Programmer Guide

Memory Enhancement Application
Milestone 3 – Team Amazing
Software Engineering Project
SWEN 670
July 13, 2021

Version 1.1

Presented by:

Shawn Kelly
Christian Ahmed
Mitch Olshansky
Chauntika Broggin
Ayodeji Famudehin
Momodou Drammeh
Nicholas Ballo

Revision History

DATE	VERSION	DESCRIPTION	AUTHOR
07/19/2021	1.0	Initial release	Christian Ahmed
08/06/2021	1.1	Update sections 1, 4, 5, 7, & 8 to ease the user to the next section	Shawn Kelly

1. Table of Contents

1. Introduction

The section below will get the user familiar with the application. A high level understand of the document and the project's purpose will be explained.

1.1. Purpose

The Memory Enhancer Programmer's Guide details the decisions, approaches and tools supporting the development and maintenance of the Memory Enhancer mobile application. It documents the most important aspects of developing the Memory Enhancer application.

For instructions on installing the mobile application and maintaining the user environment, refer to the ***Deployment and Operations Guide*** document.

For more detailed information on the users' requirements of the mobile application, refer to the SRS document: ***Memory Enhancer Software Requirements Specification***.

For detail information on the adopted design for the mobile application, refer to the ***Memory Enhancer Design*** document.

1.2. Intended Audience

The programmer's guide document is intended for the developers on the project team who will develop and maintain the Memory Enhancer application.

1.3. Technical Project Stakeholders

The successful development and implementation of the Memory enhancer mobile application depend on the following technical stakeholders:

- Requirements, Strategies and Direction – *Dr. Mir Assadullah, Roy Gordon, Malcolm Freeney*
- Technical architects – *Team Amazing architects and developer leads*
- Developers – *Team Amazing developers and developer leads*
- Testers – *Team Amazing testers and test leads*
- DevSecOps team

1.4. How to use this guide

To facilitate the user's use of this document, the sections are provided in a flow that facilitates onboarding as such:

- The ***Setting up the development environment*** section provides instructions on how to acquire and set up the various tools making up the integrated development environment (IDE).

1.5. Definitions, acronyms, and abbreviations

IDE	Integrated Development Environment

SDK	Software Development Kit
SRS	Software Requirements Specification
STT	Speech-to-Text
TTS	Text-to-Speech
Transcription	Voice recognition and transcribed to readable text
Synthesis	Conversion of readable text to audio sound
Speaker Diarization	Identification of all the speakers in a audio during the transcriptions
MVVM	Model–View–ViewModel design pattern
URL	Uniform Resource Locator
AES	Advanced Encryption Standard

1.6. References

IBM. (n.d.-a). *IBM Watson - Speech to Text*. Ibm.Com. Retrieved July 19, 2021, from https://www.ibm.com/cloud/watson-speech-to-text

IBM. (n.d.-b). *IBM Watson - Text to Speech*. Ibm.Com. Retrieved July 19, 2021, from https://www.ibm.com/cloud/watson-text-to-speech

Soni, D. (2019, July 31). *What is a singleton? - Better Programming*. Medium. https://betterprogramming.pub/what-is-a-singleton-2dc38ca08e92

Top packages. (n.d.). Dart packages. Retrieved July 19, 2021, from https://pub.dev/packages

2. Why Memory Enhancer Application is Needed

In the United States of America, there is a small, but growing population of individuals experiencing short-term memory loss. This short-term memory loss may occur due to advancing age, a symptom of larger issues such as dementia. Life expectancy has increased significantly in the last 50 years. As such, people are living longer and require a longer, greater quality of life. Team Amazing's solution to this global issue, is to create the Memory Enhancer phone application, which acts like a cross between Apple's Siri and a friendly update reminder to serve the user as a personal assistant. The voice recognition technology utilized with this phone application will aid people with disabilities that impair their short-term memory. The general approach for developing such a product will be focused around this question: "What can we do

to help people?"

3. Setting up the Development Environment

The instructions below work for the Windows and MacOS operating systems with slight differences:

- To execute command scripts, Windows uses a ***Command Windows*** while MacOS uses the ***Terminal*** application.
- The iOS emulator is automatically installed with XCode on MacOS, but it is not readily available in Windows environment. In this case the iOS testing of the application could be done with a real iOS device or through the use of a virtual machine (VM) – see instructions in the *Application Testing* section.

3.1. Android Studio

Installation

- Download the latest version of Android Studio at https://developer.android.com/studio.
- For installation on **Windows**, follow the Android Studio Windows Installation Instructions.
- For installation on **MacOS**, follow the Android Studio MacOS Installation Instructions.

3.2. Flutter and Dart

Installing Flutter will also install the Dart since it includes the Dart SDK.

Installation

- For **Windows** installation of Flutter and Dart, follow the Flutter Windows Installation Instructions.
- For **MacOS** installation of Flutter and Dart, follow the Flutter MacOS Installation Instructions.

Flutter integration to Android Studio

Now that Android Studio and Flutter have been installed, Flutter should be integrated to Android Studio.

- Open the Command Window (Windows) or the Terminal application (MacOS)
- Run the command `flutter doctor` to verify that Flutter has located the installed Android Studio.
- If Flutter cannot locate the Android application (see example in *Image 1*), run the following command `flutter config --android-studio-dir <your android studio installation path>` to set the directory where your Android Studio is installed.

```
[-] Android toolchain - develop for Android devices
    • Android SDK at /Users/obiwan/Library/Android/sdk
    ✗ Android SDK is missing command line tools; download from https://goo.gl/XxQghQ
    • Try re-installing or updating your Android SDK,
      visit https://flutter.dev/setup/#android-setup for detailed instructions.
```

Image 1 *– Result of running 'flutter doctor'*

- Run the command `flutter doctor` again until everything is green (see *Image 2*). For other issues, follow the returned instructions to address each issue.

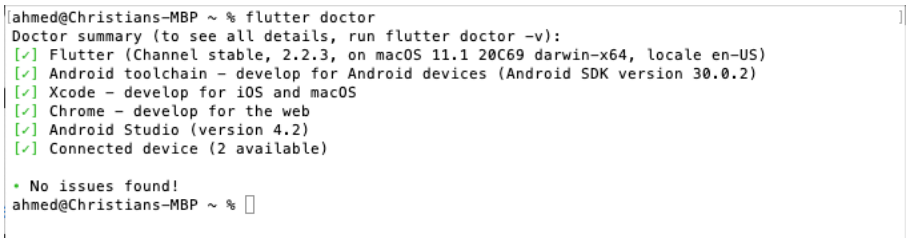

Image 2 *– Successful running of 'flutter doctor'*

Agree to Android Licenses

Before using Flutter, the user must agree to the various licenses of the Android SDK.

- Open the Command Window (Windows) or the Terminal application (MacOS)
- Run the command `flutter doctor –android-licenses` to review and agree to all the licenses.

3.3. Setting up Android emulator

To use an Android emulator, one has to be configured through Android Studio. The following instruct on the configuration of an Android emulator:

- For **Windows** installation of an Android emulator, follow the Android Emulator - Windows Installation Instructions.
- For **MacOS** installation of an Android emulator, follow the Android Emulator - MacOS Installation Instructions.

3.4. Setting up an Android device

Testing could also be done and should be done on a real device at some point. The following instruct on the configuration of an Android Studio device (smartphone):

- For **Windows** installation of an Android device, follow the Android Device - Windows Installation Instructions.
- For **MacOS** installation of an Android device, follow the Android Device - MacOS Installation Instructions.

3.5. Setting up an iOS Simulator (MacOS only)

iOS simulators come with the XCode IDE, which can be installed on MacOS only.

Installing XCode IDE

To install the XCode IDE on a MacOS, follow the XCode Installation Instructions.

Setting up the iOS simulator

To setup the iOS simulator once XCode is install, follow the iOS Simulator Setup Instructions.

3.6. Test the development environment setup

To test that your IDE was successfully installed, simply write a simple Flutter application. Tehre are couple options:

- Creating a Flutter project in Android Studio should create a simple sample project ready to be run on emulator.
- Follow these instructions to Write your first Flutter application.

3.7. GitHub Desktop

The application code repository is stored on GitHub. If one is not familiar with GitHub command, it is much simpler to use the GitHub desktop.

Follow the instructions to Install GitHub Desktop.

3.8. Troubleshooting

4. Application Development Code

The section below will go into detail of the process for developers to add they efforts to the code. The purpose of these efforts include updating the source code with new features, bug fixes, and usability improvement.

4.1. Code Repository

The Memory Enhancer code repository is located in GitHub; the repository is called **summer2021.amazing** (*https://github.com/umgc/summer2021.amazing*).

4.2. Cloning the Code Repository

To access the code, clone the summer2021.amazing repository with one of these options:

- Clone with a Git command using *https://github.com/umgc/summer2021.amazing*
- Open *https://github.com/umgc/summer2021.amazing* in GitHub Desktop

GitHub Desktop instructions

- Open the GitHub Desktop application.
- Login using your GitHub credentials created for the project.
- Once logged in, you will be asked to select a repository.
- Select the summer2021.amazing repository and provide a folder location where you would like your repository to be on your machine (*e.g. "C:\user\documents\GitHub\summer2021.amazing").*
- Select the **repository drop-down > Add > Clone Repository**.

- On the Clone a Repository pop-up (figure 1), select the **GitHub.com** tab, scroll down to locate the summer2021.amazing repository.
- Select the summer2021.amazing repository.
- Provide a local path (folder location) for the repository clone.
- Click the **Clone** button (see *Image 3*).

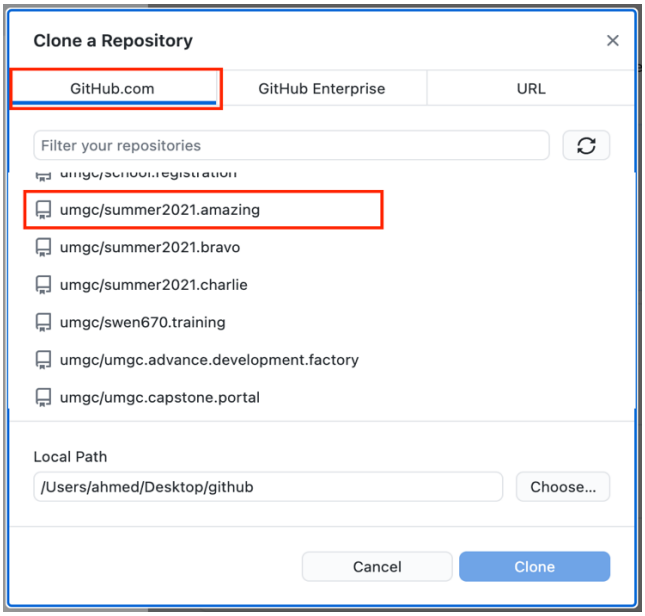

Image 3 – *Cloning code repository*

4.3. Code Branches

The team uses two code branches for development: ***Main*** and ***Developer***.

Main branch

The Main branch hold the stable code of the application. Distribution and final builds are made from this branch. Code is not moved in this branch until it has been successfully tested and integrated in the Developer branch. **Warning:** NO development should occur in this branch.

Developer branch

The Developer branch is the one all team developers will derive their personal developer branch from. Feature branch, created to develop a feature parallelly, will also be derived from the Developer branch.

Branch naming convention

To remain uniform in how our branches are created, the branch naming convention should follow the pattern `<branch-name>_<developer-name>` or `<branch-name>_<feature-name>`. For example, John DOE may create his developer branch as `developer_john`.

Create your personal developer branch

- Open the GitHub Desktop application.

- Login using your GitHub credentials created for the project.
- Clone summer2021.amazing repository if you have not done so yet (*see document **Clone Repository***).
- Select the repository summer2021.amazing (*Image 4 – red box*).
- Select the "developer" branch (*Image 4 – green box*).

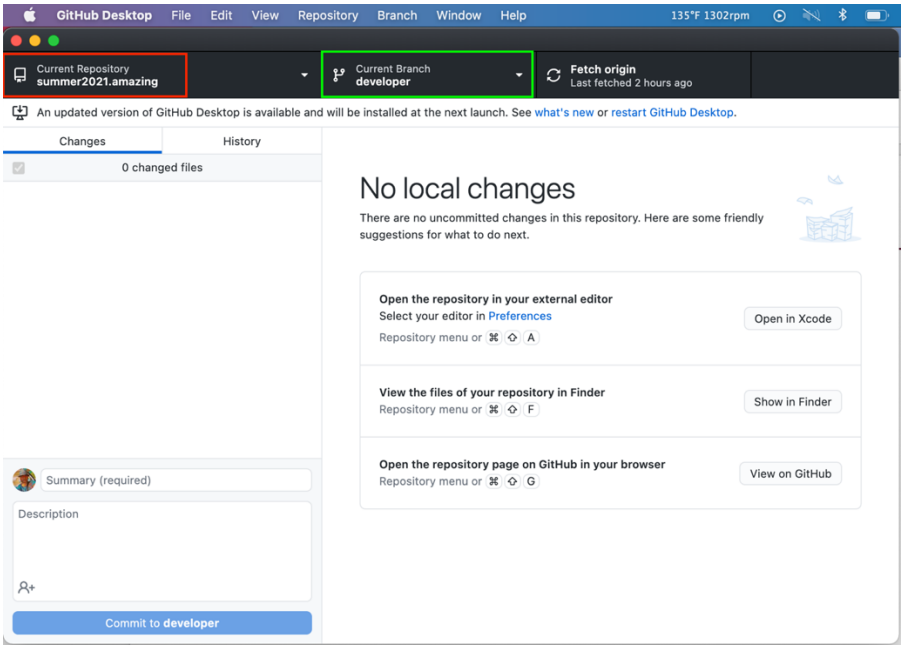

Image 4

- Select the menu ***Branch > New Branch***.

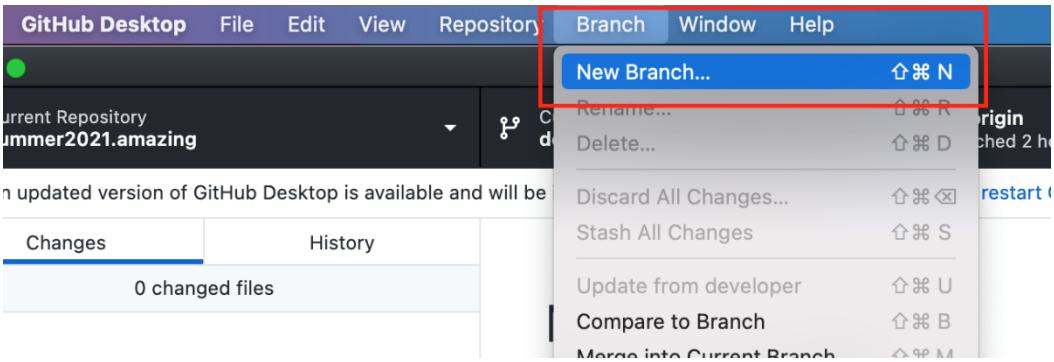

Image 5

- On the *Create a Branch* pop-up, ensure that the ***developer*** branch is the default (your new branch should be mapped to ***developer***); provide a name for you branch (*i.e.* **`developer_<your name>`**) and click the *Create Branch* button.

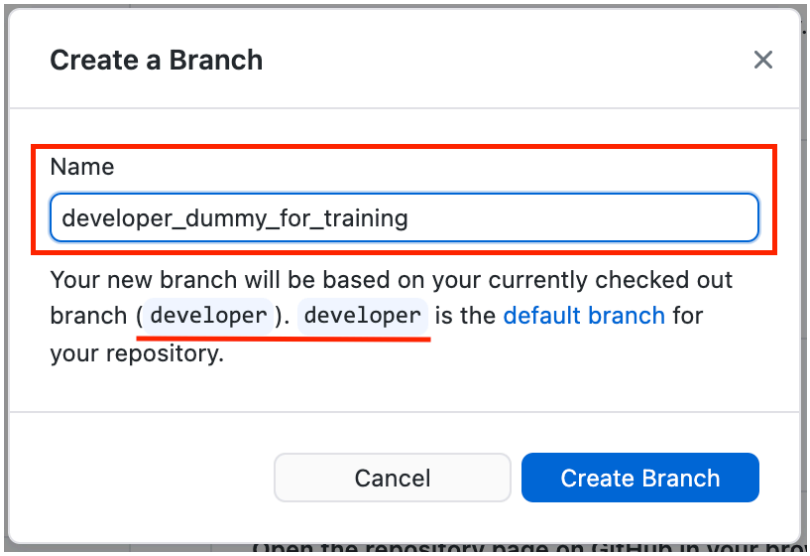

Image 6

- Your new branch is now created locally on your machine, but not on the GitHub server. You will need to publish it. Simply click the *Publish branch* button (*Image 7*).

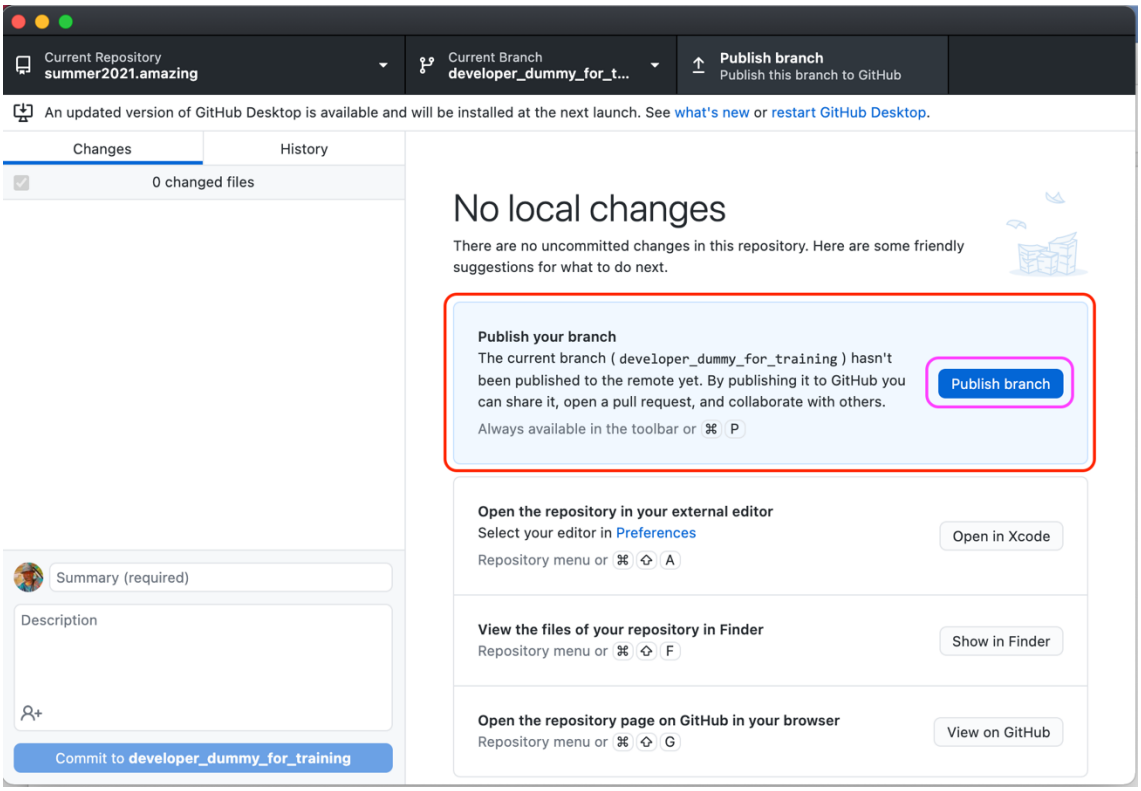

Image 7

- To get the latest code, simply select ***Fetch origin***. It will pull the latest code from the ***developer*** branch to your own branch (i.e. `developer_<your name>`).
- You are done.

Code flow between branches

The picture below (*Image 8*) depicts our code branching structure:

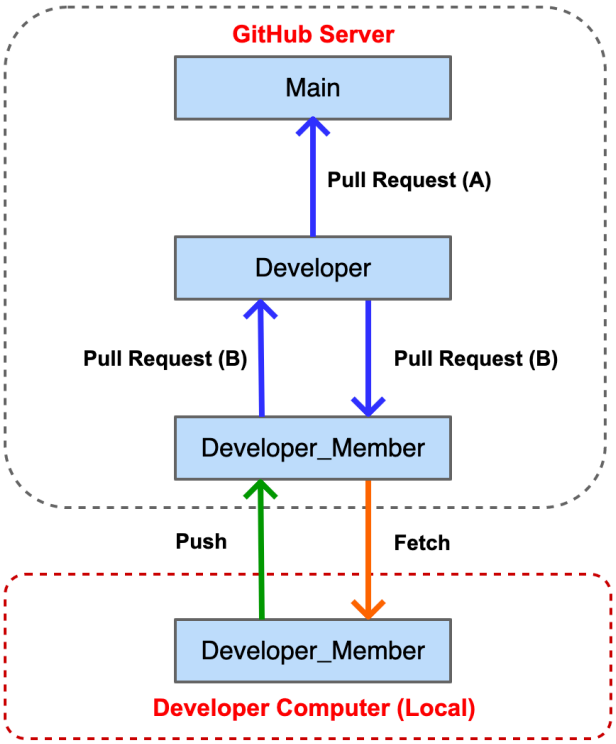

Image 8

5. Memory Enhancer Architecture

The section below will get the user familiar with the ideas behind the organization of the objects of the Memory Enhancer application.

5.1. Application Architecture

The architecture of the mobile application is organized as a set of cooperating components, as shown in the image below (*Image 9*).

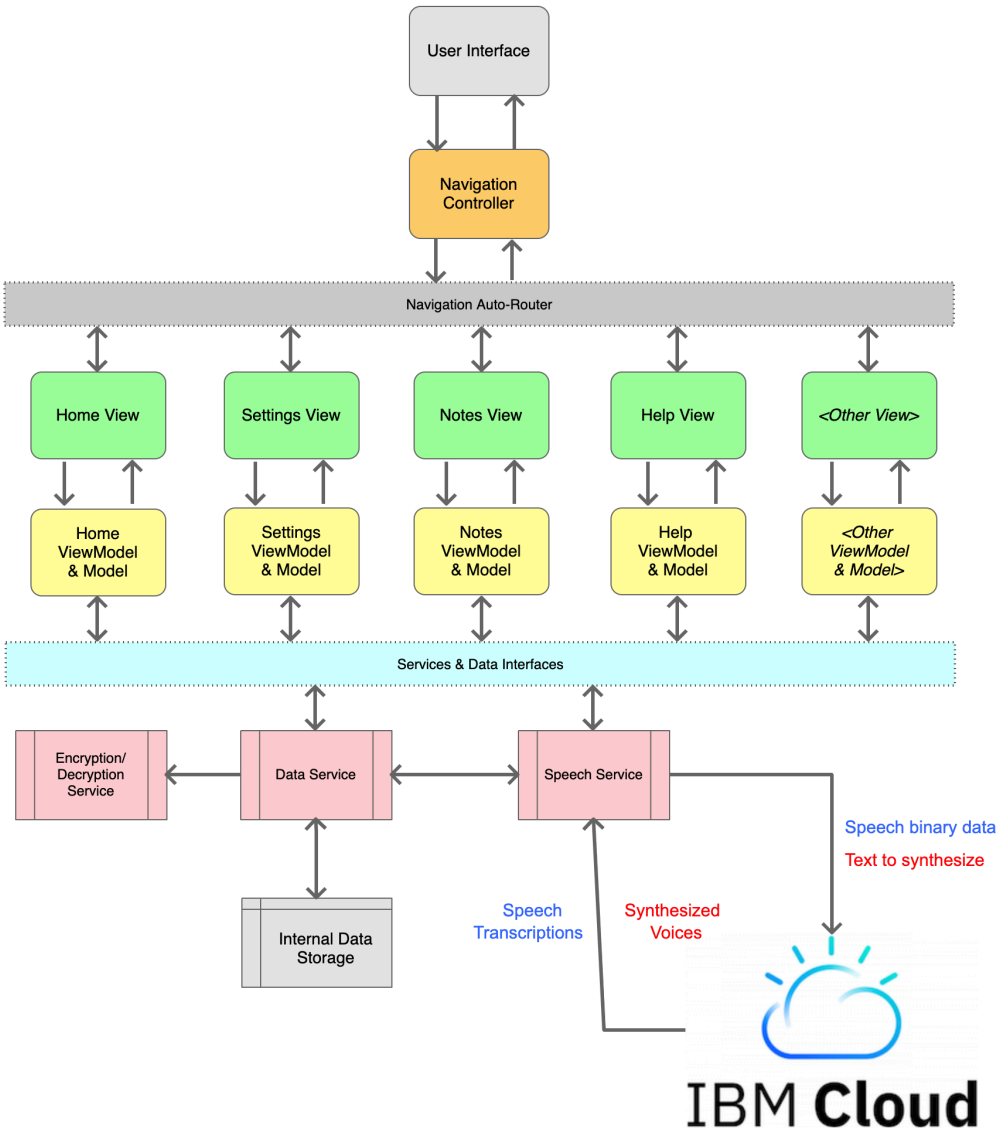

Image 9

5.2. MVVM Design Pattern

The MVVM, Model–View–ViewModel, design pattern will be used to implement the application. Similar to the MVC (Model–View–Controller) architecture, MVVM eliminates tight coupling between the application components leading to:

- Better application design
- Easier team development collaboration

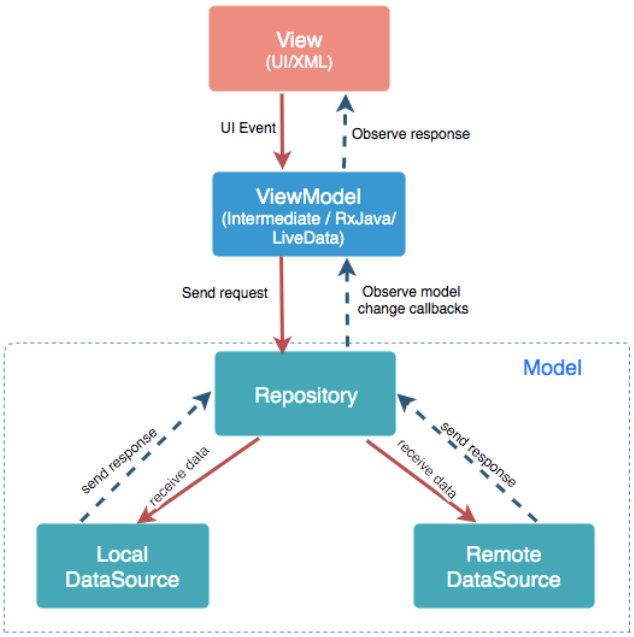

Image 10 – *MVVM design pattern*

5.3. Singleton Design Pattern

"The singleton design pattern restricts the instantiation of a class to a single instance. This is done in order to provide coordinated access to a certain resource, throughout an entire software system. Through this design pattern, the singleton class ensures that it's only instantiated once, and can provide easy access to the single instance" (Soni, 2019).

With a singleton implementation, only one instance of the singleton exists and reused throughout the application.

A singleton enables us to implement interfaces, can be passed as parameters and can be injected depending on the context (real implementation for the application functioning and mock-up for unit testing.

6. Code Structure

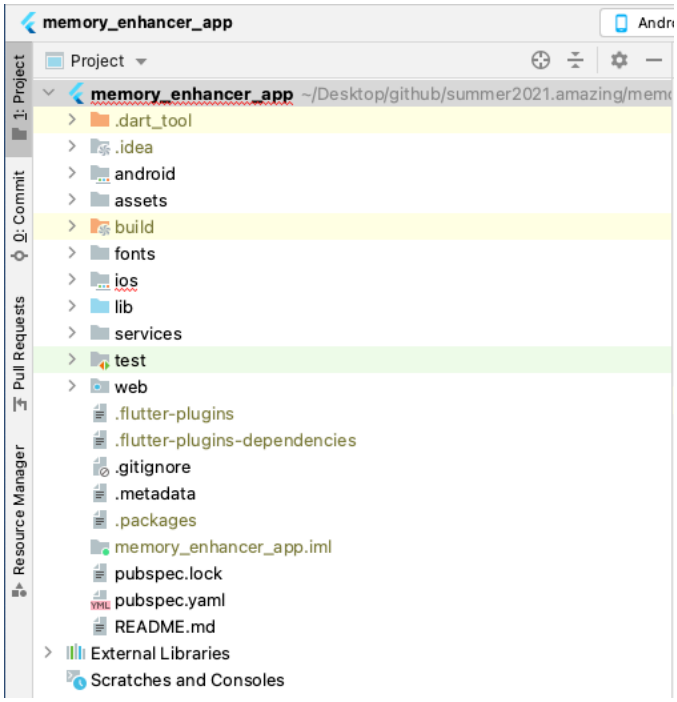

Image 11 *– Project code structure*

The image above depicts the project code structure as follow:

- **android** – folder that holds the Android native implementation. Android specific logic is done here. The native Android language for Java for this project.
- **assets** – folder where all the common assets are stored. Assets can be images, reusable files, etc.
- **fonts** – folder where the font family files are stores.
- **ios** – folder that holds the iOS native implementation. iOS specific logic is done here. The native iOS language for Swift for this project.
- **lib** – folder when the main logic of the application is implemented using the Dart language.
- **test** – folder where all the unit test files are located
- **web** – not used. Reserved for web version of the application
- **pubspec.yaml** – the project configurations file. This file includes the project important configurations such as:
    - The project name, description and version.
    - The project dependencies
    - The font families
    - The assets

7. Development

The section below will explain the tools used in order to develop the Memory Enhancement application. It is broken down into subsections explaining the main development components.

7.1. Development SDK

Flutter will be the used SDK for the development of the Memory Enhancer mobile application. Flutter is an open-source UI SDK that enables the cross-platform development of applications using one codebase. One codebase is used to generate an application in different platform, including Android and iOS. We will be able to implement the Memory Enhancer one time and deploy it both for Android and iOS devices.

The advantages include faster development and flexible user interfaces for a proper native performance.

7.2. Development Languages

Dart will be used as the main development language since it represents the foundation of Flutter by providing the language and runtimes running the Flutter applications. Dart enables the rapid development of mobile application in a cross-platform environment.

For necessary native platform specific logic, **Java** will be used for Android platform and **Swift** for the **iOS** platform.

7.3. Flutter Plugins and Libraries

Flutter plugins and libraries are available for search and review at pub.dev.

NOTE: *make sure that you imported Flutter plugins and libraries are null-safe. They are marked in pub.dev with the logo* ***"Null safety"****. If not, the imported plugin or library will cause the application build to fail because of the incompatibility with null-safe plugins and libraries already installed.*

The following Flutter plugins and libraries are used to support and facilitate reusability for the implementation of the Memory Enhancer application:

cupertino_icons

The library that provides the default set of icon assets used by Cupertino widgets. Icons used on various buttons and navigation links rely on this library.

avatar_glow

The package provides the Avatar Glow Widget with various background glowing animations. It is used to create a glow effect on the microphone (listen) button on the home page.

stacked

This library provides the architecture and widgets to implement a MVVM architecture in Flutter. It is used to build the MVVM architecture of the Memory Enhancer application.

get_it

A library providing direct service locator enabling the decoupling of the decoupling of interface from the implementation. It also facilitates the access the various services implementation from everywhere in the application. This library is, for example, used to facilitate the implementation of the speech recognition service, which can be accessed anywhere in the application by simply calling the service singleton instance.

injectable

A library that enables code generation for the ***get_it*** library. It enables the creation of the various app service as singleton (only one instance used throughout the application). It also enables us to inject service mockup for unit testing.

injectable_generator

Another library, like ***injectable***, that enables code generation for the ***get_it*** library.

auto_route

A library that facilitates the implementation the navigation for mobile applications. It is used to defines the navigation routings for the application pages.

auto_route_generator

This library enables the generation of the routes defined through the ***auto_route*** library.

_fe_analyzer_shared

This library enables the connections between the application front end and the analyzer packages.

porcupine

This library enables the implementation of an offline wake word engine. It is used in the Memory Enhancer application to initiate the application functionalities by simply saying a word.

wakelock

A plugin that enables us to maintain the mobile device screen awake.

stacked_themes

A library that enables the easy management of the application themes.

google_fonts

The Google library providing the font families.

xml

A library that enables XML manipulation, including parsing, traversing, querying, transforming and building XML documents.

nanoid

A library that enables the generation of the tiny, secure and friendly URL.

intl

A library that facilitates the handling of localized values, date and number formatting and parsing and other internationalization problems.

encrypt

A library that provides API's to encrypt and decrypt text and files. It is used by the application to encrypt the sensitive data before they are stored in local file.

youtube_player_flutter

A plugin that enables playing and streaming YouTube video through a frame. It is used by the application to support the implementation of the user "how to" videos to assist the user in using the application.

url_launcher

A plugin that facilitates to launching of a URL.

7.4. Implementing MVVM Design Pattern

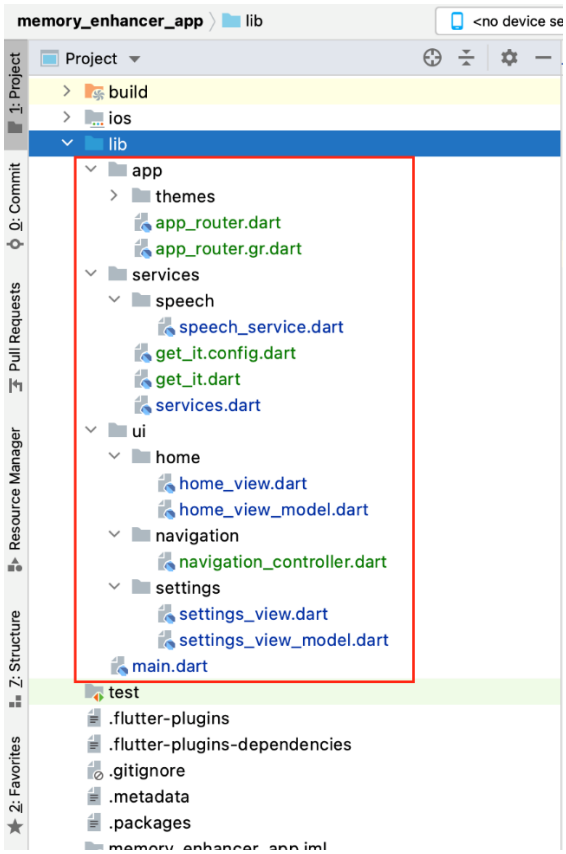

Image 12 *– Project Structure*

The MVVM implementation of the Memory Enhancer application is depicted by the project structure above (*Image 12*) and described as follow:

- **`main.dart`** – main file, now with less functionality (just the various services initialization).
- **`app_router.dart`** – for the navigation routing
- **`speech_service.dart`** – the singleton speech service. We can replace the Speech-to-text plugin to the Google API plugin here. Services functionality implemented in main.dart should be moved here.
- **`services.dart`** – all global variables will be declared here (such as *speechService*, *appRouter*, etc.).
- **`navigation_controller.bart`** – where the application navigation is implemented. Navigation business logic implemented in main.dart should be moved here.

For each view, create two (2) files: **`<functionality>_view.dart`** and **`<functionality>_view_model.dart`**.

- **`<functionality>_view.dart`** – UI design functionality will be implemented here. View design functionality implemented in main.dart should be moved here.
- **`<functionality>_view_model.dart`** – UI logic functionality will be implemented here. View business logic functionality implemented in main.dart should be moved here.

7.5. Implementing a Singleton Service

Implementing a singleton service, such as the Speech Service, enables us to have a single instance that can be reused by all the application components. To implement a singleton service:

- Create a new Dart file for the service (e.g. `speech_service.dart`)
- File should extend the class `ReactiveServiceMixin`.
- Import the following packages:

```
import 'package:injectable/injectable.dart';
import 'package:stacked/stacked.dart';
```

- Add the tag `@singleton` above the class declaration as follow:

```
@singleton
class SpeechService with ReactiveServiceMixin {
   // Class logic
}
```

- Open a terminal on the project by selecting **Terminal** at the bottom of the Android Studio IDE.

- Execute the command `flutter packages pub run build_runner build`
- The new singleton should be generated in the `get_it.config.dart` file and ready to use.

To make the new singleton available throughout the application:

- Open the `services.dart` file under `lib/services`.
- Import the instance dart file by adding `import 'package:<your singleton>.dart';`
- Create a new instance of the new singleton as such:

```
SpeechService get speechService {
  return getIt.get<SpeechService>();
}
```

- Your new service is ready to be used across the application. In the example above, to invoke the speech service, it is a simple matter of calling `speechService.<method>`

7.6. Implement Navigation for a New Page

Adding a new page and need to implement the navigation to that page, simply use the framework provided as follow:

- After creating the MVVM view, go to the `lib/app/app_router.dart` file.
- Add one entry as such `AutoRoute(page: <your-new-view-file-name>)` in the `@AdaptiveAutoRouter` array.
- Execute the command `flutter packages pub run build_runner build`
- The new navigation route should be generated in the `app_router.gr.dart` file and ready to use. It will look like this:

```
class <you-view-name>Route extends _i1.PageRouteInfo {
  const <you-view-name>Route() : super(name, path: '/');

  static const String name = '<you-view-name>Route';
}
```

To navigate to your new view, simply execute add the line below:

```
appRouter.pushAndPopUntil(<you-view-name>Route(), predicate: (_) => false);
```

8. Services Implementation

The section below introduces what technology stacks are used in the Memory Enhancer application along with the specifics as to how they work.

8.1. Speech Service

Description

The Speech Service is used for speech recognition and synthesis. The speech recognition transcribes user's voice to text, and speech synthesis convert text to speech. The Speech Service offers these two functionalities as Speech-to-Text (STT) and Text-to-Speech (TTS).

The client's initial request was for the application to be self-contained without a communication over the internet. However, no Flutter plugin or library was found to perform speaker diarization without connecting to a server. **Speaker diarization** is a needed feature as it separates all the speakers in an audio, and it will help identify only the user's voice.

Speech Cloud Service

IBM Watson cloud services is used to support the transcription with diarization and synthesis functionalities.

- **IBM Watson Speech-to-Text (STT)** is the IBM cloud service component that converts audio and voice into written text (IBM, n.d.-a). The Memory Enhancer application will use the voice to text module.
- **IBM Watson Text-to-Speech (TTS)** is the IBM cloud service component that converts written text into natural-sounding audio in a variety of voices (IBM, n.d.-b).

Implementation

The speech service is implemented in three (3) components:

- A cross-platform, implemented in Dart, which is used as the service API to the application components. This component is implemented in:

  `lib/services/speech/native_speech_service.dart`

- An Android-native component that uses the provided IBM Android-SDK to invoke the IBM Watson service APIs. This component is implemented in:

  `android/app/src/main/java/com/teamamazing/memory_enhancer_app/MainActivity.java`

- An iOS-native component that uses the provided IBM Swift-SDK to invoke the IBM Watson service APIs ***(under development).***

The **MethodChannel** is used to communicated between Flutter service and the native services.

Watson SDK Installation

To install the Watson SDK, which is a Java SDK, in the android native project folder, the following two dependencies were added to the `build.gradle` file under `android/app`.

```
dependencies {
    implementation 'com.ibm.watson:ibm-watson:9.2.0'
    implementation 'com.ibm.watson.developer_cloud:android-sdk:0.6.0'
}
```

Obtain IBM Watson Credentials

For development, each developer must obtain their own credentials to access each IBM cloud service. To obtain IBM cloud credentials, you will need an IBM cloud account. If you don't have one, create a new one at https://cloud.ibm.com/registration.

To obtain IBM Watson Speech-to-Text and Text-to-Speech credentials:

- Log in to your IBM cloud account at https://cloud.ibm.com/login
- Upon successful login, you will be redirected to the *Dashboard*.

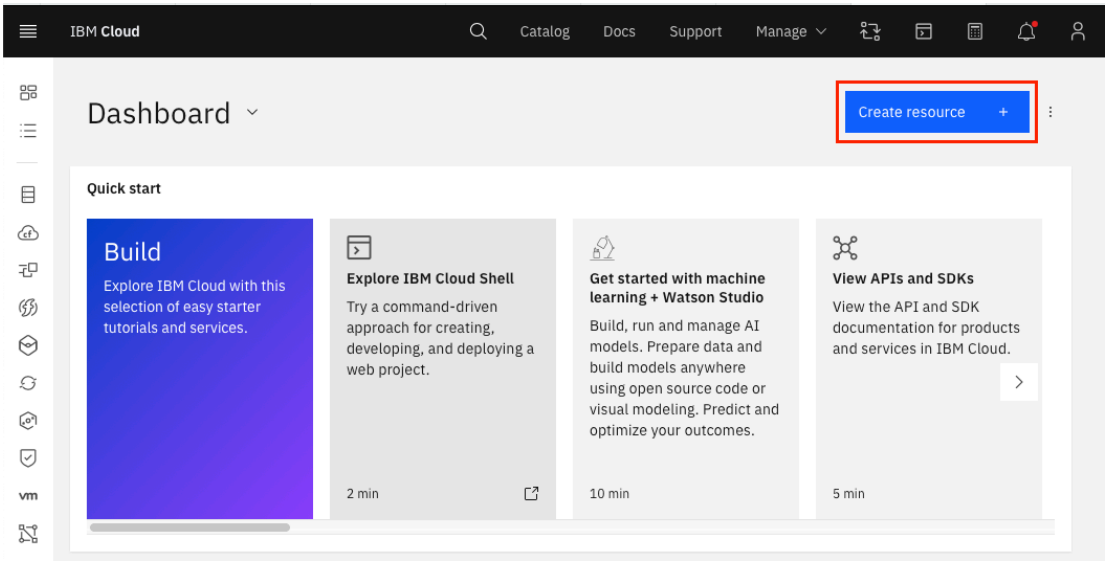

Image 13 *– IBM Cloud Dashboard*

- Click on the "Create resource +" button
- In the *IBM Cloud catalog* page, search a catalog (*i.e.* "Speech to Text" or "Text to Speech") and select the returned result
- In the Create tab,
  - Select the location closest to you. I am in Maryland and I selected "us-east"
  - Select the Lite plan, since it is only for development
  - Click on the "Create" button
- A new service resource (*i.e.* "Speech to Text" or "Text to Speech") for your account.
- On the new service resource page, copy the credentials (**API key** and **URL**)

Once you obtain the service credentials, you will have to add them to the project.

To set the credentials for Android:

- Open Memory Enhancer project in Android Studio
- Under the folder `android/src/main/res/values`, there is a `credentials.xml` file.
- Open the `credentials.xml` file.
- Insert the API key and URL for their respective service.
- Save the `credentials.xml` file and you are done.

To set the credentials for iOS:

- ***(Under development)***

8.2. Encryption Service

Description

The encryption service is used to encrypt sensitive information. It was implemented as a singleton service so that only one instance will exist and will be reused throughout the application. The service provides API methods to encrypt and decrypt texts or files. The Flutter library ***encrypt*** was used to implement the encryption and decryption functionality.

Implementation

The encryption logic uses the Advanced Encryption Standard (AES) specification, which a symmetric block cipher chosen by the United States government for the protection of their classified information.

The encryption service is implementation is located at:

`lib/services/encryption/encryption_service.dart`

9. Testing

Unit tests will ensure that every module does what it is supposed to do without issues. They will also be at the frontline of detecting any code regression. Programmers are required create unit tests for their codes.

9.1. Create a Unit Test

Unit tests should be implemented in the **`test/`** folder. To create a unit test:

- Create a Dart file in the **`test/`** folder.
- Import the Flutter package `flutter_test`

```
import 'package:flutter_test/flutter_test.dart';
```

- Import the dart file for the module you are attempting to test.
- Create a `main()` method.
- Create a single unit test method as such:

```
test("The test description", () async {

  // Your test logic here

  // Validate the run above
  expect(<expected result>, <actual result?);
});
```

- Save your test file.

9.2. Execute a Unit Test

To run a unit test class, a run configuration has to be set up. To do so:

- In Android Studio, click the configurations drop-down and select "Edit Configurations"

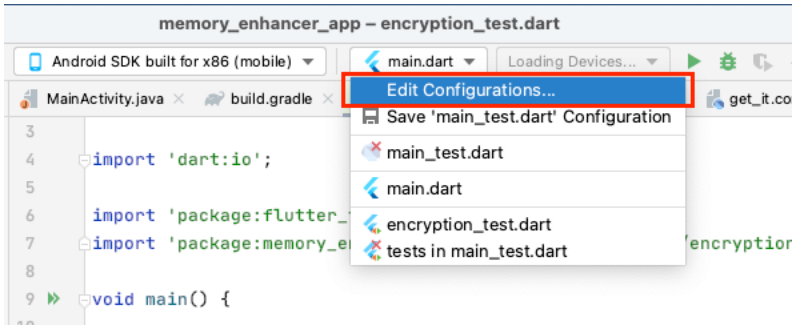

Image 14

- Click the + button on the "Run/Debug Configurations" window

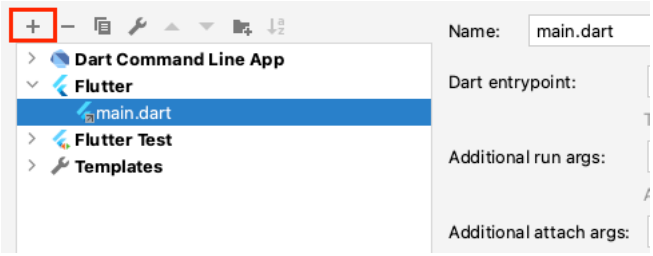

Image 15

- Select "Flutter Test"
- For the new configuration:
  - Enter a *Name* (usually the file name)
  - Select the *Test File* to test
  - Click on the OK button
- To execute the new test configuration, select it in the configuration drop-down and click on the run or debug button.

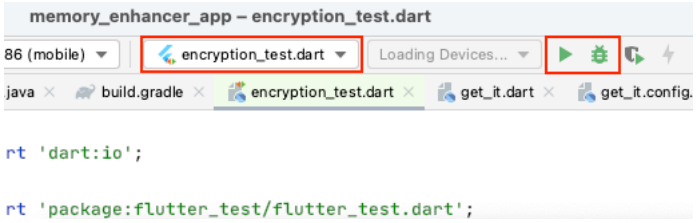

Image 16

10. Code Check-in Process

The code check-in process is dictated by the DevSecOps team, and it is subject to change anytime. To ensure the validity of the process mentioned below, consult with the DevSecOps team or the developer lead.

- Code should be committed and pushed from the *Developer* own local branch to the *Developer* personal branch on the GitHub on the server.
- Upon successful push, the developer should create a pull request from his server *Developer* personal branch to the *Developer* branch. The developer should select a team member as a reviewer to review their code. A second reviewer from the DevSecOps team will be added to the pull request.
- Upon the successful pull request review, the developer should merge his code to the *Developer*. It will now be available to everyone mapped to the *Developer* branch.

Once the code is stable, the developer must create a pull request to move their code from the *Developer* branch to the *Main* branch. The added reviewer on this pull request should only be from the DevSecOps team.