1. Setting Up Flutter on Windows and MacOS

To set up Flutter on both Windows and Mac OS, follow these step-by-step instructions:

1.1. Setting Up Flutter on Windows

- **System Requirements:**
  - Windows 10 or later (64-bit)
  - Disk Space: 1.64 GB (does not include disk space for IDE/tools).
  - Tools: Git for Windows.
- **Install Flutter SDK**
  - Download the latest stable release of the Flutter SDK from the Flutter website.
  - Extract the zip file and place the contained `flutter` in the desired installation location for the Flutter SDK (e.g., `C:\src\flutter`).
- **Update Your Path:**
  - From the Start search bar, type `env` and select "Edit the system environment variables".
  - In the System Properties window, click "Environment Variables".
  - Under User variables, check if there is an entry called `Path`:
  - If the entry exists, append the full path to `flutter\bin` using `;` as a separator from existing values.
  - If the entry does not exist, create a new user variable named `Path` with the full path to `flutter\bin` as its value.
- **Run Flutter Doctor:**
  - Open a new command prompt and run `flutter doctor`.
  - This command checks your environment and displays a report of the status of your Flutter installation.
- **Install Android Studio:**
  - Download and install Android Studio.

- Start Android Studio, and go through the ‘Android Studio Setup Wizard’. This installs the latest Android SDK, Android SDK Command-line Tools, and Android SDK Build-Tools, which are required by Flutter when developing for Android.
- Open Android Studio > Preferences > Plugins, and install the Flutter and Dart plugins.

■ **Set Up the Android Emulator:**

- In Android Studio, open AVD Manager by selecting Tools > AVD Manager.
- Follow the prompts to create a new Android Virtual Device (AVD).

■ **Additional Windows Setup:**

- If you are installing Flutter for the first time, run the `flutter doctor` command again to ensure everything is configured properly.

1.2. Setting Up Flutter on MacOS

■ **System Requirements:**

- macOS 10.14 (Mojave) or later
- Disk Space: 2.8 GB (does not include disk space for IDE/tools).

■ **Install Flutter SDK:**

- Download the latest stable release of the Flutter SDK from the Flutter website.
- Extract the zip file and place the contained `flutter` in the desired installation location for the Flutter SDK (e.g., `$HOME/flutter`).

■ **Update Your Path:**

- Open (or create) the `~/.zshrc` file and add the following line:

```
export PATH="$PATH:`pwd`/flutter/bin"
```

- Run `source ~/.zshrc` to refresh the current terminal window.

■ **Run Flutter Doctor:**

- Open a terminal and run `flutter doctor`.

- This command checks your environment and displays a report of the status of your Flutter installation.

- **Install Xcode:**
  - Download and install Xcode from the Mac App Store.
  - Open Xcode and go through the initial setup.
- **Install CocoaPods:**
  - Run the following command to install CocoaPods, used for managing iOS project dependencies:

```
sudo gem install cocoapods
```

- **Install Android Studio:**
  - Download and install Android Studio.
  - Start Android Studio, and go through the 'Android Studio Setup Wizard'. This installs the latest Android SDK, Android SDK Command-line Tools, and Android SDK Build-Tools, which are required by Flutter when developing for Android.
  - Open Android Studio > Preferences > Plugins, and install the Flutter and Dart plugins.
- **Set Up the iOS Simulator:**
  - Open Xcode and navigate to Xcode > Preferences > Components.
  - Download a simulator for the latest version of iOS.
- **Set Up the Android Emulator:**
  - In Android Studio, open AVD Manager by selecting Tools > AVD Manager.
  - Follow the prompts to create a new Android Virtual Device (AVD).
- **Verify the Setup:**
  - Run `flutter doctor` again to ensure everything is configured correctly.

2. Changing Application Name in Flutter for Android, iOS, and Web

2.1. Prerequisites

- Flutter SDK installed
- Flutter project created
- Basic knowledge of Flutter and project structure

2.2. Android

- **Update `AndroidManifest.xml`**
  - Open `android/app/src/main/AndroidManifest.xml`.
  - Locate the `<application>` tag.
  - Add or modify the `android:label` attribute with your desired application name.

```
<application
    android:name="io.flutter.app.FlutterApplication"
    android:label="Your New App Name"
    android:icon="@mipmap/ic_launcher">
     …
</application>
```

- **Update `strings.xml`**
  - Open `android/app/src/main/res/values/strings.xml`.
  - Locate the `<string name="app_name">` tag.
  - Change the content to your desired application name.

```
<string name="app_name">Your New App Name</string>
```

2.3. IOS

- **Update `Info.plist`**
  - Open `ios/Runner/Info.plist`.

- Locate the `<key>CFBundleName</key>` entry.
- Change the string value to your desired application name.

```
<key>CFBundleName</key>
<string>Your New App Name</string>
```

- **Update `Display Name`**
  - Open `ios/Runner.xcodeproj` or `ios/Runner.xcworkspace` in Xcode.
  - In the project navigator, select the `Runner` project.
  - Under the `General` tab, locate the `Display Name` field.
  - Change the display name to your desired application name.

2.4. Web

- Open `web/index.html`.
- Locate the `<title>` tag inside the `<head>` section.
- Change the content of the `<title>` tag to your desired application name.

```
<head>
    ...
    <title>Your New App Name</title>
    ...
</head>
```

3. Changing Package Name in Flutter

Changing the package name of a Flutter project involves updating multiple files and configurations for both Android and iOS platforms. Below are the detailed steps to change the package name.

3.1. Android

- **Update `applicationId` in `build.gradle`**
  - Open `android/app/build.gradle`.
  - Locate the `**applicationId**` field inside the `defaultConfig` section.
  - Change the `**applicationId**` to your new package name.

```
defaultConfig {
    applicationId "com.new.package.name"
    ...
 }
```

- **Rename directories in `src/main/java`**
  - Navigate to `android/app/src/main/java`.
  - Inside this directory, you will find directories matching your old package name (e.g., `com/old/package/name`).
  - Rename these directories to match your new package name structure (e.g., `com/new/package/name`).
- **Update package name in `AndroidManifest.xml`**
  - Open `android/app/src/main/AndroidManifest.xml`.
  - Update the `**package**` attribute in the `**<manifest>**` tag to your new package name.

```
<manifest
xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.new.package.name">
    ...
 </manifest>
```

- **Update package names in other `AndroidManifest.xml` files.**
  - Open `android/app/src/debug/AndroidManifest.xml` and `android/app/src/profile/AndroidManifest.xml`.
  - Update the `package` attribute in the `<manifest>` tag to your new package name in these files as well.

3.2. IOS

- **Update `PRODUCT_BUNDLE_IDENTIFIER` in `project.pbxproj`**
  - Open `ios/Runner.xcodeproj` or `ios/Runner.xcworkspace` in Xcode.
  - In the project navigator, select the `Runner` project.
  - Under the `Build Settings` tab, locate the `PRODUCT_BUNDLE_IDENTIFIER` field.
  - Change the bundle identifier to your new package name (e.g., `com.new.package.name`).
- **Update `Info.plist`**
  - Open `ios/Runner/Info.plist`.
  - Locate the `<key>CFBundleIdentifier</key>` entry.
  - Change the string value to your new package name.

```
<key>CFBundleIdentifier</key>
<string>com.new.package.name</string>
```

4. Changing Application Logo in Flutter

4.1. Android

- **Replace App Icons**
  - Prepare your custom app icons in the required sizes (e.g., `ic_launcher.png`) for different densities.
  - Replace the default icons located in the `android/app/src/main/res` directory with your custom icons.
  - Place your icons in the corresponding `mipmap` directories (e.g., `mipmap-mdpi`, `mipmap-hdpi`, `mipmap-xhdpi`, `mipmap-xxhdpi`, `mipmap-xxxhdpi`).
- **Update `AndroidManifest.xml`**
  - Open `android/app/src/main/AndroidManifest.xml`.
  - Locate the `<application>` tag.

- Update the `android:icon` attribute to point to your custom launcher icon.

```
<application
    android:name="io.flutter.app.FlutterApplication"
    android:icon="@mipmap/ic_launcher"
    ...
</application>
```

4.2. IOS

- **Replace App Icons**
  - Prepare your custom app icons in the required sizes (e.g., `AppIcon.appiconset`) for different devices.
  - Replace the default icons in the `ios/Runner/Assets.xcassets/AppIcon.appiconset` directory with your custom icons.
- **Update `Info.plist`**
  - Open `ios/Runner/Info.plist`.
  - Locate the `CFBundleIconName` key.
  - Update the string value to match the name of your custom icon set.

```
<key>CFBundleIconName</key>
<string>AppIcon</string>
```

4.3. Flutter Web

For Flutter web, you typically use the favicon to represent your application. To change the favicon:

- **Replace Favicon**

- Prepare your custom favicon in the required sizes (e.g., `favicon.ico`) for different devices.
- Replace the default favicon file located in the `web` directory with your custom favicon.

5. Setting up Firebase in a Flutter app using the command-line interface (CLI):

5.1. Install Firebase CLI Tools

Ensure you have the Firebase CLI tools installed globally on your system. If not, you can install them using npm:

```
npm install -g firebase-tools
```

5.2. Authenticate Firebase CLI

Run the following command to authenticate with Firebase:

```
firebase login
```

Follow the instructions in your terminal to complete the authentication process.

5.3. Create a Firebase Project

If you haven't already created a Firebase project, you can create one using the following command:

```
firebase projects:create
```

5.4. Initialize Firebase in Your Flutter App

Navigate to your Flutter project directory in the terminal and run the following command to initialize Firebase:

```
firebase init
```

Follow the prompts to select the Firebase services you want to use (e.g., Firestore, Authentication, Hosting, etc.) and choose your Firebase project from the list of existing projects.

5.5. Use Firebase Services

Now that Firebase is set up in your Flutter app, you can start using Firebase services by importing the necessary Firebase packages and following the documentation for each service.

6. Setting up Firebase in a Flutter app manually

6.1. Create a Firebase Project

- Go to the Firebase Console.
- Click on "Add project" and follow the on-screen instructions to create a new Firebase project.
- Once the project is created, you'll be redirected to the project dashboard.

6.2. Add Your App to the Firebase Project

- **Android**
  - Click on the Android icon to add an Android app to your Firebase project.
  - Follow the on-screen instructions and provide your Android package name (e.g., com.example.myapp).

- Download the `google-services.json` file and place it inside the `android/app` directory of your Flutter project.

- **IOS**
  - Click on the iOS icon to add an iOS app to your Firebase project.
  - Follow the on-screen instructions and provide your iOS bundle ID (e.g., com.example.myapp).
  - Download the `GoogleService-Info.plist` file and add it to your Xcode project (`ios/Runner` directory) as instructed.

6.3. Use Firebase Services

Now that Firebase is set up in your Flutter app, you can use Firebase services such as authentication, firestore, storage, etc., by importing the corresponding Firebase packages and following their documentation to implement the desired functionalities.

7. Enabling authentication Firebase console

7.1. Phone OTP Authentication

- **Firebase Console:**
  - Go to the Firebase Console.
  - Select your Firebase project.
  - Navigate to the "Authentication" tab from the left sidebar.
- **Enable Phone Authentication:**
  - Under the "Sign-in method" tab, enable "Phone" as a sign-in provider.
  - Under the "Sign-in method" tab, enable "Phone" as a sign-in provider.

7.2. Google Sign-In Authentication

- **Firebase Console:**
  - Go to the Firebase Console.
  - Select your Firebase project.

- Navigate to the "Authentication" tab from the left sidebar.

- **Enable Google Sign-In:**
  - Under the "Sign-in method" tab, enable "Google" as a sign-in provider.
  - Configure OAuth consent screen details if prompted.
  - Save your changes.

7.3. Apple Sign-In Authentication

- **Firebase Console:**
  - Go to the Firebase Console.
  - Select your Firebase project.
  - Navigate to the "Authentication" tab from the left sidebar.
- **Enable Apple Sign-In:**
  - Under the "Sign-in method" tab, enable "Apple" as a sign-in provider.
  - Configure the necessary details, including the team ID and service IDs obtained from your Apple Developer account.
  - Save your changes.

8. Firestore Database Collection Import Export

8.1. Install Node.js and npm:

- Download and install Node.js.
- Follow the installation instructions for your operating system.

8.2. Extract Zip Folder :

- Extract the contents of the `database.zip` file to a folder on your computer.

8.3. Open Command Prompt:

- Navigate to the extracted zip folder.
- Open a command prompt or terminal in that folder.

8.4. Generate Firebase Service Account Key:

- In your Firebase project settings, go to "Service accounts".
- Select "Node.js" as the platform.
- Click on "Generate new private key".

- Wait for the key to be generated, and it will auto-download.
- Replace the existing `config.json` file in your project folder with this new downloaded file.

8.5. Configure config.json:

- Go to your Firebase project settings and locate the app configuration JSON.
- Copy the contents of the JSON file.
- Create a new file named `config.json` in your project folder.
- Paste the copied JSON into this file.

8.6. Run Import Commands:

```
npx -p node-firestore-import-export firestore-import -a config.json -b database.json
```

8.7. Run Export Commands:

```
npx -p node-firestore-import-export firestore-export -a config.json -b database.json
```

9. Firestore Database Indexing import .

9.1. Go to Your Firebase Project Directory:

- Navigate to your Firebase project directory (in this case, the `admin` project) using the `cd` command in your terminal or command prompt:

```
cd path/to/admin
```

- Replace `path/to/admin` with the actual path to your Firebase project directory.

9.2. Deploy Firestore Indexes:

- Once you're in your Firebase project directory, you can deploy Firestore indexes using the following command:

```
firebase deploy --only firestore:indexes
```

Notes:

- Make sure you're in the correct Firebase project directory before running the deployment command.
- Ensure that you have appropriate permissions to deploy Firestore indexes in your Firebase project.
- After deploying indexes, Firestore queries will perform better as they'll be optimized based on the defined indexes.

10. To create a Google Maps API key and integrate it into a Flutter project.

10.1. Create a Google Maps API Key

- Go to the Google Cloud Console.
- Create a new project or select an existing project.
- Enable the "Maps SDK for Android", "Maps SDK for iOS", and "Maps JavaScript API" APIs for your project.
- Navigate to the "APIs & Services" > "Credentials" page.
- Click on "Create Credentials" and select "API key".
- Copy the generated API key.

10.2. Add API Key to Flutter Project

- **For Android:**
  - Open your Flutter project in Android Studio or your preferred code editor.

- Navigate to the `android/app/src/main/AndroidManifest.xml` file.
- Add the following meta-data element within the `<application>` element, replacing `"YOUR_API_KEY"` with your actual API key:

```
<application ...>
    <meta-data
        android:name="com.google.android.geo.API_KEY"
        android:value="YOUR_API_KEY"/>
    ...
</application>
```

- **For iOS (in AppDelegate using Swift):**
  - Open your Flutter project in Xcode.
  - Navigate to the `ios/Runner/AppDelegate.swift` file.
  - Inside the `application(_:didFinishLaunchingWithOptions:)` method, add the following code, replacing `"YOUR_API_KEY"` with your actual API key:

```
GMSServices.provideAPIKey("YOUR_API_KEY")
```

- **For Web:**
  - Open your Flutter web project.
  - Navigate to the `web/index.html` file.
  - Add the following script tag inside the `<head>` element, replacing `"YOUR_API_KEY"` with your actual API key:

```
<head>
    ...
```

```
<script
src="https://maps.googleapis.com/maps/api/js?key=YOUR_API_KEY
&libraries=places"></script>
</head>
```

11. To generate SHA-1 and SHA-256 keys and enter them into the Firebase Console, follow these steps:

11.1. Generate SHA-1 and SHA-256 Keys

- **For Android:**
  - Open a terminal.
  - Navigate to the Java `bin` directory. This is typically located at `C:\Program Files\Java\jdk1.x.x_x\bin`.
  - Run the following command to generate the SHA-1 key:

```
keytool -list -v -keystore "%USERPROFILE%\.android\debug.keystore" -alias androiddebugkey -storepass android -keypass android
```

  - Run the following command to generate the SHA-256 key:

```
keytool -list -v -keystore "%USERPROFILE%\.android\debug.keystore" -alias androiddebugkey -storepass android -keypass android -v
```

- **For macOS:**
  - Open Terminal.
  - Run the following command to generate the SHA-1 key:

```
keytool -list -v -keystore ~/.android/debug.keystore
-alias androiddebugkey -storepass android -keypass
android
```

- Run the following command to generate the SHA-256 key:

```
keytool -list -v -keystore ~/.android/debug.keystore
-alias androiddebugkey -storepass android -keypass
android -v
```

11.2. Run the following command to generate the SHA-256 key:

- Copy the SHA-1 and SHA-256 keys generated in the previous steps.
- Go to the Firebase Console.
- Select your Firebase project.
- Navigate to the "Project settings" > "General" tab.
- Scroll down to the "Your apps" section and select the platform (Android or iOS) for which you want to add the keys.
- Follow the instructions to add the SHA-1 and SHA-256 keys.
- Save the changes.

Note:

- If you're generating keys for a release build, you need to use the keystore file associated with your app's signing certificate.
- For release builds, you should always use a custom keystore and not the default debug keystore.
- Make sure to properly secure and manage your keystore files and keys.

12. To run a Flutter application

12.1. Using Command-Line Interface (CLI):

- **Navigate to Your Project Directory:**

  Open a terminal or command prompt and navigate to your Flutter project directory using the `cd` command.

- **Run the Application:**

  Use the `flutter run` command to run your Flutter application:

```
flutter run
```

  This command will compile your Flutter app and launch it on a connected device or emulator.

12.2. Using Integrated Development Environment (IDE):

- **Android Studio:**
  - **Open Your Project:**

    Open Android Studio and open your Flutter project.
- **Run the Application:**
  - Click on the green play button ("Run") in the toolbar.
  - Select your target device or emulator and click "OK" to run the application.
- **Visual Studio Code:**
  - **Open Your Project:**

    Open Android Studio and open your Flutter project.
  - **Run the Application:**
    - Press `Ctrl + Shift + P` (or `Cmd + Shift + P` on macOS) to open the command palette.
    - Type "Flutter: Run" and select the "Flutter: Run" command from the dropdown.
    - Choose your target device or emulator to run the application.