upon loading is completed, **a callback is fired to a custom ChromeWebClient instance**, and we **inject a JavaScript** which will **hide the navigation bar element** accordingly   **“Seamless” Submit Button**   * In the first iteration of our app, after user pressed submit button, the DeclarationActivity would still remain as we would not be able to determine if the user had completed the submission * Originally, we experimented with overriding the onPageUrl function in ChromeWebClient, which would be called every time the WebView/WebClient attempts to make a GET/POST request.   + The concept was that: If a POST request was made, it would likely be a form submit, and we can assume that the user did a successful submission   + However, during UAT, we realized that there were multiple scenarios were the TTS portal would make POST requests even though user did not make a successful declaration * The final solution involved overriding onPageAlert function in ChromeWebClient, as TTS portal will embed a JavaScript alert() in its POST response. We filter the JavaScript alert, check the current URL of the WebClient and dismiss the activity accordingly. | |
|  | 4. Reminder   * Daily push notifications will be sent out to remind the students to complete their temperature log and daily declarations   5. Running the app at the background   * We made use of services to keep the app running at the background | |