Welcome To Cheney Mobile App: Developer Documentation

1. IDE's and Project Setup

- Firstly it is important to note that YOU WILL NEED both a Windows computer and a Mac computer. The Windows computer will be used to run Android Studio, this is necessary to be able to test how the app will look and run on Android mobile phones. The Mac is necessary in order to test iPhones, this will be done by running an iPhone emulator through the use of Xcode.

- For developers using Windows Visual Studio Code, Flutter, and Android Studio will be required in order to work on the app. Visual Studio Code will be the IDE used to program the app. Flutter is the framework that is used to program the app, and Android Studio will be used to generate the Android phone emulator for you to test the app. A useful video to get all three of these setup on your device can be found here: https://www.youtube.com/watch?v=VFDbZk2xhO4&list=PLCC34OHNcOtpx9qCZNv-NbIT1Gx3BAOku

- For developers using Mac you will need to get Flutter and Xcode downloaded on your computer. This will be necessary to run the iOS Simulator to test the app on an iPhone. A useful video for getting this set up on your Mac can be found here: https://www.youtube.com/watch?v=KdO9B_CZmzo

- It is important that you also do your research on the Flutter framework and the Dart programming language.

- Flutter is a useful tool that centers itself around the idea of **Widgets**. These Widgets can be considered all the various UI components inside the app. There are essentially two different types of widgets, **Stateless** and **Stateful**. Stateless Widgets are static, meaning once they're loaded they do not change. This is useful for things such as the title bar for each page or maybe a body of text that is guaranteed to stay the same. Stateful Widgets are dynamic, they can have their values updated and changed on the fly while the app is running. Most of the custom Widgets we made during this project can be boiled down to Stateless or Stateful Widgets nested inside of Stateful Widgets.

- The Dart programming language is very similar to Java in terms of syntax. There are some key differences in terms of data structures that should be looked into ahead of time. Here is a useful video series that explains how to program in Dart: https://www.youtube.com/watch?v=HbzUzEg8Aqc&list=PLCC34OHNcOto7WU2QzVn3hnpSOYEdflVf

2. How to obtain the source code?

- Navigate to our GitHub repo, it can be found by clicking this link: https://github.com/CSCD488-Winter2024/senior-project-the-a-team

- From there you can download the repo locally onto your computer through which ever IDE you are currently using (vscode or xcode). Once that is done navigate to whichever branch you wish to work on, or if you are trying this for the first time you can run the program from the main branch.

3. How to build the software?

- To build/run the software first begin by booting up the emulator you wish to run it on. There is an example of how to do this for both Windows and Mac in the videos provided in part 1. After the emulator of your choice is running you can right click on main.dart and click "run without debugging".

4. Layout of the directory structure

- Inside the project you will find three main folders (.vscode, flutter, and pdfs). Inside the "**flutter**" folder you will see another folder called "**wtc**", this is where all the code for the project is stored. There are many other folders and files stored inside wtc folder which may seem intimidating at first, over the next few paragraphs we hope to explain the importance of each one and where certain files can be found.

- The **android** folder is used to store all the Android specific files relating to formatting and icons for the app are stored there for Android devices. Inside this folder you will see a file called debug.keystore. This file is necessary for using the Google SSO feature to sign in to the app.

- Inside the **functions** folder you will find a file called **index.js**. This file stored the code for the Firebase functions. The functions are necessary for the push notification system and for allowing admins to delete the accounts of other users.

- The **images** folder is used to store various images that are used across the app

- The **ios** folder, similarly to the android folder, stores the necessary device information for iPhones as well as the icons for the iOS version of the app.

- The **lib** folder is where the main components of the app are stored. There are many files and folders here so we will only be covering what can be found in each folder as well as the most important files. But the folders inside of lib, in addition to the names of the files inside those folders should give you a good idea of their purpose.

    - The **accountPages** folder contains all of the various pages relating to the accounts page. Inside here you will find all the pages on the app where users can edit their information.

    - The **authentication** folder contains all the files and pages relating to user account registration, logging into the app, users deleting their account etc.

    - The **components** folder contains various smaller widgets that are being reused across the app. Widgets such as buttons and text fields.

    - The **intro_screens** folder holds the pages that introduce the user to the app the very first time they log in.

    - The **pages** folder holds the majority of all the different pages in the app. Everything from the main five pages on the bottom nav bar of the app to pages that handle creating and editing posts. Many of the pages that are in the hamburger menu are stored there as well.

    - The **services** folder contains the file that handles the Single Sign On functions for both Google and Apple.

    - The **User** folder contains a file called **global_user_info.dart**. This file is used to hold the user's information and make it accessible across the app. This helps keep the number of reads to the Firestore database low.

    - The **widgets** folder is where the majority of the widgets for the app are stored. The folders **event_widgets**, **job_post**, **post_widgets**, and **volunteer_post** store all the widgets used to make up the different types of posts for the app. The **user_widgets** folder contains the widgets that are used for the search users page. There are many more widgets stored inside the widgets folder that are stand alone but their names give a clue as to their purpose and where they’re being used relative to the pages.

    - The **edit_pages** folder holds all the files that control the pages for allowing users to edit posts.

- The **event_pages** folder holds the files for the calendar and the attending event buttons

- The **map_pages** holds all the files for the map page and the individual pages for businesses displayed on the map.

- The **over_flow** folder holds the field for the “about us” page as well as the account review and the admin review pages.

- The **creation_pages** folder holds all the files that allow users to create different kinds of posts.

- The **infrastructure** folder holds all the files for the navigation buttons and the top bar.

- The **lists** folder holds all the files for the different list widgets that display the posts on the app.

- The **app.dart** file contains the code for the routing for all pages on the app. Thai includes the pages tied to the buttons on the bottom nav bar as well as all the pages in the hamburger menu.

- The **firebase_options.dart** file contains code relating to which device specific information should be loaded based on the device running the app. This means iOS, Android, Windows, MacOS, etc.

- The **main.dart** file is the entry way for the app. The main function for the app is stored there. Inside the main function all the functions that initialize the app and connect it to Firebase are performed before the app boots up.

- Towards the bottom of the wtc folder you will find a file named **pubspec.yaml**. This folder contains all the dependencies for the app. If you find a cool widget online that you would like to implement this is where you would add the dependency for that widget.

5. How to test?

- For this project we did not use a testing framework but instead opted to use a “behavior driven testing” approach. To do this we would divide up a task into the smallest parts that could be programmed. From there we would implement each piece one at a time. Once a

piece had been successfully implemented the developer would then test that function to make sure that it worked as intended and that no unexpected behaviors could be found. After that a pull request would be made on GitHub with detailed documentation of how to test the feature and what to expect. At that point another developer on the team would repeat the testing progress and approve the pull request if the testing was successful, otherwise requesting changes if unexpected or faulty behavior occurred.

- This testing process was effective but certainly not the most efficient. This team recommends that in the future if another team decides to re-open this project that they research a testing framework to use with Flutter.

6. Build Release / Project Deployment

- We are currently working on deploying the project on both the Google Play store as well as the Apple App store.

- For deploying the app on the Google Play store here is a useful video that will guide you through the process: https://www.youtube.com/watch?v=Jk4X3EDXi7s&t=202s

- For deploying the app on the Apple App store here is a useful video that will guide you through the process: https://www.youtube.com/watch?v=0zgDF81ZLrQ&t=651s