MemorEz Programmer's Guide

Combined Version 1

Combined/Co-Authored by James Eble

UMGC SWEN670 – Spring 2022 (From submissions by FlutteringMind and RememberAll)

Professor Dr. Mir Assadullah

Table of Contents (Section 1: Patient Mode)

1 Introduction 8
1.1 Purpose 8
1.2 Intended Audience 8
1.3 Definitions, Acronyms, and Abbreviations 10
1.4 References 11
2 Tools 12
2.1 IDEs 12
2.1.1 Visual Studio Code 12
2.1.2 Android Studio 12
2.1.2.1 Download and Installation 12
2.1.2.2 Setting Up an Emulator 12
2.1.2.3 Setting Up a Device 12
2.1.3 Setting Up a Simulator (MacOS) 13
2.1.3.1 Download and Install Xcode 13
2.1.3.2 Setting Up and iOS Simulator 13
2.1.4 Setting Up an Emulator 13
2.2 GitHub 13
2.3 Pub.dev 13
3 Frameworks and Languages 14
3.1 Dart 14
3.1.1 Overview 14
3.1.2 Installation 14
3.1.3 Null Safety 14
3.2 Flutter 14
3.2.1 Overview 14
3.2.2 Installation 14
4 Application Development Code 16
4.1 Code Repository 16
4.2 Cloning the Code Repository 16

4.3 Code Branches 17
5 Platforms 21
5.1 Android 21
5.2 iOS 21
6 File Structure 22
6.1 Overview 22
6.2 pubspec.yaml 22
6.2.1 Description 22
6.2.2 Fields 22
6.3 Assets and Images 24
7 User Interface 25
7.1 Front-End Components 25
7.2 Front-End Screens 26
7.3 Navigation and Routing 33
8 Data and Backend 36
8.1 JSON Serialization 36
8.2 Shared Preferences and SQLite 37
9 Accessibility and Internationalization 38
9.1 Accessibility 38
9.2 Internationalization 39
10 Packages 41
10.1 Package Usage 41
10.2 Package Implementation 42
10.2.1 MobX 44
10.2.2 i18n 45
11 Services 47
11.1 Application Services 47
12 AWS Lex 48
12.1 Overview 48
12.2 API Endpoint 48

13 Testing 49
14 Exceptions 50
15 Appendices 52
15.1 Credits 52
15.2 Credits to Previous Cohorts 52

Table of Contents (Section 2: Caregiver Mode)

1 Introduction 54
1.1 Purpose 54
1.2 Intended Audience 54
1.3 Definitions, Acronyms, and Abbreviations 55
1.4 References 55
2 Tools 57
2.1 IDEs 57
2.1.1 Visual Studio Code 57
2.1.2 Android Studio 57
2.1.2.1 Download and Installation 57
2.1.2.2 Setting Up an Emulator 57
2.1.2.3 Setting Up a Device 58
2.1.3 Setting Up a Simulator (MacOS) 58
2.1.3.1 Download and Install Xcode 58
2.1.4 Setting Up an iOS Simulator 58
2.2 GitHub 58
2.3 Pub.dev 58
3 Frameworks and Languages 59
3.1 Dart 59
3.1.1 Overview 59
3.1.2 Installation 59
3.1.3 Null Safety 59
3.2 Flutter 59
3.2.1 Overview 59
3.2.2 Installation 59
3.2.3 Hot Reload 60
4 Application Development Process 61
5 Platforms 63

5.1 Android 63
5.2 iOS 63
5.3 Web 63
6 File Structure 64
6.1 Overview 64
6.2 pubspec.yaml 64
6.2.1 Description 64
6.2.2 Fields 65
6.3 Assets and Images 66
7 User Interface 67
7.1 Widgets 67
7.1.1 Basic and Layout Widgets 67
7.1.2 Other Widgets 67
7.2 Navigation and Routing 68
8 Data and Backend 69
8.1 JSON Serialization 69
8.2 Natural Language Understanding 70
8.2.1 Connecting to LEX via NLU 70
8.2.2 LEX/NLU Components and Architecture 70
9 Accessibility and Internationalization 73
9.1 Accessibility 73
9.2 Internationalization 74
10 Packages 75
10.1 Package Usage 75
10.2 Package Implementation 75
10.2.1 MobX 77
10.2.2 i18n 77
11 Services 80
12 Testing 81

13 Exceptions 82

Revision History for Combined Programmer's Guide:

Version	**Date**	**Reason**	**Approved By**
1.0	3/24/2022	Milestone 3	James Eble

Section 1: Patient Mode

Information in this section is copied from the team FlutteringMind Programmer's Guide for the MemorEz application.

1 Introduction

This document details programming for the implementation and the detailed instructions of implementation of the MemorEZ application including all the tools and development software that the team of Spring 2022 semester of SWEN670 have utilized. In this section, purpose, intended audience, acronyms, and references are described.

1.1 Purpose

For the Spring 2022 semester of SWEN670 at UMGC, the class has been assigned to detail and implement an application to be used by those with short term memory loss (STML) disabilities. This document details programming for the implementation of the MemorEZ version of the application including tools and development software. This documentation for Programmer's Guide can be used as a helpful guide for the future developers for SWEN670 in the upcoming semesters.

1.2 Intended Audience

This document is intended for use by 2022 Spring semester and future iterations of UMGC SWEN670 IT: Software Engineering courses.

Table 1.1

2022 Spring Semester – SWEN670

Team Role	Team Member Names
Product Owner	Dr. Mir Assadullah
Subject Matter Expert (SME)	Andrea (Nursing homeowner) Dr. Evangelista
Team Role	**Team Member Names**
Mentor	Roy Gordon Robert Wilson Daniel Avery
Project Manager	Selina Zaman

Requirements Specialist	Vanessa Stringer
Business Analyst	Selina Zaman Sean LaMonica Vanessa Stringer Anusha Ramanan Daryle Urrea Joseph Jewell
UI/UX Designer	Joshua Fischer
Lead Developer	Daryle Urrea
Developer	Joseph Jewell Vanessa Stringer Selina Zaman Joshua Fischer
Tester	Anusha Ramanan Sean LaMonica Daryle Urrea Joseph Jewell Selina Zaman

1.3 Definitions, Acronyms, and Abbreviations

- **AOT** – Ahead-of-time compiler
- **ARIA** – Accessible Rich Internet Applications
- **Dart** – Programming language used in the Flutter framework
- **Flutter** – Framework for developing cross platform applications
- **IDE** – Integrated Development Environment
- **JIT** – Just-in-time compiler
- **pubspec.yaml** – Configuration file used in Flutter framework
- **RPC** – Remote Procedure Call ("Remote procedural call", n.d.)
- **Sentinel -** A Sentinel is used to indicate that the normal response is not available. ("Sentinel class", n.d.)
- **STML** – Short Term Memory Loss
- **UI** – User Interface
- **VM** – Virtual Machine

1.4 References

Accessibility. (n.d.). Flutter. Retrieved February 10, 2022, from https://docs.flutter.dev/development/accessibility-and-localization/accessibility .

Blazebrain. (2021, December 4). *Using services in flutter*. DEV Community. from https://dev.to/blazebrain/using-services-in-flutter-572h

Build apps for any screen. (n.d.). Flutter. Retrieved February 9, 2022, from https://flutter.dev/ .

Dart overview. (n.d.). Dart. Retrieved February 9, 2022, from https://dart.dev/overview .

Exception Class. (n.d.). Flutter. Retrieved February 10, 2022, from https://api.flutter.dev/flutter/dart-core/Exception-class.html .

Flutter and the pubspec file. (n.d.). Flutter. Retrieved February 12, 2022, from https://docs.flutter.dev/development/tools/pubspec .

Internationalizing Flutter apps. (n.d.). Flutter. Retrieved February 10, 2022, from, https://docs.flutter.dev/development/accessibility-and-localization/internationalization .

karina karina 55722 gold badges77 silver badges1515 bronze badges, et al. "What Does an Question (?) Mark and Dot (.) in Dart Language?" *Stack Overflow*, 1 Apr. 1967, https://stackoverflow.com/questions/56223120/what-does-an-question-mark-and-dot-in-dart-language.

Remote procedural call. (n.d.) Flutter. Retrieved February 11, 2022, from https://en.wikipedia.org/wiki/Remote_procedure_call .

Run apps on the Android Emulator. (n.d.). Android Developers. Retrieved February 9, 2022, from https://developer.android.com/studio/run/emulator#install .

Sentinel class. (n.d.). Flutter. Retrieved February 11, 2022, from https://api.flutter.dev/flutter/vm_service/Sentinel-class.html .

Supported platforms. (n.d.). Flutter. Retrieved February 10, 2022, from https://docs.flutter.dev/development/tools/sdk/release-notes/supported-platforms .

SWEN670, Project Plan, Team Mesmerize. (2021, August 30). Project Plan. Memory_magic_project_plan_final.docx Retrieved from https://umgc-cappms.azurewebsites.net/previousprojects

The pubspec file. (n.d.). Dart. Retrieved February 12, 2022, from https://dart.dev/tools/pub/pubspec .

Testing flutter apps. Flutter. (n.d.). from https://docs.flutter.dev/testing#unit-tests

Using Packages. (n.d.). Flutter. Retrieved February 10, 2022, from https://docs.flutter.dev/development/packages-and-plugins/using-packages .

2 Tools

There are various types of tools that the team utilized for this project.

2.1 IDEs

2.1.1 Visual Studio Code

- Download corresponding version of Visual Studio Code from: https://code.visualstudio.com/download
- For installation on Windows follow instructions at: https://code.visualstudio.com/docs/setup/windows
- For installation on macOS follow instructions at: https://code.visualstudio.com/docs/setup/mac

2.1.2 Android Studio

2.1.2.1 Download and Installation

- Download corresponding version of Android Studio from: https://developer.android.com/studio
- For instructions on how to install Android Studio on Windows see: https://developer.android.com/studio/install
- For instructions od how to install Android Studio on macOS see: https://developer.android.com/studio/install

2.1.2.2 Setting Up an Emulator

Information in this section is taken from the Android Developer website ("Run apps on the Android Emulator," n.d.).

To install the Android Emulator in Android Studio, select the **Android Emulator** component in the **SDK Tools** tab of the **SDK Manager**. For instructions on updating tools with the SDK Manager see: https://developer.android.com/studio/intro/update#sdk-manager

2.1.2.3 Setting Up a Device

For instructions about setting up and running applications on a hardware device see: https://developer.android.com/studio/run/device

2.1.3 Setting Up a Simulator (macOS)

2.1.3.1 Download and Install Xcode
To download and install Xcode visit the Mac App Store:
https://apps.apple.com/us/app/xcode/id497799835?mt=12

2.1.3.2 Setting Up an iOS Simulator
For instructions about how to set up an iOS simulator on macOS see this tutorial:
https://www.macinstruct.com/tutorials/setting-up-an-ios-simulator-on-your-mac/

2.1.4 Setting Up an Emulator

2.2 GitHub

Project code is stored in a GitHub repository that can be accessed using command line tools. Alternatively, programmers can use a desktop version of GitHub that can be downloaded from https://desktop.github.com/ . For more information on GitHub implementation see section 4.

2.3 Pub.dev

The pub.dev website hosts the official repository for Dart and Flutter applications. Packages are installed using the Flutter Pub Add command. The installed packages must also be imported in the code file.

3 Frameworks and Languages

3.1 Dart

Information in this section is derived from the Dart language website ("Dart overview", n.d.).

3.1.1 Overview

Dart is a programming language utilized by the Flutter framework. For mobile applications Dart uses the Dart VM with just-in-time (JIT) compilation and an ahead-of-time (AOT) compiler. For web applications Dart uses compilers to translate Dart into JavaScript.

3.1.2 Installation

Dart is automatically downloaded with the latest versions of Flutter. For Flutter installations see section 3.2 of this document. To install Dart independently see: https://dart.dev/get-dart

3.1.3 Null Safety

A consideration when using Dart is its support for null safety. This means that variables in dart cannot have a null value unless specifically indicated. Here is an example of how to use dart to declare a variable that accepts null value:

int? myInt = null; (returns null if the value is null)

It is important to consider null safety when selecting packages in dart.

3.2 Flutter

3.2.1 Overview

Flutter is a development framework that is used to build cross platform applications. For more about platforms see section 7 of this document. Flutter uses Dart for its programming language. For more information about Dart see section 2 of this document.

3.2.2 Installation

- For system requirements and installation of Flutter on Windows see: https://docs.flutter.dev/get-started/install/windows
- For system requirements and installation of Flutter on macOS see: https://docs.flutter.d3.2.3 Hot Reload
- Flutter provides a hot reload ev/get-started/install/macos

feature through the Dart VM. This feature allows code changes to be seen in the currently running application without rebuilding the entire app. To implement a hot reload, the flutter app must be running in debug mode. If running in an IDE the hot reload is run by clicking the **Save All** button. If running from the console then type r and press **Enter**.

4 Application Development Code

This section will provide the process details adapted by developers during the code development of the application.

4.1 Code Repository

The MemorEz application code is located in GitHub repository under the name "spring 2022" (https://github.com/umgc/spring2022)

4.2 Cloning the Code Repository

To access the code, clone the spring2022 repository with one of these options:

• Clone with a Git command using https://github.com/umgc/spring2022

• Open https://github.com/umgc/spring2022 in GitHub Desktop

GitHub Desktop instructions

• Open https://github.com/umgc/spring2022

• Login using your GitHub credentials created for the project.

• Once logged in, you will be asked to select a repository.

• Select the spring2022 repository and provide a folder location where you would like your repository to be on your machine (e.g. "C:\user\documents\GitHub\spring2022").

• Select the repository drop-down > Add > Clone Repository.

• On the Clone a Repository pop-up, select the GitHub.com tab, scroll down to locate the spring2022 repository.

• Select the spring2022 repository.

• Provide a local path (folder location) for the repository clone.

• Click the Clone button.

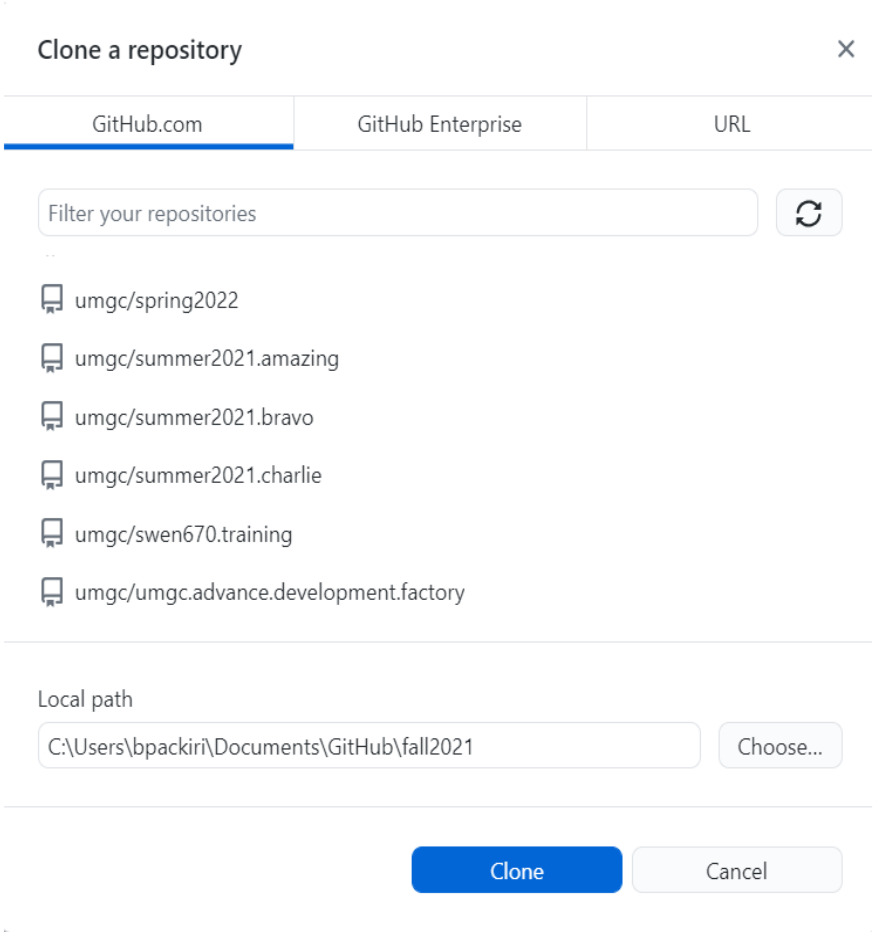


Figure 1 Repository

4.3 Code Branches

The team uses two code branches for development: Main and Development

Main Branch

The Main branch holds the stable version of application code, 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 Development branch. Warning: NO development should occur in this branch.

Development Branch

The Development branch is used by all development team members to clone and have their local branch to do their work. The same branch is also used for any parallel development work.

Branch naming convention

To remain uniform in how the branches are created, the branch naming convention should follow the <feature>/<feature-name> or<bug>/<bug-name> For example, a developer may create his developer branch as feature/settings or bug/calendar search.

Create your personal developer branch

• Open the GitHub Desktop application.

• Login using your GitHub credentials created for the project.

• Clone spring2022 repository.

• Select the repository spring2022.

• Select the "development" branch.

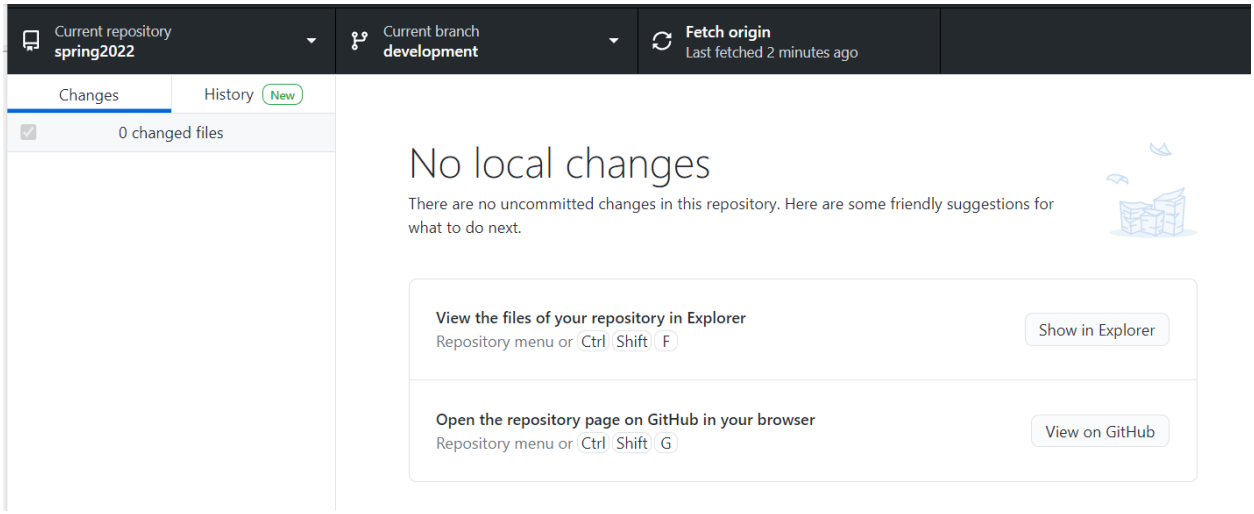


Figure 2 GitHub Desktop

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

•On the Create a Branch pop-up, ensure that the development branch is the default (your new branch should be mapped to development); provide a name for your branch (i.e. feature_) and click the Create Branch button.

•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 .

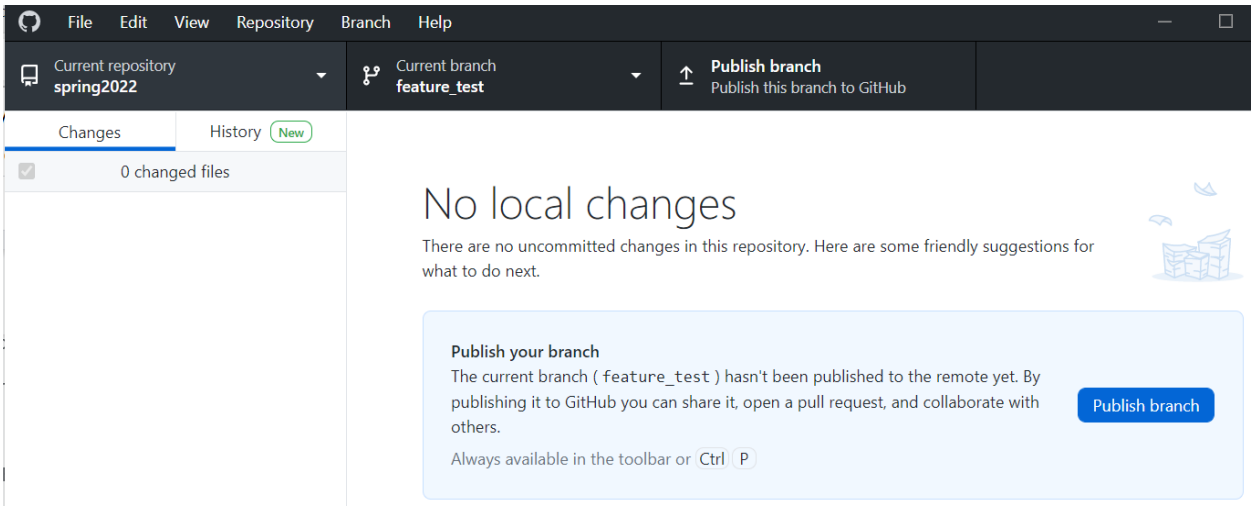


Figure 3 GitHub Desktop

• To get the latest code, simply select Fetch origin. It will pull the latest code from the development branch to your own branch.

• You are done.

Code flow between branches

The picture below depicts our code branching structure:

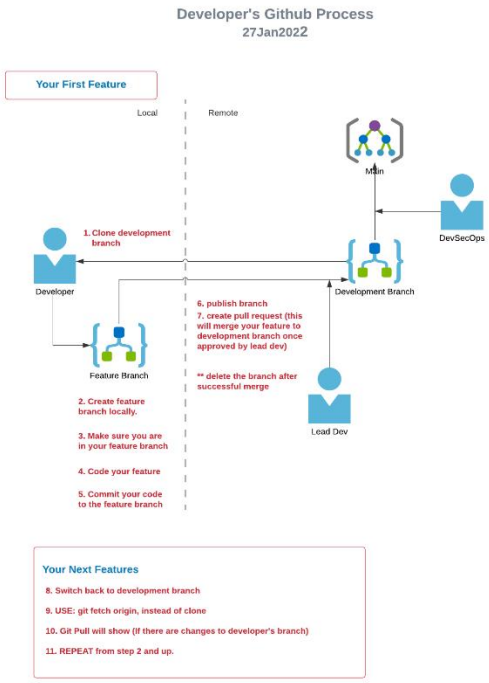


Figure 4 GitHub workflow

5 Platforms

The following information lists the current supported IOS and Android platform as stated in the Flutter documentation (Supported platforms, n.d.).

5.1 Android

- **Supported:** API 19 & above
- **Not Supported:** Android SDK 18 & below

5.2 iOS

- **Supported:** iOS 9 & above
- **Not Supported:** iOS 8 & below

6 File Structure

6.1 Overview

The Model View ViewModel pattern separates UI and business logic for maintainability, testability and extensibility. The ViewModel acts as the mediator between the View and the Model, the Model represents the database related queries and the View is where the users interact through the widgets built in the screen.

6.2 pubspec.yaml

6.2.1 Description

The function of pubspec.yaml in a Flutter application is described in this section contains information from the official Flutter website (Flutter and the pubspec file, n.d.).

The pubspec file is automatically generated containing the metadata when a new Flutter application is created. The YAML programming language is used to create the pubspec file. The project dependencies, development dependencies, and versions are all listed in this file. Before building the application, the IDE uses this file to import appropriate files for each dependency.

6.2.2 Fields

The available fields in the pubspec file covered in this section contain information derived from the official Dart website (The pubspec file, n.d.).

Table 6.1

Fields Supported in the Pubspec File

Field	Requirement/Description
name	Required for every project as the package's name.
version	As every package has a version, it is required for packages that are hosted on the pub.dev site.
description	Optional for locally hosted packages but required for packages that are hosted on the pub.dev site.
homepage	This URL will point to the package's homepage (or source code repository). Optional to have.

repository	Optional to have and the URL points to the package's source code repository.
issue_tracker	Optional to have and the URL points to an issue tracker for the package.
documentation	Optional to have and the URL points to documentation for the package.
dependencies	Can be deleted if the package has no dependencies.
dev_dependencies	Can be deleted if the package has no dev dependencies.
dependency_overrides	Can be deleted if there is no need to override any dependencies.
environment	Required as of Dart 2.
executables	Optional to have and can be used to put a package's executables on your PATH.
platforms	Optional to have and can be used to explicitly declare supported platforms on the pub.dev site.
publish_to	Optional and specifies where to publish a package.
false_secrets	Optional and specify files to ignore when conducting a pre-publishing search for potential leaks of secrets.

Flutter Note: Additional fields for configuring the environment and managing assets can be added to Pubspecs for Flutter apps.

6.3 Assets and Images

Assets and Images are static files that are used by the app. These files must be declared as assets within the pubspec.yaml file.

- **care_giver_patient_image.png:** Image asset rendered in the homepage screen.
- **help_content.json:** Contains a list of sections for the help screen.
- **lite-model_mobilebert_1_metadata_1.tflite:** Model data file for BertQA, a question and answer feature for the NLU service.
- **Lex_Credentials.xml:** Contains credentials for connecting to AWS Lex.
- **animation.json:** A JSON-based animation file used for onboarding splash screens.

7 User Interfaces

7.1 Front End Components

The front-end components are located in the lib > Screens section of the application. The components are further distributed into the following folders, each representing a feature:

- Calendar – files related to the calendar feature of the application.
- Components – files related to the video player used on the Help screen.
- Mic – files related to the application microphone feature.
- Note – files related to the note screens of the application.
- Onboarding – files related to the onboarding screens of the application.
- Profile – files related to the profile screen of the application.
- Settings – files related to the settings screen of the application.
- Splash – files related to the initial splash screen of the application.
- Tasks – files related to the task feature screens.

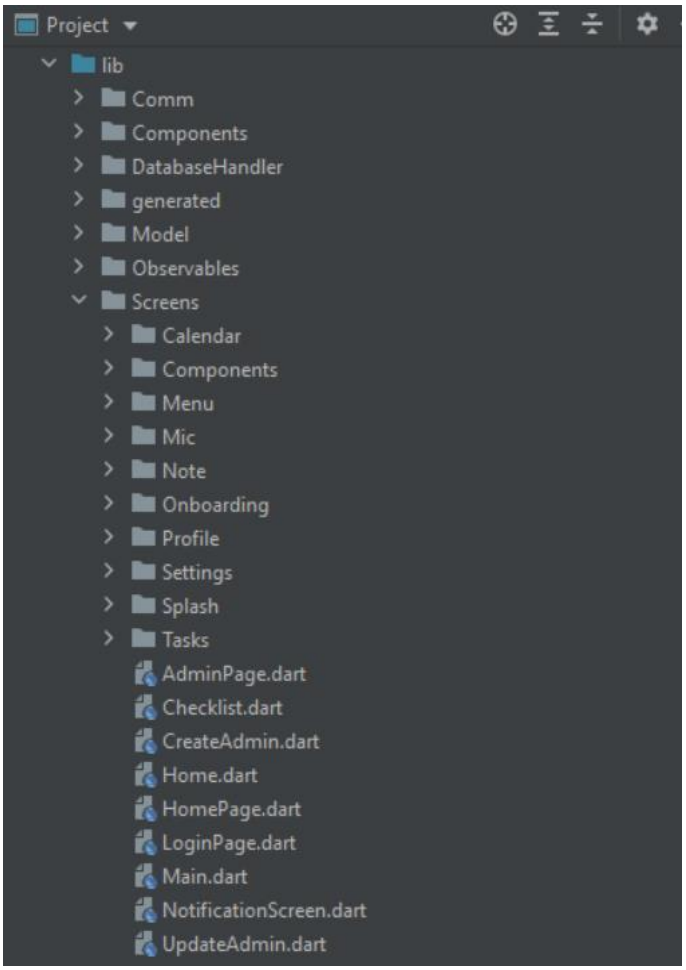


Figure 5 Code Structure of Front-End Components

7.2 Front-End Screens

The MemorEz application utilizes a modularized design consisting of a series of screens representing the application's features including the calendar, notes, profile, tasks, and settings. Content and features visible on screens in the Patient mode of the application will differ from the content and features available in the Caregiver mode of the application. The screens featured below may differ from the final product.

The Splash screen is the initial screen displayed to a user upon successfully downloading the MemorEz application. The user can elect to begin using the application as a patient or set up Caregiver mode.

Figure 6 Splash Screen

The Menu is set as the home screen once the user has *** and displays a column of buttons that navigate the user to the screens of the application.

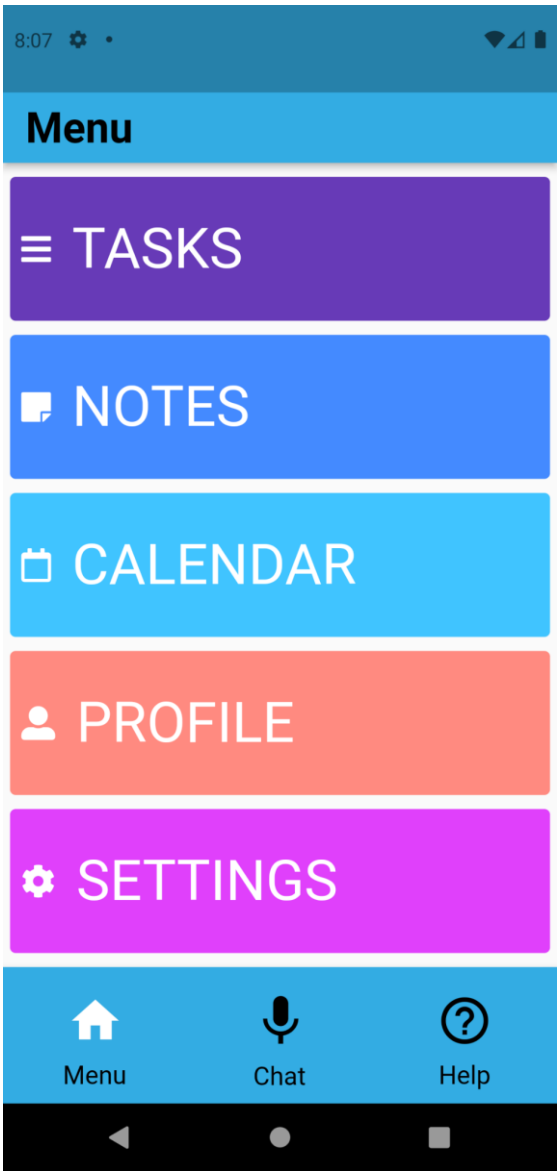


Figure 7 Home Screen

The Tasks screen enables the user to view and complete tasks assigned by a caregiver on the device via the Caregiver mode of the application.

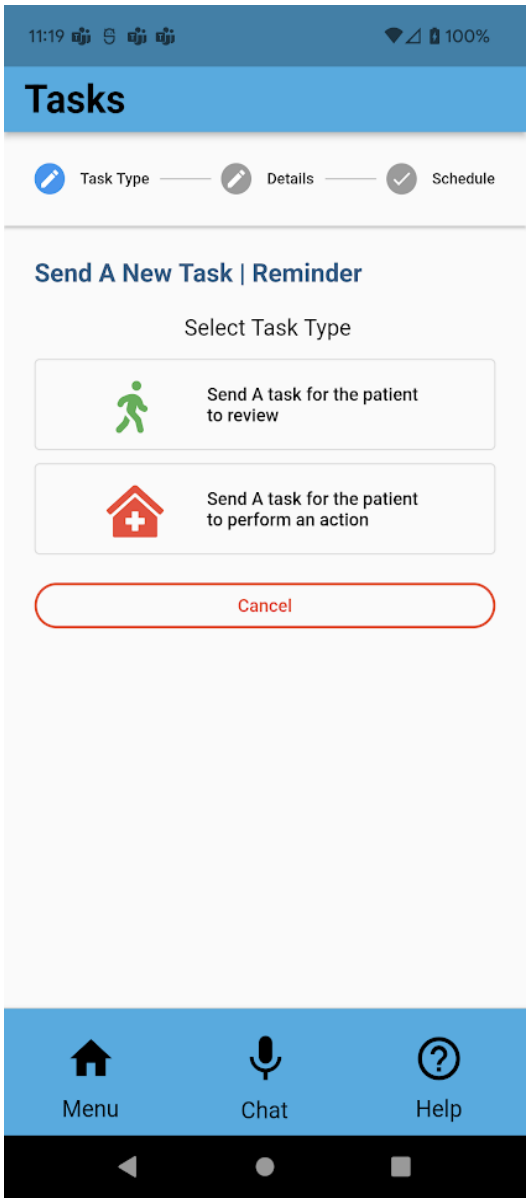


Figure 8 Tasks Screen

The Notes screen allows a user to view existing notes, search for an existing note, and add a new note. Selecting the Add Note button enables the user to say or enter text that can be saved as a note. A note can be modified by tapping the desired note.

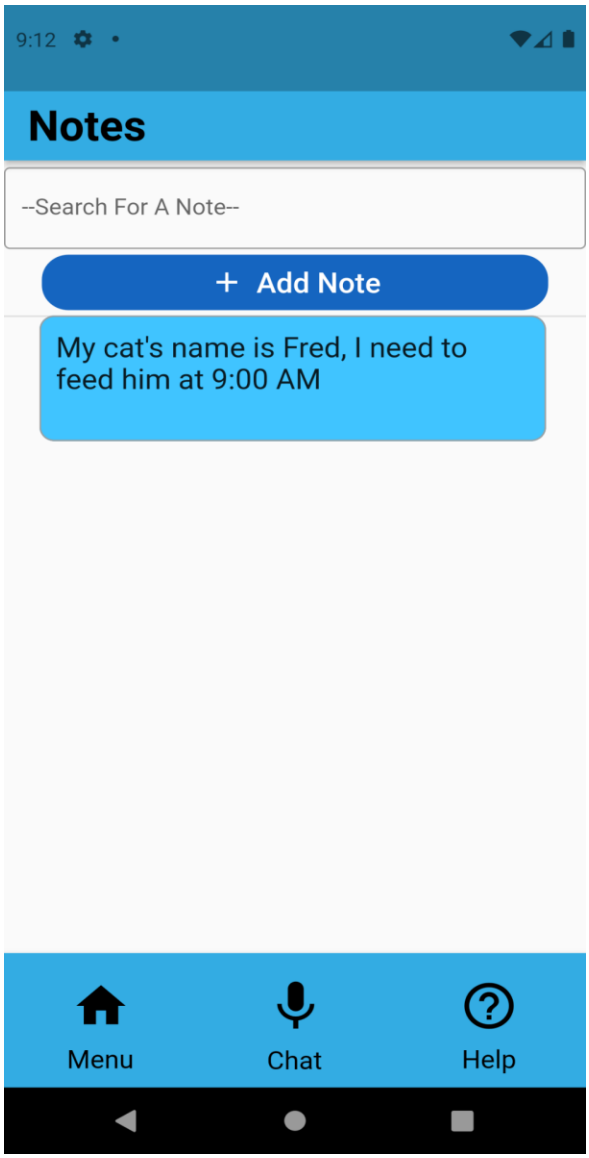


Figure 9 Notes Screen

The Calendar screen displays a calendar that can be viewed by week, month, or day.

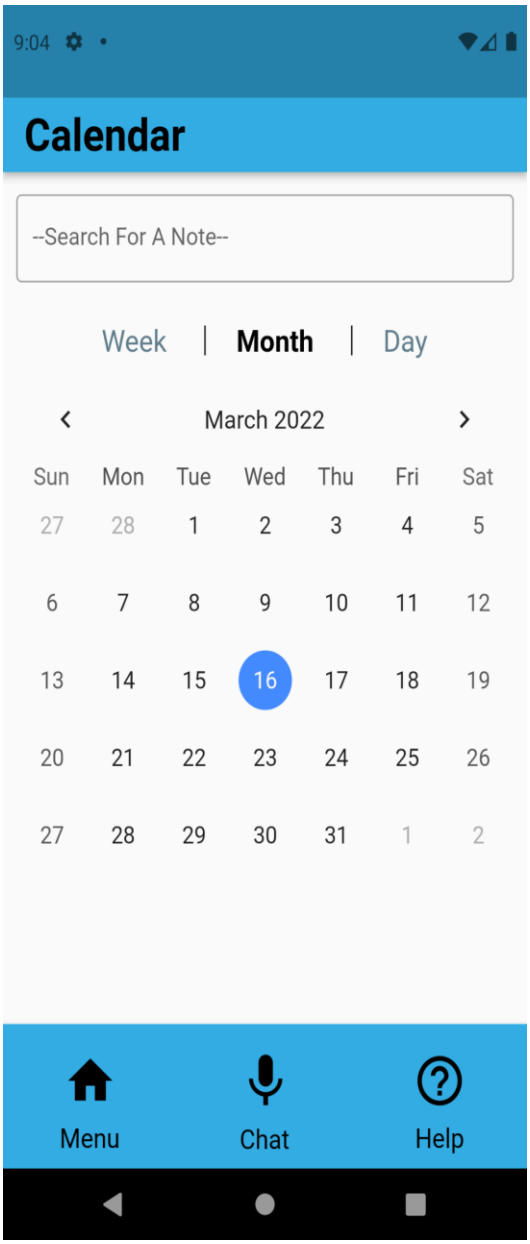


Figure 10 Calendar Screen

The Profile screen displays information entered and modified from the caregiver mode in a scrollable read-only list divided into sections.

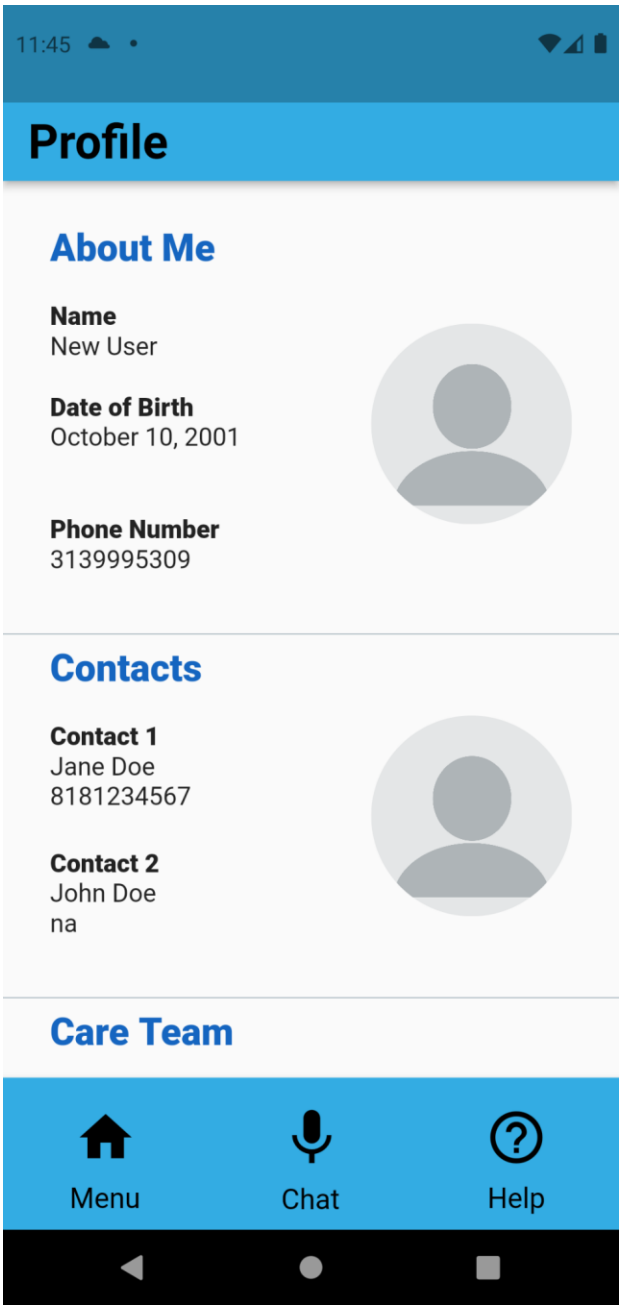
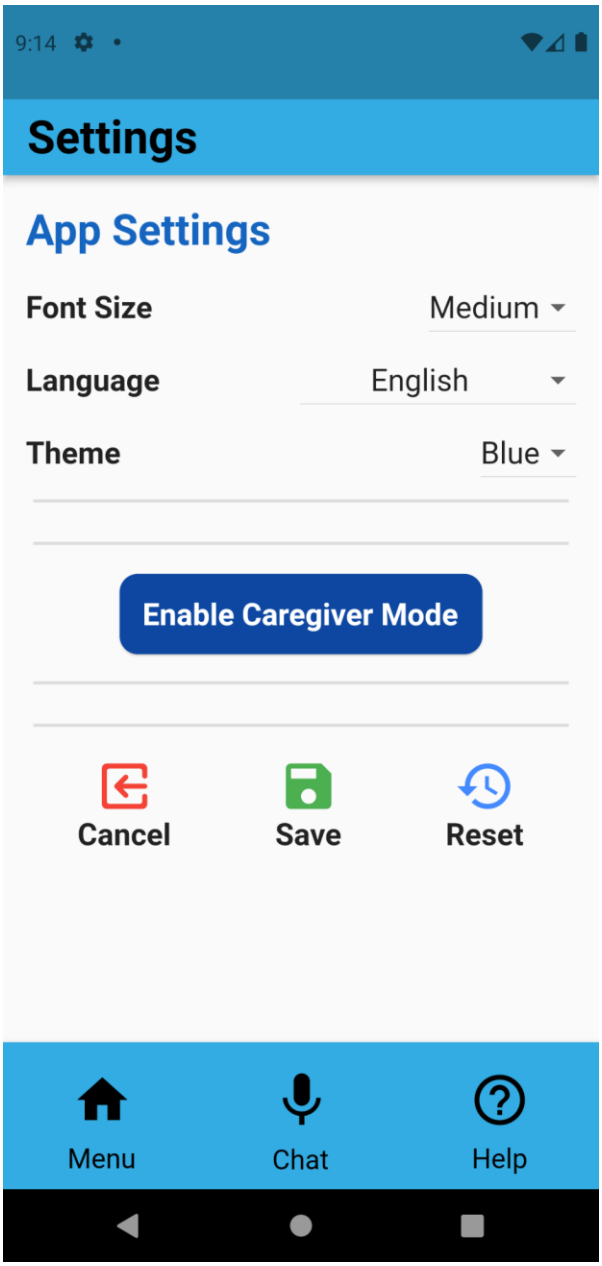


Figure 11 Profile

The Settings screen allows the user to configure font size, language, and theme from patient mode. Additionally, the Settings screen provides access to enabling the caregiver mode where additional settings and application features can be accessed by the user.



7.3 Navigation and Routing

The MemorEz application leverages the use of a menu home screen for users with large, labeled buttons that are easily distinguished by color. The navigation interface of the application is designed to be intuitive and simple. At the top of the screen, a scaffold persists throughout the screens of the application and displays the name of the screen that the user is currently on. At the bottom of the screen, a three button navigation bar persists throughout all screens on the application to allow the user to return to the Menu home screen, utilize the mic feature, or navigate to the Help screen

Navigation in the application from the Menu uses the flutter_mobx Observer widget to listen to observables and upon selection of a menu screen button returns the desired screen as displayed in the following code:

```
Widget build(BuildContext context) {
  final noteObserver = Provider.of<NoteObserver>(context, listen: false);
  final menuObserver = Provider.of<MenuObserver>(context);
  final screenNav = Provider.of<MainNavObserver>(context);
  return Observer(
    builder: (_) => (menuObserver.currentScreen == MENU_SCREENS.MENU)
…//code for TextButton with screen name
onPressed: () {
   screenNav.changeScreen(MAIN_SCREENS.TASKS);
}
```

The Navigator and Router widgets are also used throughout the feature screens of the application in combination with the materialPageRoute constructor and named routes to enable use of Navigator.push/Navigator.pop, allowing the user to navigate to a new screen and back. The following code allows the user to navigate to a new "Add Medication" screen using Navigator.push.

```
…//elevated button code
  onPressed: () => {
                        Navigator.push(
                         context,
                         MaterialPageRoute(
                           builder: (_) => AddMedicationCard(
                              updateMedicationList:
                                 _updateMedicationList),
                         ),
```

The user can navigate back to the previous screen from the Add Medication screen by selecting the back arrow icon, which uses Navigator.pop to return the user to the Profile screen.

```
GestureDetector(
          onTap: () => Navigator.pop(context),
          child: Icon(
```

```
                Icons.arrow_back_ios,
                size: 30,
                color: Theme.of(context).primaryColor,
              ),
),
```

8 Data and Backend

This section describes the data and storage implementations in the MemorEZ application. While the logic/backend of the application is built using flutter/dart, there are a few specific plugins that were used for data storage and data persistence in the application.

8.1 JSON Serialization

Information on JSON serialization in this section is taken from the official Flutter website ("JSON and serialization", n.d.).

The MemorEZ application uses JSON data objects and serializes JSON in model classes derived from legacy code implementation. The model classes in the MemorEZ application contain:

- a set of variable values
- a toJson()method that maps the variable values to the corresponding JSON property and returns a String version of the JSON object
- a Factory constructor that calls .fromJson() that creates an instance of the class from the map structure

Once the model class is defined an object is decoded by passing a user map to the constructor:

Map<String, dynamic> userMap = jsonDecode(jsonString);

var user = User.fromJson(userMap)

To encode the object pass it to the dart:convert library's jsonEncode() method:

String json = jsonEncode(user);

The following is a list of the model class files in the MemorEZ application:

- CalendarEvent.dart
- Help.dart
- LexResponse.dart
- NLUAction.dart
- NLUResponse.dart
- NLUState.dart
- Note.dart
- PersonalDetail.dart
- Setting.dart
- UserModel.dart

- user.dart
- Task.dart

8.2 Shared Preferences and SQLite

The caregiver mode and profile screens also use a combination of shared preferences and SQLite for storage and use of data. Information on shared preferences and sqflite plugins are taken from pub.dev.

The shared_preferences 2.0.13 plugin is used:

- https://pub.dev/packages/shared_preferences
    - Wraps platform-specific persistent storage for simple data. Supported data types are in, double, bool, String and List<String>.

The sqflite 2.0.2 plugin is used:

- https://pub.dev/packages/sqflite
    - Support transactions and batches, automatic version management during open, helpers for insert/query/update/delete queries, and DB operation executed in background thread on iOS and Android.
    - Database helpers were implemented for both caregiver mode and user profile.

The shared preferences and SQLite are used in tandem to enable caregiver mode and save/update/delete the caregiver login credentials. These two plugins are also used to save/update/delete profile information in the patient mode(ex: allergies/medications).

9 Accessibility and Internationalization

9.1 Accessibility

Information about accessibility in this section is taken from the official Flutter website ("Accessibility", n.d.).

- For Android:
    - Install the Accessibility Scanner for Android (see https://play.google.com/store/apps/details?id=com.google.android.apps.accessibility.auditor&hl=en)
    - Enable the Accessibility Scanner from **Android Settings > Accessibility > Accessibility Scanner > On**
    - Navigate to the Accessibility Scanner 'checkbox' icon button to initiate a scan
- For iOS:
    - Open the iOS folder of your Flutter app in Xcode
    - Select a Simulator as a target, and click **Run** button
    - In Xcode, select **Xcode > Open Developer Tools > Accessibility Inspector**
    - In the Accessibility Inspector, select **Inspection > Enable Point to Inspect,** and then select the various user interface elements in running Flutter app to inspect their accessibility attributes
    - In the Accessibility Inspector, select **Audit** in the toolbar, and then select **Run Audit** to get a report of potential issues
- For web:
    - Open Chrome Dev Tools (or similar tools in other browsers)
    - Inspect the HTML tree containing the ARIA attributes generated by Flutter
    - In Chrome, the "Elements" tab has an "Accessibility" sub-tab that can be used to inspect the data exported to semantics tree

9.2 Internationalization

Information about internationalization in this section are derived from the official Flutter website ("Internationalizing Flutter apps, n.d.).

To use flutter_localizations, add the package as a dependency to your pubspec.yaml file:

```
dependencies:
 flutter:
  sdk: flutter
 flutter_localizations: # Add this line
  sdk: flutter        # Add this line
```

Next run pub get packages, then import the flutter_localizations library and specify localizationDelegates and supportedLocales for MaterialApp:

```
import 'package:flutter_localizations/flutter_localizations.dart';
return const MaterialApp(
 title: 'Localizations Sample App',
 localizationsDelegates: [
  GlobalMaterialLocalizations.delegate,
  GlobalWidgetsLocalizations.delegate,
  GlobalCupertinoLocalizations.delegate,
 ],
 supportedLocales: [
  Locale('en', ''), // English, no country code
  Locale('es', ''), // Spanish, no country code
 ],
 home: MyHomePage(),
);
```

After introducing the flutter_localizations package and adding the code above, the Material and Cupertino packages should now be correctly localized in one of the 78 supported locales. Widgets should be adapted to the localized messages, along with correct left-to-right and right-to-left layout.

10 Packages

10.1 Package Usage

Information in this section about using packages is taken from the official Flutter website ("Using packages", n.d.). The following instructions use the package css_colors: as an example.

To add the package, css_colors, to an app:

- Depend on it
    - Open the pubspec.yaml file located inside the app folder and add css_colors: under dependencies.
- Install it:
    - From the terminal: Run flutter pub get.

OR

    - From Android Studio/IntelliJ: Click **Packages get** in the action ribbon at the top of pubspec.yaml.
    - From VS Code: Click **Get Packages** located in the right side of the action ribbon at the top of pubspec.yaml.
- Import it:
    - Add a corresponding import statement in the Dart code.
- Stop and restart the app, if necessary
    - If the package brings platform-specific code (Kotlin/ Java for Android, Swift/Objective-C for iOS), that code must be built into your app. Hot reload and hot restart only update the Dart code, so a full restart of the app might be required to avoid errors like MissingPluginException when using the package.

  content: https://docs.flutter.dev/development/packages-and-plugins/using-packages

10.2 Package Implementation

This section lists and describes the packages used in the MemorEZ application. Each of the packages serve a different purpose and can be seen throughout the application. Packages can be found in pub.dev, where they have been published for use. Pub.dev is a web page that contains a variety of packages and dependencies that can be used in Flutter. Table 10.2.1 shows a list of packages that were used in order to complete the MemorEZ application, as well as their description. These can be found in the pubspec.yaml file of the application.

Table 10.2.1

Packages and Description

Packages	Description
flutter_localizations	Package dependency is used to set up a localization.
speech_to_text	Package that exposes the device specific speech to text recognition ability.
translator	Package that allows the device specific translation abilities.
highlight_text	Packages that allows the ability to highlight words in a text.
avatar_glow	Package that allows the ability to animate the background.
font_awesome_flutter	Package that allows a set of Flutter icons. Includes free icons like regular, solid, and brands.
path_provider	Packages that allows finding commonly used files in the filesystem.
xml	Package that is a lightweight library for parsing, traversing, querying, transforming and building XML documents.
encrypt	Package that allows a set of high-level APIs over PointyCastle for two-way cryptography.
shared_preferences	Package that Wraps platform specific persistent storage for simple data
timeago	Package that gives a library useful for creating fuzzy timestamps.
flutter_searrch_bar	Package that allows for an expandable floating search bar.
flutter_tts	Package for Text-to-Speech.
intl	Package that contains code to deal with internationalized/localized messages, bi-directional text, other internationalization issues, and date and number formatting and parsing.
porcupine	Package that allows always listening voice enabled hearing into the application.

mobx	Package that is a state management library that simplifies the ability to connect and reactivate data with the UI of the application
flutter_mobx	Package that provides a set of Observer widgets that will automatically rebuild when the tracked observables change.
provider	Package that is a wrapper around InheritedWidgets, which makes it easier to reuse them.
quiver	Packages that is a set of utility libraries that makes using Dart libraries more convenient and easier, in addition to adding functionality.
http	Package that is a composable, multi-platform, Future-based API for HTTP requests.
json_serializable	Package that provides builders for handling a json package
amazon_cognito_identity_dart_2	Package that is the unofficial amazon cognito identity.
google_fonts	Package that allows the use of many fonts in the application
random_color	Package that allows for random colors in the application.
chat_bubbles	Package that gives the ability to use chat bubbles in the application
dcdg	Package that is a small command line that gives the ability to create a class diagram from a dart package.
sigv4	Package that is a dart library used for signing Amazon Web Services.
sqflite	Package that connects SQLite plugin to Flutter.
toast	Package that is a library for Flutter

Note. See https://github.com/umgc/spring2022/blob/development/pubspec.yaml for a complete list of packages.

10.2.1 MobX

Information in this section about using the MobX package is derived from the official MobX for Dart and Flutter website (MobX.dart team, n.d.).

The MemorEZ application makes use of observable state patterns through the use of the MobX dart package. In order to work with MobX in flutter the following packages must be installed through the pubspec.yaml file:

1. mobx: ^2.0.6+1 – under dependencies
2. flutter_mobx: ^2.0.4 – under dependencies
3. mobx_codegen: ^2.0.5+2 – under dev_dependencies
4. build_runner: ^2.1.7 – under dev_dependencies

In order to implement an observable pattern a store class must be declared. The store class must have the following boiler plate statements (using a todo app as an example):

- An import of the MobX package as:
  - import 'package:mobx/mobx.dart';
- A part declaration for code generated by MobX as:
  - part 'todo.g.dart';
- An instantiated class as:
  - class ToDo = _ToDo with _$Todo;
- Finally, an abstract class as:
  - abstract class _ToDo with Store {}

The programmer must run the flutter pub run build_runner watch –-delete-conflicting-outputs command in order for the generated code file part to be auto-generated.

The following annotations are used in the MobX store classes:

- @Observable
- @Computed
- @Action

The following is a list of the observable files in the MemorEZ application:

- CalendarObservable.dart
- ChecklistObservable.dart
- HelpObservable.dart

- MenuObservable.dart
- MicObservable.dart
- NoteObservable.dart
- NotificationObservable.dart
- OnboardObservable.dart
- ScreenNavigator.dart
- SettingObservable.dart
- TasksObservable.dart

10.2.2 i18n

Information about i18n importing in this section is derived from the pub.dev dart/flutter website (muha.dev, n.d.). Legacy implementations of i18n is derived from the previous semester of UMGC SWEN670 (fall2021.mesmerize, 2021).

To install i18n in the Flutter app use the IDE console to enter the command: $ flutter pub add i18n. Then to import i18n into a file add the statement: import 'package:i18n/i18n.dart';

In the main.dart file of the application the MaterialApp class includes the following widget:

```
localizationsDelegates: [
 i18n,
 GlobalMaterialLocalizations.delegate,
 GlobalWidgetsLocalizations.delegate,
 GLobalCupertinoLocalizations.delegate
],
```

The following languages are currently supported in the MemorEZ application:

- ar-SY (Arabic)
- en-US (English)
- es-US (Spanish)
- pt-BR (Portuguese)
- zh-CN (Chinese)

Each of the mentioned languages has a corresponding JSON file in the application that stores key:value sets of phrases for the language. Strings served to the UI should be replaced by calling I18n.of(context)! followed by a dot and the key English phrase. For example, to translate the string "Menu" in the app

replace it in the code with I18n.of(context)!.menuScreenName. Once this is done the instance of the string "Menu" should be translated when the device is switched to a supported language.

11 Services

11.1 Application Services

This section lists and describes the services incorporated into the MemorEZ application. Services focus on one distinct feature of the application and will offer a specific functionality for that feature. This, in turn, will keep the code as clean as possible by helping reduce the number of repeated code in the codebase. Table 11.1 lists the services that have been used in creation of **the MemorEZ application, as well as a description of the service.**

Table 11.1

Services and Description

Packages	Description
HelpService.dart	This service aids in holding the information for the help section and has the ability to load them from a json file.
LocaleService.dart	This service aids in the language service for the application.
NoteService.dart	This service aids in the creation and saving of notes in the application, including daily checked notes and saves it to local storage.
OnboardingService.dart	This service aids in the onboarding process of the application.
SettingService.dart	This service aids in the changing of settings in the application and saves it to local storage.
TranslationService.dart	This service aids in the translation of languages in the application.
VoiceOverTextService.dart	This service aids in the voice over text or converting text into speech portion of the application.

Note. See https://github.com/umgc/spring2022/tree/development/lib/Services for a complete list of packages.

12 AWS Lex

12.1 Overview

The information in this section partially conforms to the Medium.com website's basic AWS Lex and Flutter integration instructions (Singh, 2021).

A backend service file and a front-end UI file are required for the most basic implementation of the AWS Lex service. The LexService.dart file for the backend and the ChatBubble.dart file for the UI are the respective files in the MemorEZ application.

The http 0.13.4 dart package is used in the LexService.dart file to exchange http requests with the AWS Lex Service. A $ flutter pub add http command needs to be used to add the package to the application. The import 'package:http/http.dart' as http; declaration is used to import it.

For its http requests, the requestUrl variable is concatenated as an https encrypted link.

The import 'package:sigv4/sigv4.dart'; statement in the LexService.dart file imports the sigv4 library. The package is added to the pubspec.yaml file by adding the sigv4: 5.0.0 requirement. The library is used to construct and authenticate a client for sending an AWS service request.

The UI for the LexService file is provided by the ChatBubble.dart file. With the chat bubbles: 1.1.0 dependency in the pubspec.yaml file, the chat bubbles package is added to the application.

12.2 API Endpoint

The MemorEZ app uses a chatbot in AWS Lex. An API endpoint is used for the MemorEZ app to connect to the service. Provided below is the endpoint URL during the POST request:

https://runtime-v2-lex.us-west-2.amazonaws.com/bots/bot_id/botAliases/bot_alias/botLocales/en_US/sessions/userId/text

13 Testing

A unit test assesses a single function/method/class. The goal is to verify the accuracy of a logical unit. In order to confirm the correctness of a unit, unit tests were run on both IOS and Android devices, ensuring that the function/method/class ran on multiple devices. The following is a list of unit tests run in the creation of the MemorEZ application, and their success case:

General Tests:
1. Application Opens Up Success case: The application opens up to the Menu screen in Patient mode
2. The system successfully completes onboarding credentials Success case: The system asks for Caregiver Credentials, App Settings, and gives Patient Intro Video when first opened
3. From any page, click Menu Success case: The application will bring the user to the home screen.
4. From any page, click on Chat Success case: The application will bring you to the chat screen.
5. From any page, click on Help Success case: The application will navigate to the Help screen.
Menu Screen:
6. Click on "Tasks" Success case: The screen will navigate to the Tasks section.
7. Click on "Notes" Success case: The screen will navigate to the Notes section.
8. Click on "Calendar" Success case: The screen will navigate to the Calendar section.
9. Click on "Profile" Success case: The screen will navigate to the Profile section.
10. Click on "Settings" Success case: The screen will navigate to the Settings section.
Tasks Screen:
11. Double tap on Active Task Success case: Task moves to Completed Task.
12. Double tap on Completed Task Success case: Task moves to Active Task.
13. The user can complete a mood task Success case: The user completes the mood task and results are sent to the caregiver.
Note Screen:
14. Search for note Success case: Searched note will appear..
15. Add Note Success case: Note will be added
16. Delete Note Success case: Note is deleted.
17. Update Note Success case: Note will be updated.
Calendar Screen
18. View the Calendar in week, month, or day Success case: Ability to view the calendar in week, month, and day mode.
19. Click a date to see note/appointment

14 Exceptions

Exceptions listed below are taken from the official Flutter website (Exception class, n.d.).

Table 13.1

Exceptions

Exception	Description
DeferredLoadException	Thrown when a deferred library fails to load.
FormatException	Exception thrown when a string or some other data does not have an expected format and cannot be parsed or processed.
IOException	Base class for all IO related exceptions.
IsolateSpawnException	Thrown when an isolate cannot be created.
MissingPluginException	Thrown to indicate that a platform interaction failed to find a handling plugin.
NetworkImageLoadException	The exception thrown when the HTTP request to load a network image fails.
NullRejectionException	Exception for when the promise is rejected with a null or undefined value.
OSError	An Exception holding information about an error from the operating system.
PathException	An exception class that's thrown when a path operation is unable to be computed accurately.
PlatformException	Thrown to indicate that a platform interaction failed in the platform plugin.
RemoteException	An exception that was thrown remotely. This could be an exception thrown in a different isolate, a different process, or on an entirely different computer.
SentinelException	Thrown when an RPC response is a Sentinel.
SerializationException	Thrown to indicate a serialization error.
SourceSpanException	A class for exceptions that have source span information attached.
TestFailure	An exception thrown when a test assertion fails.
TickerCanceled	Exception thrown by Ticker objects on the TickerFuture.orCancel future when the ticker is canceled.

TimeoutException	Thrown when a scheduled timeout happens while waiting for an async result.
WebDriverException	Base exception for anything unexpected happened in Web Driver requests.

15 Appendices

15.1 Credits

The following members contributed to the development of this software:

Dr. Mir Mohammed Assadullah (Product Owner, Stakeholder)

Dr. Andrea Evangelista (Subject Matter Expert)

James Eble

Brian Avadikian

Selina Zaman

Daryle Urrea

Vanessa Stringer

Lizset Chavez Chacaltana

Robert Edwards

Joshua Fischer

Joseph Jewell

Sean LaMonica

Andrew Nicolette

Anusha Ramanan

Yusufu Sanu

Vivek Singh

Eyob Woldehana

Robert Wren

15.2 Credits to Previous Cohorts

Johnny Lockhart, Jeroen Soeurt, Michelle Monfort, Robert Wilson, Ayodeji Famudehin, Chauntika Broggin, Christian Ahmed, Mitchell Olshansky, Mod Drammeh, Nicholas Ballo, Shawn Kelly, Raul Benavides, Maddison Dunning, Alec Baileys, Benjamin Cushing, Elshaday Mesfin, Tyler Puschinsky, Michael Le, Debashis Jena, Austin Johnson, Prince Antwi Aboagye, Didimus Kimbi, Damion Sevilla, Rebecca Johnson, Addisu Worku, Matthew Setiawan, Obinna Okonkwo, Andrew Rohn, Joseph Kalfus,

Firehiwot Chari, Eskedar Endashw, Malik Webster, Leela Subramanian, Presley Muwan, Christian Cruz Jimenez, Daniel Avery, Karen Crumb, Kevin Bell, Sami Salim, Teresa Balbi, Endalkachew Girma

Section 2: Caregiver Mode

Information in this section is copied from the team RememberAll Programmer's Guide for the MemorEz application.

1 Introduction

1.1 Purpose

For the Spring 2022 semester of SWEN670 at UMGC, the class has been assigned to detail and implement an application to be used by those with Short Term Memory Loss (STML) disabilities. This document details programming for implementing the MemorEz version of the application, including tools and development software. This document may guide developers and SWEN 670 upcoming students who would like to integrate new features and functionalities into the MemorEz App.

1.2 Intended Audience

This Programmer's Guide document is intended for current and future developers of the MemorEz App, especially the 2022 spring semester and future cohorts of UMGC SWEN670 IT: Software Engineering courses.

Table 1: 2022 Spring Semester - SWEN 670 Project Stakeholders

Name	Role
Dr. Mir Assadullah	Stakeholder (Project Owner)
Roy Gordon	Stakeholder (Project Advisor)
Robert Wilson	Stakeholder (Project Advisor)
Daniel Avery	Stakeholder (Project Advisor)
James Eble	General Project Manager
Brian Avadikian	RememberAll Project Manager
Genet Asmelash	Business Analyst
Vivek Singh	Business Analyst
Lizset Chavez	Developer
Johnnie Webb	Developer
Robert Edwards	Developer
Yusufu Sanu	Developer

Eyob Woldehana	Business Analyst/ Tester

1.3 Definitions, Acronyms, and Abbreviations

- **AOT** – Ahead-of-time compiler
- **API -** Application Program Interface
- **ARIA** – Accessible Rich Internet Applications
- **Dart** – Programming language used in the Flutter framework
- **Flutter** – Framework for developing cross platform applications
- **IDE** – Integrated Development Environment
- **JIT** – Just-in-time compiler
- **MVVM**- Model View View-Mode
- **N**LU – Natural Language Understanding
- **pubspec.yaml** – Configuration file used in Flutter framework
- **RPC** – Remote Procedure Call ("Remote procedural call", n.d.)
- **SDK** - Software Development Kit
- **Sentinel -** A Sentinel is used to indicate that the normal response is not available. ("Sentinel class", n.d.)
- **STML** – Short Term Memory Loss
- **UI** – User Interface
- **VM** – Virtual Machine

1.4 References

Accessibility. (n.d.). Flutter. Retrieved February 10, 2022, from https://docs.flutter.dev/development/accessibility-and-localization/accessibility .

Build apps for any screen. (n.d.). Flutter. Retrieved February 9, 2022, from https://flutter.dev/ .

Dart overview. (n.d.). Dart. Retrieved February 9, 2022, from https://dart.dev/overview .

Exception Class. (n.d.). Flutter. Retrieved February 10, 2022, from https://api.flutter.dev/flutter/dart-core/Exception-class.html .

Flutter and the pubspec file. (n.d.). Flutter. Retrieved February 12, 2022, from https://docs.flutter.dev/development/tools/pubspec .

Flutter named routes. (n.d). Retrieved from Flutter - Named Routes - GeeksforGeeks

Internationalizing Flutter apps. (n.d.). Flutter. Retrieved February 10, 2022, from, https://docs.flutter.dev/development/accessibility-and-localization/internationalization .

NLU Tech Video Presentation. (SWEN Fall 2021). https://www.youtube.com/watch?v=d4i6bzY5mUM

Remote procedural call. (n.d.) Flutter. Retrieved February 11, 2022, from https://en.wikipedia.org/wiki/Remote_procedure_call .

Run apps on the Android Emulator. (n.d.). Android Developers. Retrieved February 9, 2022, from https://developer.android.com/studio/run/emulator#install .

Sentinel class. (n.d.). Flutter. Retrieved February 11, 2022, from https://api.flutter.dev/flutter/vm_service/Sentinel-class.html .

Supported platforms. (n.d.). Flutter. Retrieved February 10, 2022, from https://docs.flutter.dev/development/tools/sdk/release-notes/supported-platforms .

The pubspec file. (n.d.). Dart. Retrieved February 12, 2022, from https://dart.dev/tools/pub/pubspec .

Tongue Twisters (October, 2021) Programmer Guide For Natural Language Understanding. Retrieved from https://umgc-cappms.azurewebsites.net/previousprojects

Using Packages. (n.d.). Flutter. Retrieved February 10, 2022, from https://docs.flutter.dev/development/packages-and-plugins/using-packages .

Widget Catalog. (n.d.) Retrieved from Widget catalog | Flutter

2 Tools

A step by step installation procedure (including MemorEZ import and ide setup) is included in the Run Book.

2.1 IDEs

Development of the app can use Visual Studio Code or Android Studio IDEs. For our project we have used Android Studio.

2.1.1 Visual Studio Code

- Download corresponding version of Visual Studio Code from: https://code.visualstudio.com/download
- For installation on Windows follow instructions at: https://code.visualstudio.com/docs/setup/windows
- For installation on macOS follow instructions at: https://code.visualstudio.com/docs/setup/mac

2.1.2 Android Studio

2.1.2.1 Download and Installation

- Download corresponding version of Android Studio from: https://developer.android.com/studio
- For instructions on how to install Android Studio on Windows see: https://developer.android.com/studio/install
- For instructions od how to install Android Studio on macOS see: https://developer.android.com/studio/install

2.1.2.2 Setting Up an Emulator

Information in this section is taken from the Android Developer website ("Run apps on the Android Emulator," n.d.).

To install the Android Emulator in Android Studio, select the **Android Emulator** component in the **SDK Tools** tab of the **SDK Manager**. For instructions on updating tools with the SDK Manager see: https://developer.android.com/studio/intro/update#sdk-manager

2.1.2.3 Setting Up a Device

For instructions about setting up and running applications on a hardware device see:
https://developer.android.com/studio/run/device

2.1.3 Setting Up a Simulator (macOS)

2.1.3.1 Download and Install Xcode

To download and install Xcode visit the Mac App Store:
https://apps.apple.com/us/app/xcode/id497799835?mt=12

2.1.4 Setting Up an iOS Simulator

For instructions about how to set up an iOS simulator on macOS see this tutorial:
https://www.macinstruct.com/tutorials/setting-up-an-ios-simulator-on-your-mac/

2.2 GitHub

Project code is stored in a GitHub repository that can be accessed using command line tools. Alternatively, programmers can use a desktop version of GitHub that can be downloaded from https://desktop.github.com/ . For more information on GitHub implementation see section 4.

2.3 Pub.dev

The pub.dev website hosts the official repository for Dart and Flutter applications. Packages are installed using the Flutter Pub Add command. The installed packages must also be imported in the code file.

3 Frameworks and Languages

3.1 Dart

Information in this section is derived from the Dart language website ("Dart overview", n.d.).

3.1.1 Overview

Dart is a programming language utilized by the Flutter framework. For mobile applications Dart uses the Dart VM with just-in-time (JIT) compilation and an ahead-of-time (AOT) compiler. For web applications Dart uses compilers to translate Dart into JavaScript.

3.1.2 Installation

Dart is automatically downloaded with the latest versions of Flutter. For Flutter installations see section 3.2 of this document. To install Dart independently see: https://dart.dev/get-dart

3.1.3 Null Safety

A consideration when using Dart is its support for null safety. This means that variables in dart cannot have a null value unless specifically indicated. Here is an example of how to use dart to declare a variable that accepts null value:

```
int? myInt = null;
```

It is important to consider null safety when selecting packages in dart.

3.2 Flutter

Information in this section is derived from the Flutter framework website ("Build apps for any screen", n.d.).

3.2.1 Overview

Flutter is a development framework that is used to build cross platform application. For more about platforms see section 7 of this document. Flutter uses Dart for its programming language. For more information about Dart see section 2 of this document.

3.2.2 Installation

- For system requirements and installation of Flutter on Windows see: https://docs.flutter.dev/get-started/install/windows

- For system requirements and installation of Flutter on macOS see: https://docs.flutter.dev/get-started/install/macos

3.2.3 Hot Reload

Flutter provides a hot reload feature through the Dart VM. This feature allows code changes to be seen in the currently running application without rebuilding the entire app. To implement a hot reload, the flutter app must be running in debug mode. If running in an IDE the hot reload is run by clicking the **Save All** button. If running from the console then type r and press **Enter**.

4 Application Development Process

Flutter framework eased the MemorEz application process by providing packages and widgets that are made available for developers. This open-source development environment allowed the cross-platform development with pre-configured packages to be imported and used see Figure1. In addition, by integrating the android studio and running android or iOS emulators, it allows to hot reload so that the code can be modified concurrent to the application running.

Figure 1: Flutter Framework

https://www.techgropse.com/blog/wp-content/uploads/2020/02/features-of-flutter-framework.png

When a new project is created, there are default files and directories that are created Figure 2.

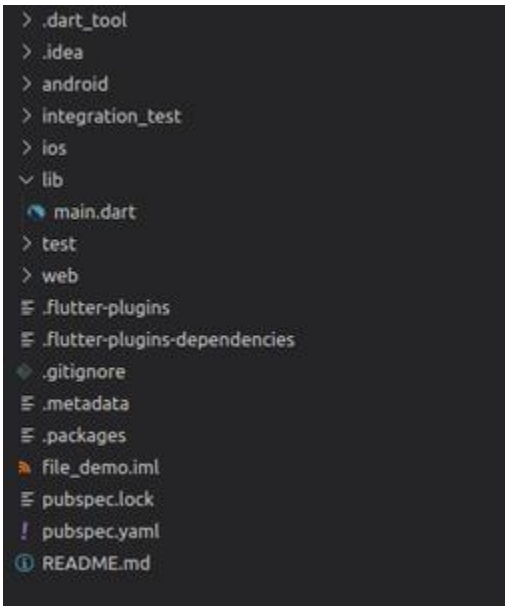


Figure 2: Flutter default directories and files

For the MemorEz app we created asset library at the root of the folder that hosts all the assets such as icons, videos and images. The lib folder contains files categorized by Utility, Services, Screen, Model, components, database handler, generated, and observables Figure 3.

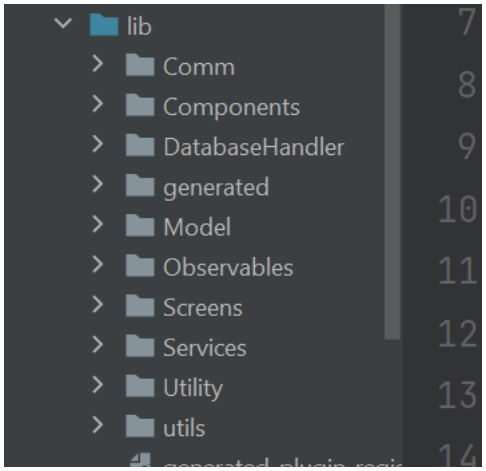


Figure 3: MemorEz lib folder

- Screens include the screen that stores onboarding pages such as login/sign-up screen, home screen and others. Utility contains all the app logic, whereas Models are the collection of data that are used in combination with widgets to complete the user interface of the app. In addition, several external Dart packages and Flutter plugins are included.
- android: This folder is provided on the creation of the project by default and contains files specific for android development.
- iOS: This folder is provided on the creation of the project by default and contains files specific for iOS development.
- assets: This folder is created to store images, videos and NLU, which contains files used in the NLU integration. Refer section 6.3 of this document for details.
- i18n : Contains files used in language translation.
- Lib:This folder is created by default when the package of created and additional files that are needed for this app development are located under this folder.
- test: files used for the unit test are located under this folder. i18nconfig.json: This is the file where configurations for language translation are located.

pubspec.yaml: All the packages and dependencies imported are located in this folder. Refer to section 6.2 for details.

5 Platforms

Information in this section is derived from the official Flutter website ("Supported platforms", n.d.).

5.1 Android

- **Supported:** API 19 & above
- **Not Supported:** Android SDK 15 & below

5.2 iOS

- **Supported:** iOS 9 & above
- **Not Supported:** iOS 8 & below

5.3 Web

- **Supported:** Chrome 84 & above
- **Supported:** Firefox 72.0 & above
- **Supported:** Safari on El Capitan & above
- **Supported:** Edge 1.2.0 & above

6 File Structure

6.1 Overview

The Model View View-Model (MVVM) pattern places the model, database, and views separately, which makes the application code easy to understand and read. This pattern follows the separation of concerns principle.

The views only display the UI, and no business logic is included in the view. Most of the UI logic is being interacted with from the View-Model via the observables. The view observes the data that the view model exposes. When the data changes, the views will be updated automatically. Part of the app project that belongs to the view is placed under the screen folder. The model is where the business logic occurs. MemorEz files are organized into separate directories for models, database, and screens, separating the application model from its view and the business logic.

6.2 pubspec.yaml

6.2.1 Description

This section describes the purpose of the pubspec.yaml in a Flutter application. Information in this section is taken from the official Flutter website ("Flutter and the pubspec file", n.d.).

The pubspec file is automatically generated when a new Flutter application is generated. This file includes the metadata about the project. The pubspec file is written in the YAML programing language. This file is where the project dependencies, development dependencies and versions are identified. This file is used by the IDE to import corresponding files for each dependency before it builds the application.

6.2.2 Fields

This section covers the available fields in the pubspec file. Information in this section is taken from the official Dart website ("The pubspec file", n.d.).

Table 2: Fields Supported in the Pubspec File

Field	Requirement/Description
name	Required for every package.
version	Required for packages that are hosted on the pub.dev site.
description	Required for packages that are hosted on the pub.dev site.
homepage	Optional. URL pointing to the package's homepage (or source code repository).
repository	Optional. URL pointing to the package's source code repository.
issue_tracker	Optional. URL pointing to an issue tracker for the package.
documentation	Optional. URL pointing to documentation for the package.
dependencies	Can be omitted if your package has no dependencies.
dev_dependencies	Can be omitted if your package has no dev dependencies.
dependency_overrides	Can be omitted if you do not need to override any dependencies.
environment	Required as of Dart 2.
executables	Optional. Used to put a package's executables on your PATH.
platforms	Optional. Used to explicitly declare supported platforms on the pub.dev site.
publish_to	Optional. Specify where to publish a package.
false_secrets	Optional. Specify files to ignore when conducting a pre-publishing search for potential leaks of secrets.

Flutter Note: Pubspecs for Flutter apps can have additional fields for configuring the environment and managing assets.

6.3 Assets and Images

Asset folder contains the Mp4 file for help, images, Natural Language Understanding (NLU) and lottie animation json. The list of all the project assets is below:

-assets/help/example_help.mp4
- assets/help/help_content.json
- assets/images/caregiver_patient_image.png
- assets/images/mic.png
- assets/images/Note.png
- assets/images/Cloud.png
- assets/images/Trigger.png
- assets/images/Setting.png
- assets/images/Help.png
- assets/images/dropdownarrow.png
- assets/NLU/lite-model_mobilebert_1_metadata_1.tflite
- assets/lottie/animation.json

- assets/lottie/done.json

7 User Interface

7.1 Widgets

MemorEz app is built using several widgets provided by Flutter framework via Widget catalog to extend stateful and stateless widgets. Both platform-specific widgets Material and Cupertino widgets have been used. Below is the list of the most commonly used widgets that have been incorporated into the app.

7.1.1 Basic and Layout Widgets

Center: To center children widgets

Text: To display and style text.

TextButton: To display flat button without showing the border outline.

Theme: To give the app consistent theme.

Image: To display an image.

Appbar: For the tab bar and navigation bars.

Column: To layout a list of child widgets in a vertical direction.

Container: To combine common painting, positioning, and sizing widgets

ElevatedButton: a button from a material design to display elevated buttons.

Icon: to display a material design icon.

Row: To layout a list of child widgets in the horizontal direction.

Scaffold: To design visual layout structure.

SizedBox: To wrap child widgets with specified height and width

Padding: to inset child widgets by a specified value

Expanded: to expand a child of row, column or flex.

MaterialPageRoute: To push a new route on the screens.

7.1.2 Other Widgets

DropDownMenuItem: used for selectlanguage.dart to select language from the menu in addition DropDownButton and BoxDecoration were used.

ListView: This scrolling widget is used to display children widgets one after another in the scroll direction.

GestureDetector: Used for noteSearchDelegate.dart, this is a widget that detects gestures. It also attempts to recognize gestures that correspond to its non-null callbacks

DataTable: used for noteTable.dart along with DataColumn and DataCell children widgets

Wrap: To display children of the widget in multiple horizontal or vertical runs.

Stepper: To display progress by prompting to the next step.

Navigator: To navigate between the routes

DateTimePicker: Used for viewnote.dart

SwichListTitle:Used for notification screen

content: https://docs.flutter.dev/development/ui/widgets

7.2 Navigation and Routing

Flutter provides the Navigator widget, and the Router widget, which can be used together.

Navigation and routing for MemorEz app leverages the materialPageRoute constructor that makes the Navigator.push/Navigator.pop available. The app used named routes to reduce code duplication.

For example: the following code on the contactcard.dart file was written file to navigate to edit profile page.

```
child: ProfileWidget(
 imagePath: user.imagePath2,
 onClicked: () {
  _conUserId.text == 'Admin'
     ? Navigator.push(context,
        MaterialPageRoute(builder: (_) => EditProfilePage()))
     : Navigator.push(context,
        MaterialPageRoute(builder: (_) => MainNavigator()));
 },
),
```

8 Data and Backend

The data storage for MemorEz is implemented using SQLite by importing the following packages from pub.dev into DBHelper.dart file.

```
import 'package:MemorEz/Model/UserModel.dart';
import 'package:path_provider/path_provider.dart';
import 'package:sqflite/sqflite.dart';
import 'package:path/path.dart';
```

Database was created leveraging the sqflite package as follows:

```
_onCreate(Database db, int intVersion) async {
  await db.execute("CREATE TABLE $Table_User ("
      " $C_UserID TEXT, "
      " $C_Phone TEXT, "
      " $C_Password TEXT, "
      " PRIMARY KEY ($C_UserID)"
      ")");
}
```

The UserModel constructor from the UserModel.dart is imported to the Helper.dart file. Using the asynchronous programming and the Future class user login info was obtained, updated and deleted.

8.1 JSON Serialization

Information on JSON serialization in this section is taken from the official Flutter website ("JSON and serialization", n.d.).

The MemorEz application uses JSON data objects and serializes JSON within model classes derived from legacy code implementation. The model classes in the MemorEz application contain:

- a set of variable values
- a toJson()method that maps the variable values to the corresponding JSON property and returns a String version of the JSON object
- a Factory constructor that calls .fromJson() that creates an instance of the class from the map structure

Once the model class is defined an object is decoded by passing a user map to the constructor:

Map<String, dynamic> userMap = jsonDecode(jsonString);

var user = User.fromJson(userMap)

To encode the object pass it to the dart:convert library's jsonEncode() method:

String json = jsonEncode(user);

The following is a list of the model class files in the MemorEz application:

- CalendarEvent.dart
- Help.dart
- LexResponse.dart
- NLUAction.dart
- NLUResponse.dart
- NLUState.dart
- Note.dart
- PersonalDetail.dart
- Setting.dart
- UserModel.dart
- user.dart

8.2 Natural Language Understanding

For the Natural Language Understanding (NLU), we have integrated Amazon Lex, which has been developed by Fall 2021 SWEN, graduate students to build Memory Magic Application. Amazon Lex is a service that is provided by Amazon Web Services (AWS) for creating conversational interfaces on web and mobile devices, among other platforms and is accessible for developers to use for creating chatbots. The detail on NLU is provided in this Youtube video Presentation.

For the complete suite of documentation on the NLU tool, please see the documentation completed by the team called Tongue Twisters from the Fall 2021 SWEN 670 Cohort. Their entire semester was dedicated to the NLU modules functionality, and as such, their documentation is a focused suite of resources around this key functionality (Tongue Twisters, 2021).

8.2.1 Connecting to LEX via NLU

In order to access the LEX services, a special set of credentials needs to be set in the project and delivered to the LEX endpoints to allow the processing of the message payload. The contents of this credentials document is passed down to each cohort through the professor/mentors of the project. Inclusion of this document is covered in detail in the Deployment Guide.

8.2.2 LEX/NLU Components and Architecture

The architecture and component description presented below is pulled directly from the tongue twisters documentation regarding the LEX/NLU.

-NOTE-

The following content is presented directly from the authors/architects of the LEX/NLU Architecture. To view this information in its original context, please see the references to the Tongue Twisters Documentation in this programmers guide.

-NOTE-

Services Supporting MemorEz NLU

- Amazon Lex V2: Application to provide natural language understanding powered by AWS.
- BERT Question and Answer: BERT, or Bidirectional Encoder Representations from Transformers is a method of pre-training language representations which obtains state-of-the-art results on a wide array of Natural Language Processing tasks.

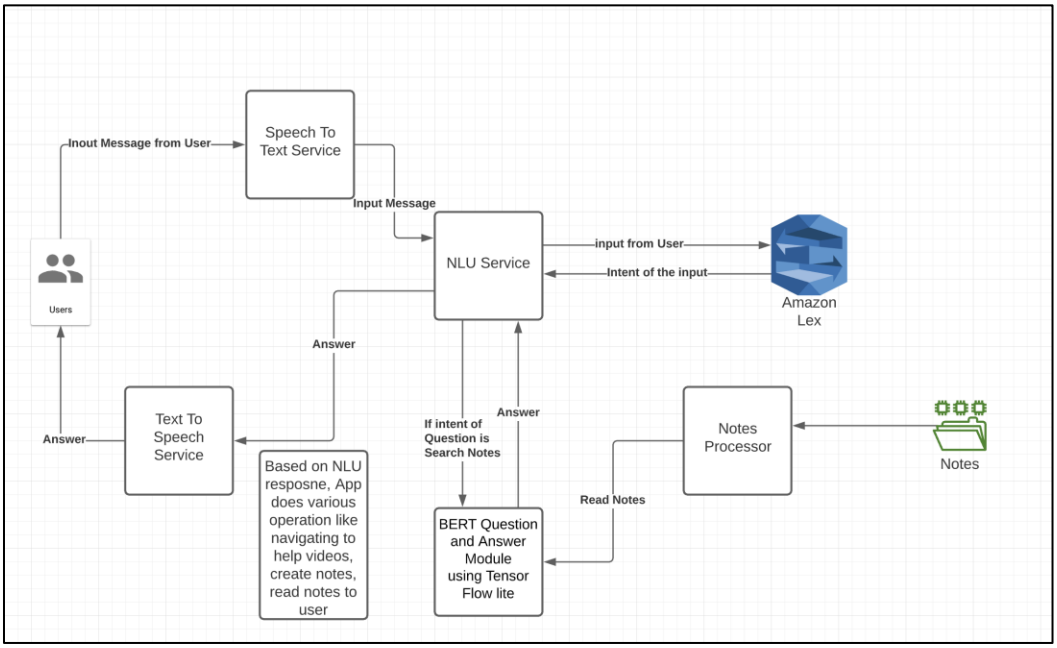


Figure 1: Image Credit- Tongue Twisters Sp.2021 NLU Architecture

The files created by the tongue twisters are still utilized in this spring 2022 project, and include the following files:

· LexService.dart -> Wrapper for Amazon Lex V2 contains method to get input message from MemoryApp and get response from Amazon Lex.

· NLULibService.dart -> acts as interface to Memory app to pass in the input message, NLULibService interacts with LexService and get the right NLUResponse based on the intent of the input message.

· NLUResponse.dart -> based on the input, NLULibService provides output in NLUResponse format.

· NLUAction.dart -> contains various action type enumerators passed as part of NLUResponse.

· NLUState.dart -> Used to determine the state of the current interaction from Memory app and NLUService, passed as part of NLUResponse.

· LexResponse.dart -> Used to deserialize the json output obtained from Amazon Lex to LexResponse object.

· BertQAService.dart -> Used as interface to BertQuestionAnswerer API, by loading lite-model_mobilebert_1_metadata_1 from assets/NLU folder.

· BertQAHelperService.dart -> Used to load all exports required for BertQuestionAnswerer API to work.

· BertNLClassifierService.dart -> Used to integrate BertNLClassifier API for Bert related models that require Wordpiece and Sentencepiece tokenizations outside the TFLite model.

· BertNLClassifierService.dart -> Used to integrate BertNLClassifier API for Bert related models that require Wordpiece and Sentencepiece tokenizations outside the TFLite model. Takes a single string as input, performs classification with the string and outputs pairs as classification results.

· BertQuestionAnswererTaskService.dart -> Used to integrate BertQuestionAnswerer API which loads a Bert model and answers questions based on the content of a given passage.

-NOTE-

The following content above is presented directly from the authors/architects of the LEX/NLU Architecture. To view this information in its original context, please see the references to the Tongue Twisters Documentation in this programmer's guide.

-NOTE-

9 Accessibility and Internationalization

9.1 Accessibility

Information about accessibility in this section is taken from the official Flutter website ("Accessibility", n.d.).

- For Android:
    - Install the Accessibility Scanner for Android (see https://play.google.com/store/apps/details?id=com.google.android.apps.accessibility.auditor&hl=en)
    - Enable the Accessibility Scanner from **Android Settings > Accessibility > Accessibility Scanner > On**
    - Navigate to the Accessibility Scanner 'checkbox' icon button to initiate a scan
- For iOS:
    - Open the iOS folder of your Flutter app in Xcode
    - Select a Simulator as a target, and click **Run** button
    - In Xcode, select **Xcode > Open Developer Tools > Accessibility Inspector**
    - In the Accessibility Inspector, select **Inspection > Enable Point to Inspect,** and then select the various user interface elements in running Flutter app to inspect their accessibility attributes
    - In the Accessibility Inspector, select **Audit** in the toolbar, and then select **Run Audit** to get a report of potential issues
- For web:
    - Open Chrome Dev Tools (or similar tools in other browsers)
    - Inspect the HTML tree containing the ARIA attributes generated by Flutter
    - In Chrome, the "Elements" tab has an "Accessibility" sub-tab that can be used to inspect the data exported to semantics tree

9.2 Internationalization

Information about internationalization in this section are derived from the official Flutter website ("Internationalizing Flutter apps, n.d.).

To use flutter_localizations, add the package as a dependency to your pubspec.yaml file:

```
dependencies:
 flutter:
  sdk: flutter
 flutter_localizations: # Add this line
  sdk: flutter        # Add this line
```

Next run pub get packages, then import the flutter_localizations library and specify localizationDelegates and supportedLocales for MaterialApp:

```
import 'package:flutter_localizations/flutter_localizations.dart';
return const MaterialApp(
 title: 'Localizations Sample App',
 localizationsDelegates: [
  GlobalMaterialLocalizations.delegate,
  GlobalWidgetsLocalizations.delegate,
  GlobalCupertinoLocalizations.delegate,
 ],
 supportedLocales: [
  Locale('en', ''), // English, no country code
  Locale('es', ''), // Spanish, no country code
 ],
 home: MyHomePage(),
);
```

After introducing the flutter_localizations package and adding the code above, the Material and Cupertino packages should now be correctly localized in one of the 78 supported locals. Widgets should be adapted to the localized messages, along with correct left-to-right and right-to-left layout.

10 Packages

10.1 Package Usage

Information in this section about using packages is taken from the official Flutter website ("Using packages", n.d.). The following instructions use the package css_colors : as an example.

To add the package, css_colors, to an app:

- Depend on it
    - Open the pubspec.yaml file located inside the app folder and add css_colors: under dependencies.
- Install it:
    - From the terminal: Run flutter pub get.

OR

    - From Android Studio/IntelliJ: Click **Packages get** in the action ribbon at the top of pubspec.yaml.
    - From VS Code: Click **Get Packages** located in the right side of the action ribbon at the top of pubspec.yaml.
- Import it:
    - Add a corresponding import statement in the Dart code.
- Stop and restart the app, if necessary
    - If the package brings platform-specific code (Kotlin/ Java for Android, Swift/Objective-C for iOS), that code must be built into your app. Hot reload and hot restart only update the Dart code, so a full restart of the app might be required to avoid errors like MissingPluginException when using the package.

  content: https://docs.flutter.dev/development/packages-and-plugins/using-packages

10.2 Package Implementations

Packages are hosted in the Dart environment to manage shared libraries and tools (Dart n.d). Several packages are imported from the Flutter pub.dev site to develop the MemorEz app. Packages version constraints with the **Caret Syntax** were used to avoid installing packages with breaking changes. Each package is integrated by running the **pub get** command. Pub gets all the packages and other packages that are needed. Pub supports two types of dependences. Packages that are also required at runtime are placed under dependencies whereas the modules which are only required during development are placed under dev_dependencies. The dev dependency was used as per Flutter's recommendation to

reduce size and get the pub run faster. For a detailed information on dependencies, refer to Package dependencies | Dart

dependencies:

```
  flutter:
    sdk: flutter
  flutter_localizations:
    sdk: flutter
  speech_to_text: ^5.4.2
  translator: 0.1.7
  highlight_text: ^1.1.0
  avatar_glow: ^2.0.1
  font_awesome_flutter: ^9.1.0
  path_provider: ^2.0.2
  xml: ^5.1.2
  encrypt: ^5.0.0
  shared_preferences: ^2.0.6
  timeago: ^3.1.0
  flutter_search_bar: ^3.0.0-dev.1
  flutter_tts: ^3.2.1
  intl: ^0.17.0
  porcupine: ^1.9.13
  mobx: ^2.0.0
  flutter_mobx: ^2.0.0
  provider: ^6.0.0
  quiver: ^3.0.1
  http: ^0.13.3
  json_serializable: ^6.1.4
  amazon_cognito_identity_dart_2: ^1.0.5
  google_fonts: ^2.1.0
  random_color: ^1.0.6-nullsafety
  chat_bubbles: ^1.1.0
  dcdg: ^4.0.1
  sigv4: ^5.0.0
  sqflite: ^2.0.2
  toast: ^0.1.5
```

dev_dependencies:

```
  flutter_test:
    sdk: flutter
  flutter_launcher_icons: ^0.9.2
  build_runner: ^2.1.7
  i18n_json: ^0.9.1
  mockito: ^5.0.17
```

10.2.1 MobX

Information in this section about using the MobX package is derived from the official MobX for Dart and Flutter website (MobX.dart team, n.d.).

The MemorEz application makes use of observable state patterns through the use of the MobX dart package. In order to work with MobX in flutter the following packages must be installed through the pubspec.yaml file:

1. mobx: ^2.0.6+1 – under dependencies
2. flutter_mobx: ^2.0.4 – under dependencies
3. mobx_codegen: ^2.0.5+2 – under dev_dependencies
4. build_runner: ^2.1.7 – under dev_dependencies

In order to implement an observable pattern a store class must be declared. The store class must have the following boiler plate statements (using a todo app as an example):

- An import of the MobX package as:
    - import 'package:mobx/mobx.dart';
- A part declaration for code generated by MobX as:
    - part 'todo.g.dart';
- An instantiated class as:
    - class ToDo = _ToDo with _$Todo;
- Finally, an abstract class as:
    - abstract class _ToDo with Store {}

The programmer must run the flutter pub run build_runner watch –-delete-conflicting-outputs command in order for the generated code file part to be auto-generated.

The following annotations are used in the MobX store classes:

- @Observable
- @Computed
- @Action

The following is a list of the observable files in the MemorEz application:

- CalendarObservable.dart
- ChecklistObservable.dart
- HelpObservable.dart

- MenuObservable.dart
- MicObservable.dart
- NoteObservable.dart
- NotificationObservable.dart
- OnboardObservable.dart
- ScreenNavigator.dart
- SettingObservable.dart
- TasksObservable.dart

10.2.2 i18n

Information about i18n importing in this section is derived from the pub.dev dart/flutter website (muha.dev, n.d.). Legacy implementations of i18n is derived from the previous semester of UMGC SWEN670 (fall2021.mesmerize, 2021).

To install i18n in the Flutter app use the IDE console to enter the command: $ flutter pub add i18n. Then to import i18n into a file add the statement: import 'package:i18n/i18n.dart';

In the main.dart file of the application the MaterialApp class includes the following widget:

```
localizationsDelegates: [
 i18n,
 GlobalMaterialLocalizations.delegate,
 GlobalWidgetsLocalizations.delegate,
 GLobalCupertinoLocalizations.delegate
],
```

The following languages are currently supported in the MemorEz application:

- ar-SY (Arabic)
- en-US (English)
- es-US (Spanish)
- pt-BR (Portuguese)
- zh-CN (Chinese)

Each of the mentioned languages has a corresponding JSON file in the application that stores key:value sets of phrases for the language. Strings served to the UI should be replaced by calling I18n.of(context)!

followed by a dot and the key English phrase. For example, to translate the string "Menu" in the app replace it in the code with I18n.of(context)!.menuScreenName. Once this is done the instance of the string "Menu" should be translated when the device is switched to a supported language.

11 Services

Services class from Flutter is created to perform service for a user such as returning actionable items. Services are used to abstract packages and group set of features. After the service package has been imported, "import package:flutter/services.dart", the following files have been created under the Services folder.

HelpService.dart

LocaleService.dart

OnboardingService.dart

NoteService.dart

SettingService.dart

TranslationService.dart

VoiceOverTextService.dart

12 Testing

Unit testing has been conducted on each user story that a developer has completed. As a developer moves the completed user story to the testing branch on Trello, members who assign themselves as testers from FlutterMind and RememberAll teams have conducted Black box unit testing of the MemorEz application see Figure 4.

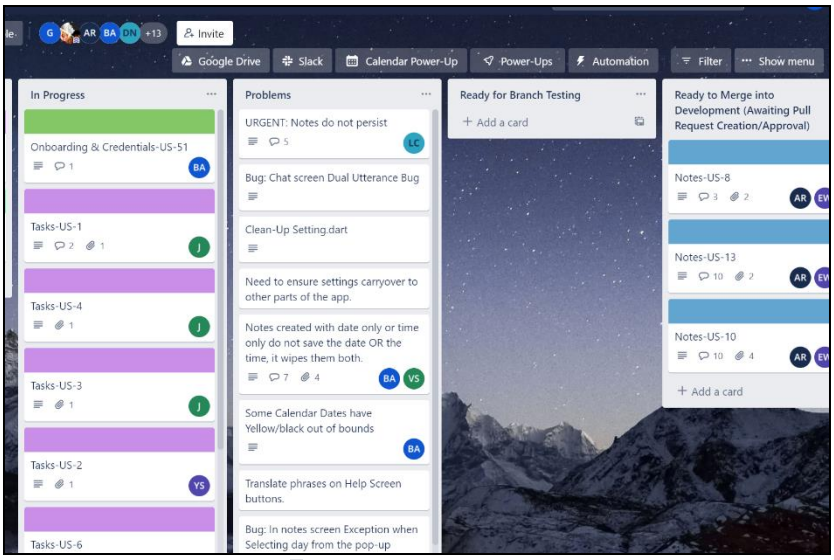


Figure 4: Trello testing branch

Each developer's updates have been tested using Android and iOS emulators as well as phones before the update was integrated into the main development branch on GitHub. A minimum of three testers has confirmed testing. The unit testing details are referenced in section 5.1 of the Deployment and Operations Guide (Runbook).

13 Exceptions

Information in this section about exceptions is taken from the official Flutter website ("Exception class", n.d.).

Table 3: Exceptions

Exceptions	Description
DeferredLoadException	Thrown when a deferred library fails to load.
FormatException	Exception thrown when a string or some other data does not have an expected format and cannot be parsed or processed.
IntegerDivisionByZeroException	Deprecated
IOException	Base class for all IO related exceptions.
IsolateSpawnException	Thrown when an isolate cannot be created.
MissingPluginException	Thrown to indicate that a platform interaction failed to find a handling plugin.
NetworkImageLoadException	The exception thrown when the HTTP request to load a network image fails.
NullRejectionException	Exception for when the promise is rejected with a null or undefined value.
OSError	An Exception holding information about an error from the operating system.
OutsideTestException	Can be omitted if you do not need to override any dependencies.
PathException	Required as of Dart 2.
PlatformException	Optional. Used to put a package's executables on your PATH.
RemoteException	Optional. Used to explicitly declare supported platforms on the pub.dev site.
RPCError	Optional. Specify where to publish a package.
OSError	Optional. Specify files to ignore when conducting a pre-publishing search for potential leaks of secrets.
OutsideTestException	
PathException	An exception class that's thrown when a path operation is unable to be computed accurately.

Exceptions	Description
PlatformException	Thrown to indicate that a platform interaction failed in the platform plugin.
RemoteException	An exception that was thrown remotely. This could be an exception thrown in a different isolate, a different process, or on an entirely different computer.
RPCError	
SentinelException	Thrown when an RPC response is a Sentinel.
SerializationException	Thrown to indicate a serialization error.
SourceSpanException	A class for exceptions that have source span information attached.
TestFailure	An exception thrown when a test assertion fails.
TickerCanceled	Exception thrown by Ticker objects on the TickerFuture.orCancel future when the ticker is canceled.
TimeoutException	Thrown when a scheduled timeout happens while waiting for an async result.
WebDriverException	Base exception for anything unexpected happened in Web Driver requests.