GSoC 2023 Project Proposal

Manash Kumar

Indian Institute of Technology (IIT-BHU), Varanasi

Rocket Launcher Visualizer / SpaceX Rocket Visualizer

Table of Content

Personal Information

Name: Manash Kumar
Department: BTech in Electrical Engineering (Semester 4)
Address: IIT (BHU) - Varanasi, Varanasi (221005), Uttar Pradesh, India
Email: manashkumar1216@gmail.com / manash.kumar.cd.eee21@iitbhu.ac.in
Contact: (+91) 7004164334
GitHub: SagittariusA11 · GitHub
LinkedIn: Manash Kumar - Flutter Developer - PetMojo | LinkedIn
Resume: ManashKumar_Resume.pdf

A dynamic BTech student at IIT (BHU) - Varanasi with passion for technology and software development. Experienced in Flutter development, proficient in Flutter, Node.js, and Python. Committed to staying ahead of technology and delivering exceptional software solutions. Interested in astronomy and skilled in cyber forensics. Aiming to continuously expand professional abilities and deliver exceptional results. I'm also very interested in astrophysics, astronomy and space related technologies from my fascination with the vastness and complexity of the universe. I'm particularly drawn to the latest discoveries and advancements in space exploration and how they shed light on the origins and evolution of our cosmos.

Project Development Experience

Tech Team, Prastuti 2023:

I have been part of a tech team that has made a mobile app for the fest using **Flutter** for frontend and **Node.js** for the backend. It has a responsive screen based on the device and uses **Google OAuth** for login. It is among my projects which are deployed and **live on PlayStore**. GitHub, PlayStore

Location Sharing App:

I have made a location sharing app using **Flutter** and **Google Maps API**. In this app one could share their location to some group of people for a preferred amount of time. The live location and history as well will be shown on **google map integrated in the app itself** using google maps API which location being marked with different colours/symbols for each person. GitHub

Winter Internship (2022) at PetMojo:

I did my winter internship at PetMojo. Taking care of their pet through a mobile app as a customer service provider.I used **Flutter** to integrate several **Rest-APIs** in the app, worked with **OneSignal**, **MixPanel**, **Exotel** and several others. It has two apps, one for customers (pet parents) and other for partners (service providers). PlayStore (Customer), (Partner).

Programming languages and other technologies:

Flutter, Kotlin, Java, Android Development, Python, HTML & CSS, JavaScript, Firebase, Node.js, MongoDB, Git, C++, Cyber Forensics, OpenCV, DSA, Android Studio

Project Description

Technical Aspect:

I am planning to build the app using flutter for frontend and calling APIs using **dart package** called "**retrofit**" for collecting data of rocket launches from APIs such as "https://www.rocketlaunch.live/api" and "https://docs.spacexdata.com/" and "https://spaceflightnewsapi.net/" this API to access different news articles and related information about the launch to show on the app also Celestrak API "https://celestrak.org/" for satellite data. We use "**google_maps_flutter**" for the live location of the user, "**google_directions_api**" for finding routes to the nearest launch site and "**geocoding**" to convert the named launch site location to coordinates. After calling the API it will return data in **JSON format** which we will convert to **KML format** file and store it locally using "**shared_preferences**" and then will send it to Liquid Galaxy for visualisation. There will be a **map screen** as well in the app to visualise the launch in the app itself if it is not connected to Liquid Galaxy. I will use **embedded code** sent to LG Rig using the KML format of **placemarkers of the launch site** with a YouTube live video link pasted by users (**of which instructions will be mentioned**) in the input text field which will telecast live launch video on LG alongside visualisation.

Accessibility Features:

Using dart packages for speech recognition "**speech_to_text**" and text-to-speech (TTS) "**flutter_tts**" as plugins in flutter apps that can be used to build accessibility features for differently abled persons in flutter apps. **Speech recognition can help users with visual**

impairments to interact with the app by converting their spoken words into text, while **TTS can assist users with hearing impairments** by reading out the text displayed on the screen **using audio files provided for the content on the screen**. These features will also be used to **read out the details of the launch** to the person. Also the app will include **multiple colour themes** including conventional dark theme and light theme to **help people with some colour blindness** so they can choose the theme which is best suited for them. It will also include **multiple languages** so that people can use their local language to use the app. For this I will use the "**flutter_translate**" dart package. **Dynamic font size** adjustable using range slider to help **users with visual impairments**.

Optional Features:

I would also like to include animations of the rocket while launching, 3D models of the different rockets on the launch site from sources such as **Sketchfab** or **GrabCAD** or **NASA 3D Resources**. And if this may raise some concern for data size being sent to LG then I would implement animation by **continuously sending KML files to LG in a uniform short time period interval** with minute changes in payload which when shown on screen will resemble continuous video or animation being played. I will use **Collada files of 3D rocket models** to show in the app and users will be able to interact with it like rotating it along different axes for better overall visualisation. And for 3D model visualisation on LG Rig I'm exploring the option of sending **collada files embedded in KML file format** but this may exceed data transfer limit to LG Rig hence for now I'm looking to **resize or compress these files** to encounter the issue but if it doesn't work then we will go with 2D model representation as explained above.

Building the app:

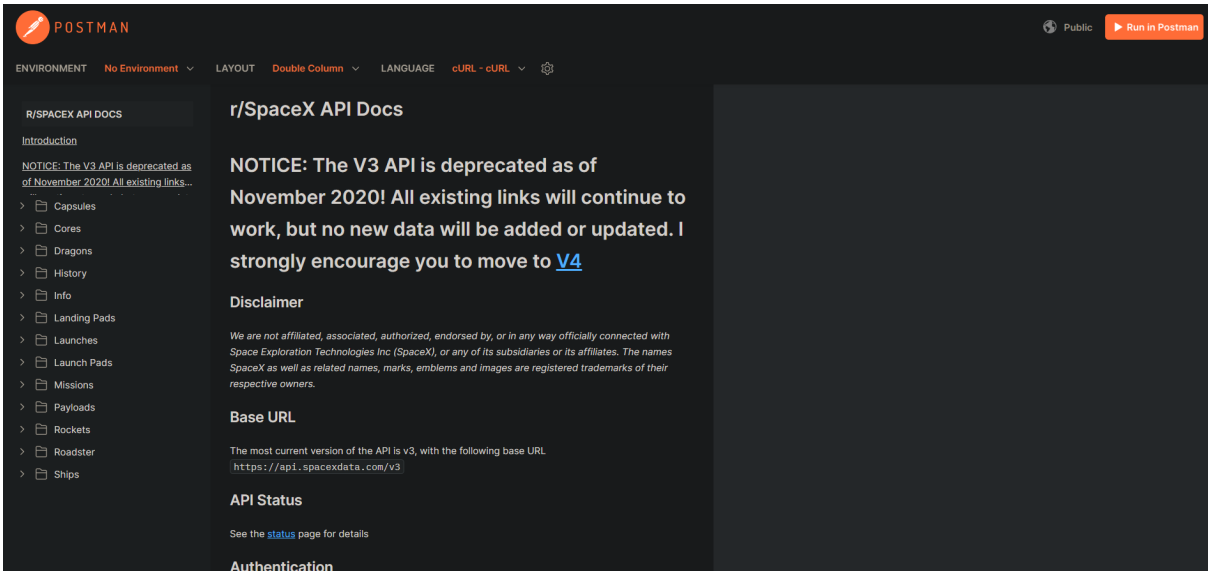
Creating the project "Rocket Launcher Visualizer" using flutter supported for **Android** with UI designed mainly for tablets of **screen size around 10 inches**. For now to depict the functioning and flow of the actual application we will use **example apps**. Let's break this process down into different steps to understand it in a chronological manner. After setting up the project these steps will be followed accordingly to meet the expected result.

Calling API:

Let's first create a project that calls the API for any one launch using SpaceX API.

API: "**https://api.spacexdata.com/v3/launches/{{flight_number}}**" where in place of **{{flight_number}}** we have to write the specific number for the flight we are searching for.

Here only for example I have used V3 of the API but in the project we will be using V4 as **V3 is deprecated as of November 2020**, Although **all existing links will continue to work, but no new data will be added or updated** as mentioned on the SpaceX API doc website. Link has been attached and a screen shot has also been attached for the same. And the link for documentation of V4 of the SpaceX API has been attached here.



Example: https://api.spacexdata.com/v3/launches/65

1. Fetching launch

```dart
import 'package:http/http.dart' as http;
import 'dart:convert' as convert;

import 'package:spacex_api_demo/models/launch.dart';

class LaunchService {

  Future<Launch> fetchLaunch() async {
    var response = await
http.get(Uri.parse('https://api.spacexdata.com/v3/launches/upcoming'));

    var json = convert.jsonDecode(response.body);
    var launch = Launch.fromJson(json[0]);
    return launch;
  }

}
```

This will return us data in JSON format for this specific flight that will look something like this of which a part is shown here.

```
{
  "flight_number": 65,
  "mission_name": "Telstar 19V",
  "mission_id": [
    "F4F83DE"
  ],
  "launch_year": "2018",
  "launch_date_unix": 1532238600,
  "launch_date_utc": "2018-07-22T05:50:00.000Z",
  "launch_date_local": "2018-07-22T01:50:00-04:00",
  "is_tentative": false,
  "tentative_max_precision": "hour",
  "tbd": false,
  "launch_window": 7200,
  "rocket": {
    "rocket_id": "falcon9",
    "rocket_name": "Falcon 9",
    "rocket_type": "FT",
```

Here we have flight number, mission_name, launch_year, launch_date, rocket_name and many more data in the response like the launch site as attached below. Click here to see the full JSON response for this API call.

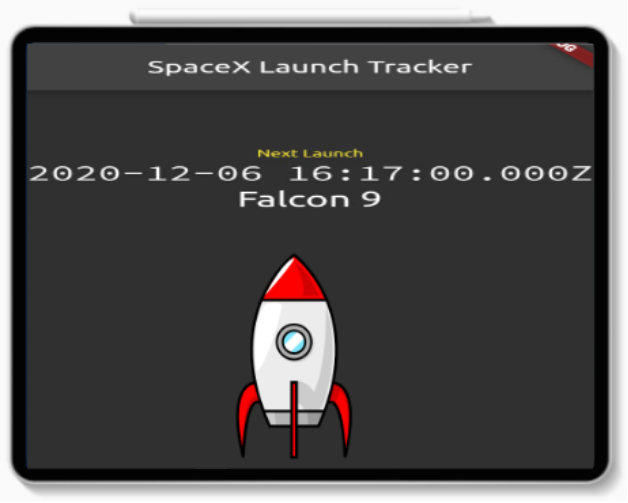
```
"launch_site": {
    "site_id": "ccafs_slc_40",
    "site_name": "CCAFS SLC 40",
    "site_name_long": "Cape Canaveral Air Force Station Space Launch
Complex 40"
  },
```

2. Extracting needed data from the response

```
class Rocket {
  final String rocketName;

  Rocket({this.rocketName});

  Rocket.fromJson(Map<dynamic, dynamic> parsedJson)
    : rocketName = parsedJson['rocket_name'];
```

```
}

class Launch {
  final DateTime launchUTC;
  final Rocket rocket;

  Launch({this.launchUTC, this.rocket});

  Launch.fromJson(Map<dynamic, dynamic> parsedJson)
    :launchUTC = DateTime.parse(parsedJson['launch_date_utc']),
    rocket = Rocket.fromJson(parsedJson['rocket']);
}
```

3. Showing them on display



We will then use these data to display on our app of which sample code and sample UI is as shown below. The screenshot for the data received from API has been shown here. The sample code is linked here.

Converting launch site name to coordinates:

To convert the launch site name that we have received in the response of the API we will use the "**geocoding**" package. First of all we need to install the package in our project which we will do by adding these lines:

```
// for installation

import 'package:geocoding/geocoding.dart';

// adding this in pubspec.yaml
dependencies:
  geocoding: ^2.1.0
```

Now we will use the "**locationFromAddress**" method of this package to convert the address "Launch Site name" to coordinates as shown below:

```
List<Location> locations = await locationFromAddress("Gronausestraat 710, Enschede");
```

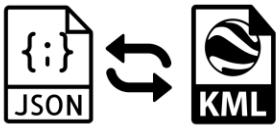
Hence we will have the required coordinates to generate the KML file.

Converting JSON data to KML:

We have fetched data from the API to our Flutter app, and we store it in **String, Map<String, dynamic> or JSON format**. For the Liquid Galaxy to be able to read this data, we must provide the **coordinates** and output the data in **KML data format** so that the Liquid Galaxy can visualise it at a later stage.

Firstly we obtain the data from the SpaceX API about the launch in **JSON** format and then we convert our **JSON data into GeoJSON** using the following Dart code that uses the "**geojson_vi**" package. We convert the **JSON file to KML data format** to visualise this data in the Liquid Galaxy.

We convert API data into both **GeoJSON and KML formats**, and later we Save them in **Read and Write files**. These modifications in the Flutter app will help us visualise the data at a later stage.



Storing of KML Data:

I plan to use the "**path_provider**" plugin to store our data in a KML file. We are also required to **read, write or update the stored KML data in the local**, which we can implement as follows:

- Create a reference to the file location:

```
Future<File> get _localFile async {
  file path - await _localPath;
  return File ('$path/countre.kml');
}
```

- **Write data** to the file:

```
Future writeCounter(int counter) async {
  final file = await _localFile;
  return file.writeAsString('$counter');
}
```

- **Read data** from the file:

```
Future<int> readCounter() async {
  try {
    final file = await _localFile;
    // Read the file
    final contents = await file.readAsString();
    return int.parse(contents);
  } catch (e) {
    // If encountering an error, return 0
    return 0;
  }
}
```

- To **save the KML file** in the downloads directory:

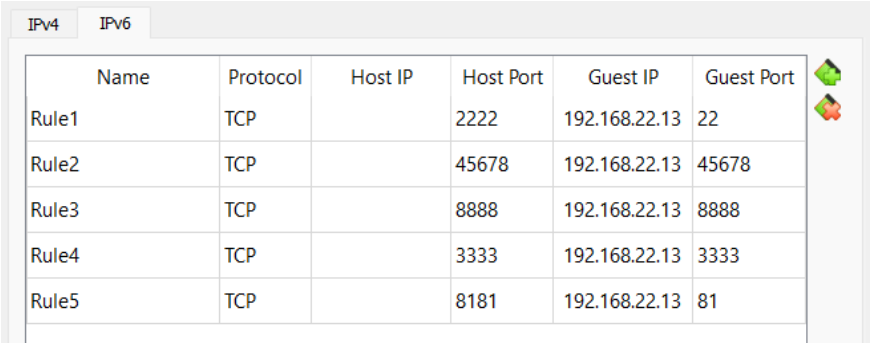
```
import 'dart:io';
import 'package:path_provider/path_provider.dart';

class KMLGenerator {
  static generateKML(data, filename) async {
    try {
      final downloadsDirectory =
          await DownloadsPathProvider.downloadsDirectory;
        var savePath = downloadsDirectory.path;
        final file = File("$savePath/$filename.kml");
        await file.writeAsString(data);
        return Future.value(file);
    } catch (e) {
      print(e);
      return Future.error(e);
    }
  }
}
```

Now our JSON data has been converted to KML files and are now stored in the device locally. Now we need to connect our app with the Liquid Galaxy and then share this file with our Liquid Galaxy to visualise it.

Connect the Flutter app to the LG rig:

To connect our Flutter app to the LG rig, we need to set up the Liquid Galaxy Rig first, then we will enable **Port Forwarding**, as shown below:

The credentials will be the same for both IPv4 and IPv6 as attached in the screenshot below.



IPv4 IPv6

Name	Protocol	Host IP	Host Port	Guest IP	Guest Port
Rule1	TCP		2222	192.168.22.13	22
Rule2	TCP		45678	192.168.22.13	45678
Rule3	TCP		8888	192.168.22.13	8888
Rule4	TCP		3333	192.168.22.13	3333
Rule5	TCP		8181	192.168.22.13	81

Port number **22** is the one Liquid Galaxy uses. Then we establish the connection through **web sockets**, for this we will use "**socket_io_client**".

Socket.IO is a library that enables low-latency, bidirectional and event-based communication between a client and a server. It helps us to **establish a connection between the Master machine of the Liquid Galaxy and our Flutter app**.

Above is the visual representation of the connection established and following is an example code for the same.

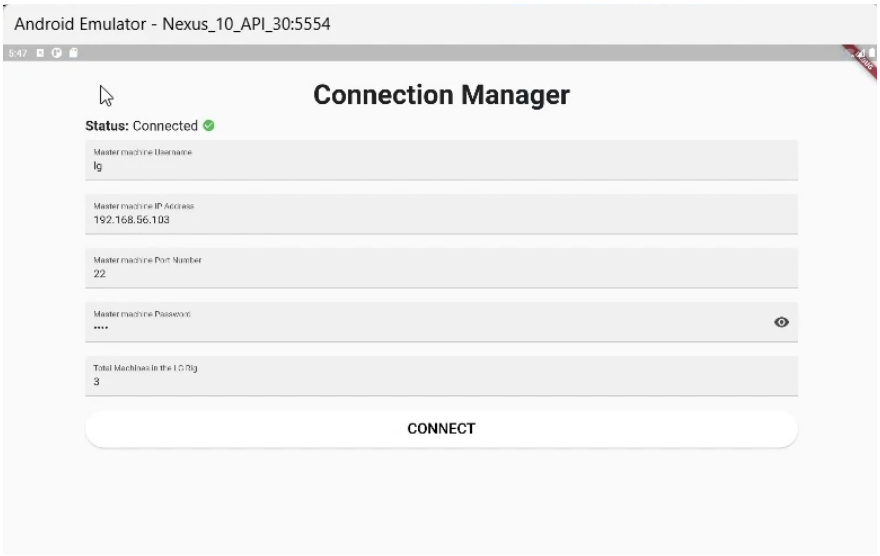
The following code uses the "**ssh2**" dart package to connect the tablet to the Liquid Galaxy Cluster.

```
connect() async {
  SharedPreferences preferences = await SharedPreferences.getInstance();
  await preferences.setString('master_ip', ipAddress.text);
  await preferences.setString('master_password', password.text);
```

```
SSHClient client = SSHClient(
    host: ipAddress.text,
```

```
    port: 22,
    username: "lg",
    passwordOrKey: password.text,
  );
  try {
    await client.connect();
    showAlertDialog('Connected!', '${ipAddress.text} Host is reachable');
    setState(() {
      connectionStatus = true;
    });
    // open logos
    await LGConnection().openDemoLogos();
    await client.disconnect();
  } catch (e) {
      showAlertDialog('Oops!',
          '${ipAddress.text} Host is not reachable. Check if the
information given is correct and if the host can be reached');
      setState(() {
        connectionStatus = false;
      });
    }
  }
```

We can see in this screenshot that the app is successfully connected to the Liquid Galaxy and hence it's showing "**connected**" beside the state.



View this video to see successful implementation of this method to connect to Liquid Galaxy.

Sending KML Data to the Liquid Galaxy:

The following code helps us **send KML data** directly from our **Flutter app to the Liquid Galaxy** after we have **successfully connected** our Tablet to the Liquid Galaxy rig:

```
  launchToLG(ProjectViewArgs args) {
  Project? p = args.project;
  // create kml based on geodata attribute
  String content = KML.buildKMLContent(args.project.geodata.markers,
    args.project.geodata.areaPolygon,
args.project.geodata.landingPoint);
  KML kml = KML(args.project.projectName, content);
  // send to LG
  LGConnection().sendToLG(kml.mount(), p).then((value) {
    buildOrbit(args);
    setState(() {
      isOpen = true;
    });
  }).catchError((onError) {
      print('oh no $onError');
      if (onError == 'nogeodata')
        showAlertDialog('No GeoData',
          'It looks like you haven\'t added any geodata to this
project.');
      showAlertDialog('Error launching!',
          'An error occurred while trying to connect to LG');
    });
  }
```

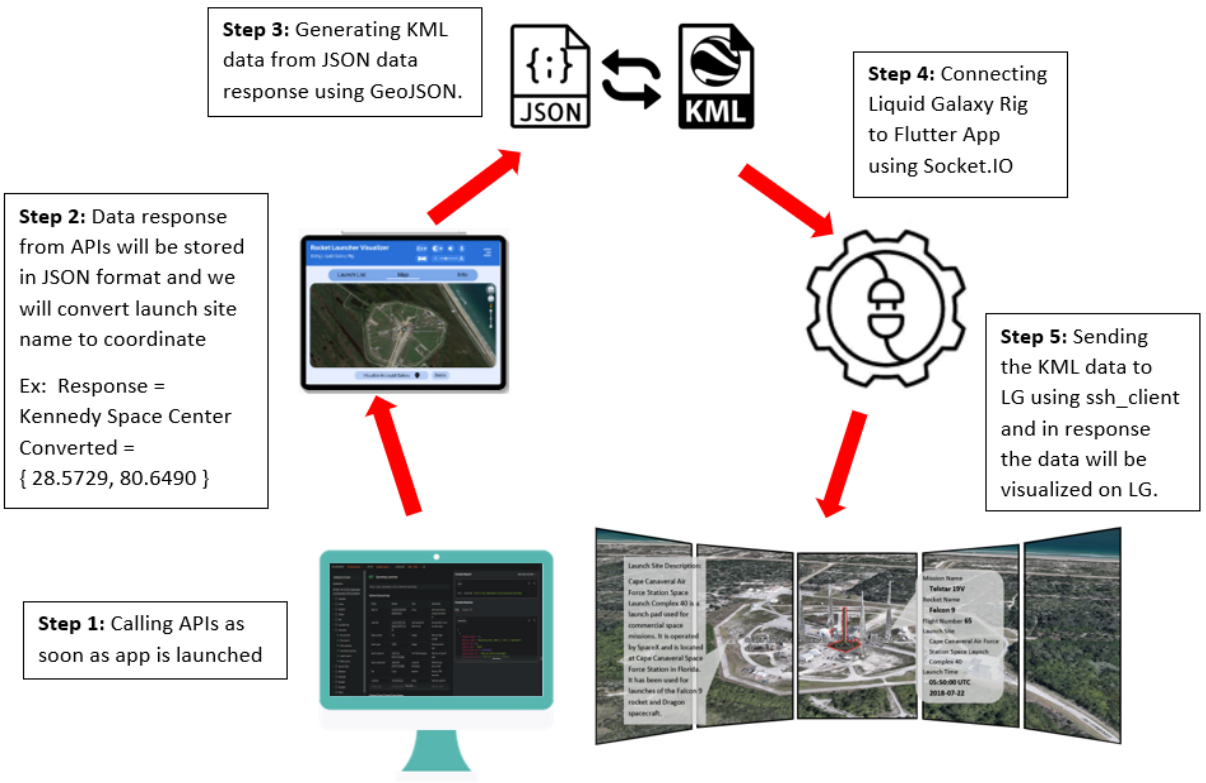
View this demonstration video to see how when the KML file is sent to Liquid galaxy and when executed it flies to the destination as in the file.

Summarised Project Description

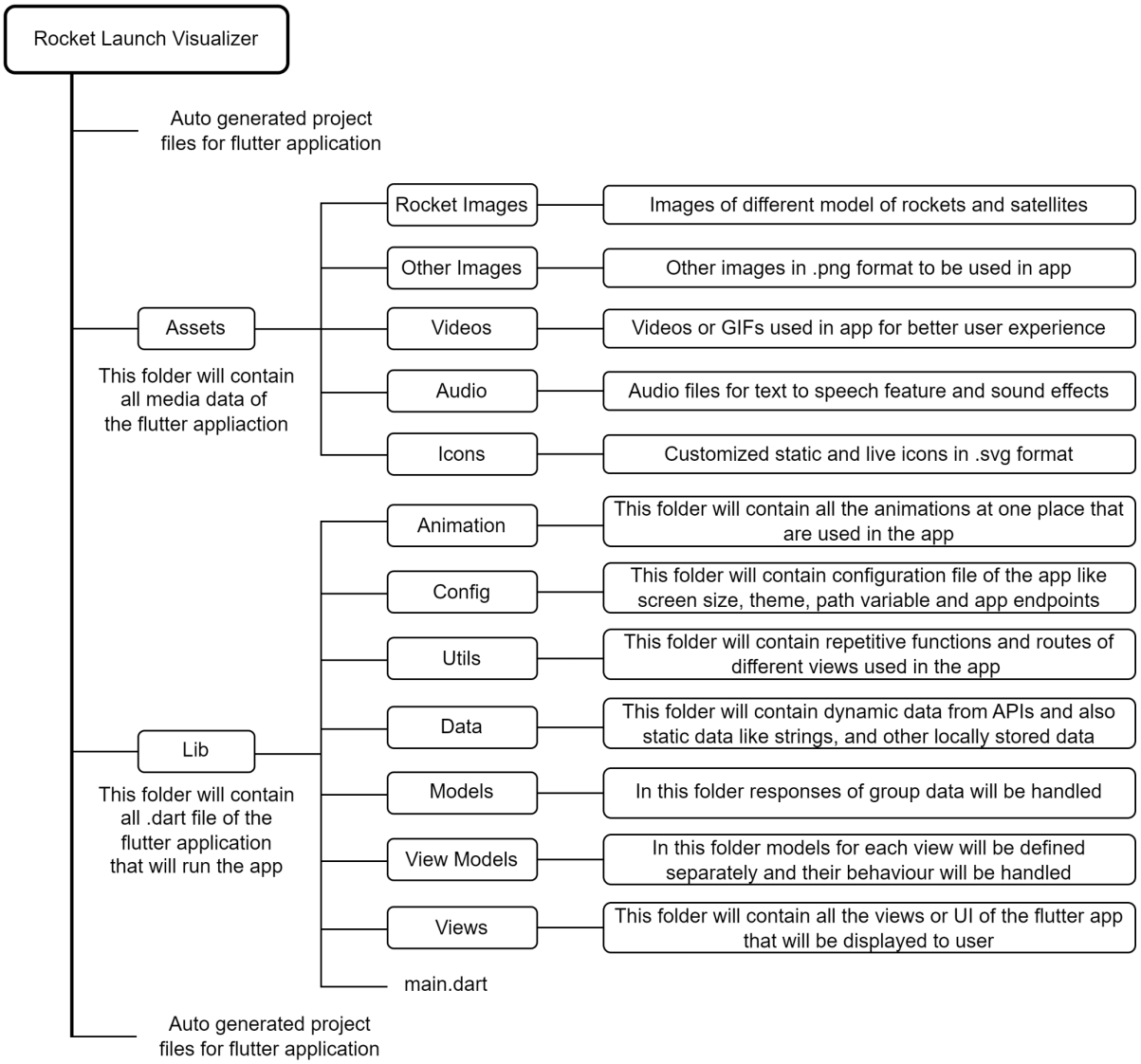
Building the App:

Brief explanation of the approach and path followed to visualiseRocket Launch data from API on Liquid Galaxy Rif using Flutter App.

1. We first called the SpaceX API using Dart Native package **retrofit** and will make **service and modules files** to handle the data received and API calling.
2. We would convert the **launch site name into geometric coordinates** using package **geocoding**.
3. Then we convert the received JSON data information into both **GeoJSON format KML data formats** using the coordinates of the launch site. We store them in the Android Tablet's memory.
4. Now we establish a connection between the Flutter app and our Liquid Galaxy rig with the help of **Socket.IO** and **ssh2**.
5. We send the **KML Data to the Liquid Galaxy**, as already explained, to get the visualisation.



Project file structure has been shown here in flow chart format

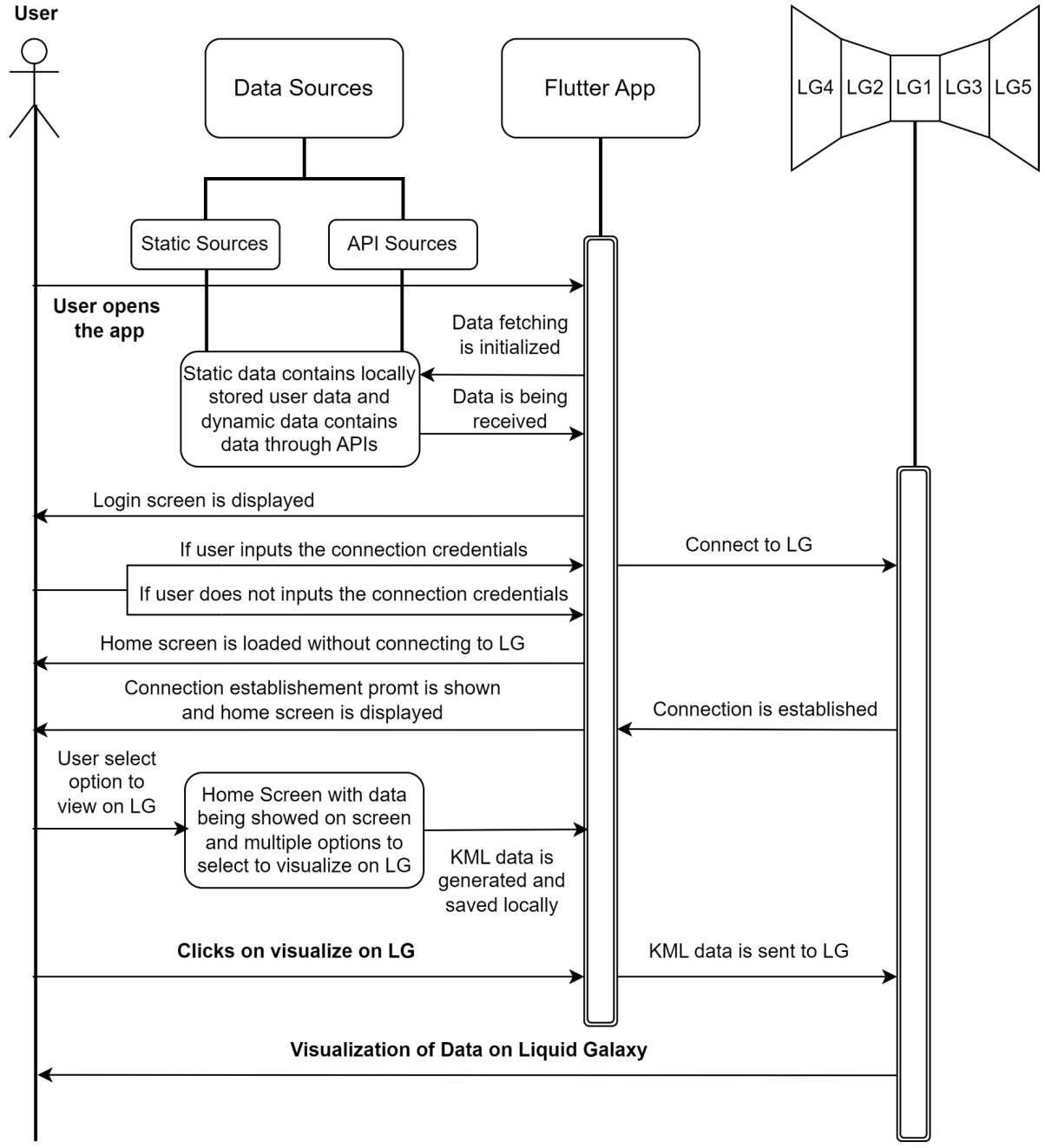


Using the App:

1. App will start with a **splash screen having animation** and showing the logo of Liquid Galaxy and the name of the project and credits to open source APIs and other public resources used.
2. After that a **login screen** will pop up in which one needs to fill **connection credentials to connect to Liquid Galaxy** and if the Rig is not available they may proceed without connecting. They can connect later on as well from inside the app.
3. After login the **main home page screen** will be there with three tabs namely "Launches", "Map", "More information" and a menu bar button on top right.
4. Menu bar will contain all essential things related to the app. It will contain settings, help, about, connection manager, and history. Each option will contain individual detail.

5. The **app bar of all the screen** will contain an icon button to toggle between **colour theme, text to speech, voice command,** and **dynamic font size**.
6. There will be a button to visualise on Liquid Galaxy after all the prerequisite tasks have been done it will show the designated place on the screen and simultaneously on Map in the app. **Users can control Liquid Galaxy using the Map as well**.
7. The map on the app will have different map modes. There will be small rocket and launch pad icons on the map and on Liquid Galaxy as well beside the location.

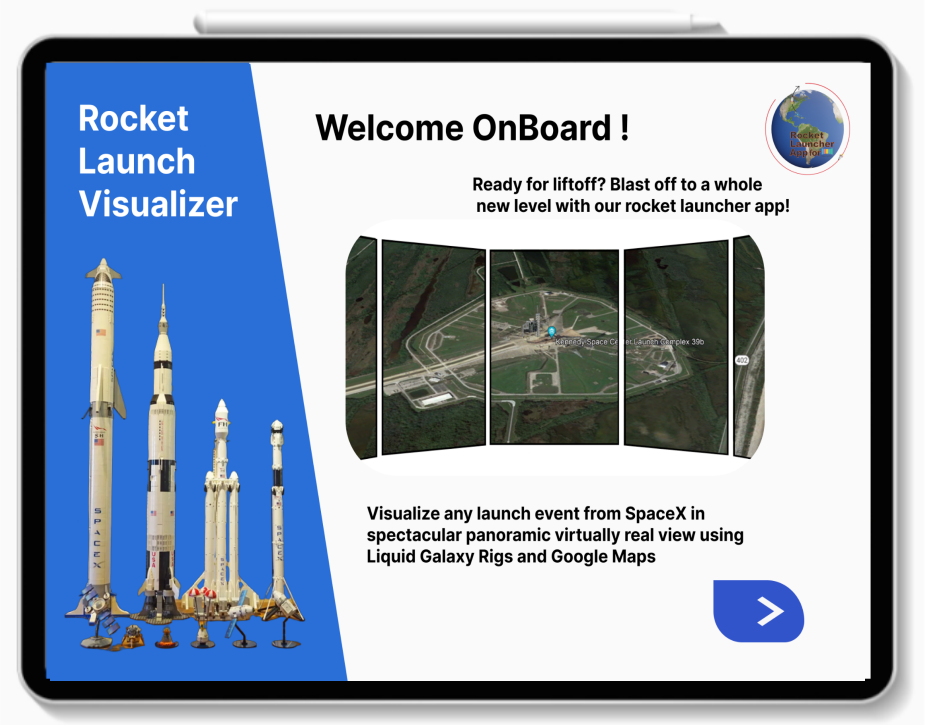
Here is the UML representation of the workflow and user interaction with the app:



Mockup UX/UI and layout of the App

Mockups of Tablet screen:

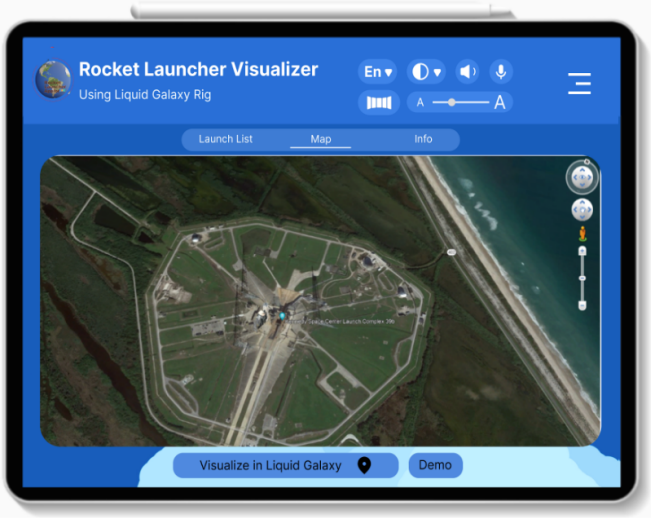
Click here for the FIGMA link of the design and click here for a **walkthrough of the UI/UX of the app** and to see how the **UI/UX will interact with the user and the user experience with UI/UX of the app**. Few example UI screenshots of Tablet screen for representation purposes are attached below with explanation of UI/UX and how it will respond to user interaction. The UI will be responsive to device screen configuration.



This is the first screen user will see **after the splash screen**. Then on clicking next the user will be directed to the home **screen** as shown below.

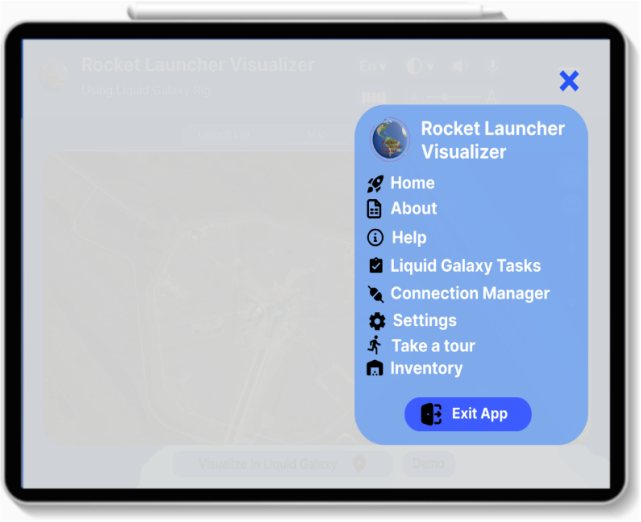
After the user has logged in they will be prompted with a **connection manager screen**. Here they will have the option **either to connect to Liquid Galaxy initially or skip the step for now**. Then they will be redirected to the home screen with three tabs to switch to.

In this screen they will have a horizontal **scroll list** for upcoming and past rocket launch events. There will be multiple **accessibility features** that can be accessed directly like **theme**, **font size**, **language**, **text to speech**, **speech to text**, and a menu bar as well. Also taping on any launch event will take the user to the more information tab for that specific event. Users will have the option to search by name, filter by custom date range and also sort the list shown.



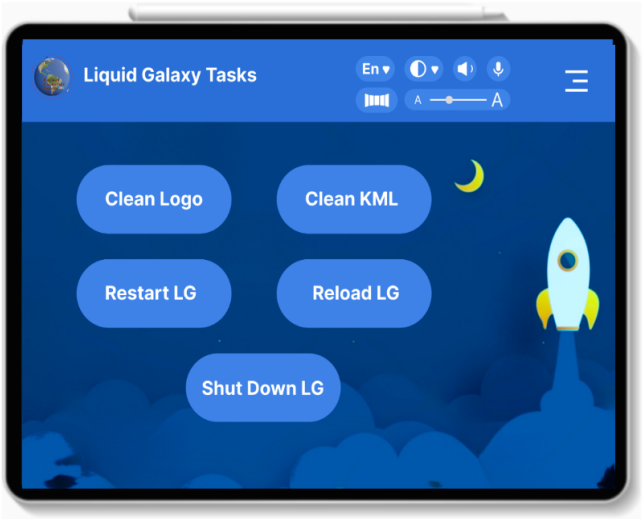
This is the **integrated google map view** in the application itself in case if users do not have access to Liquid Galaxy Rig. Users can interact with this google map widget to see the launch site for whichever launch mission they have selected.

In this tab users can see detailed information about the mission with an **image or 3D model of the rocket** and an option to watch video of launch by pressing the button at the bottom. And on the right side of the screen **description of both the mission and launch pad is mentioned**. Users can also watch the live launch on youtube by clicking on the "**Watch Video**" button.

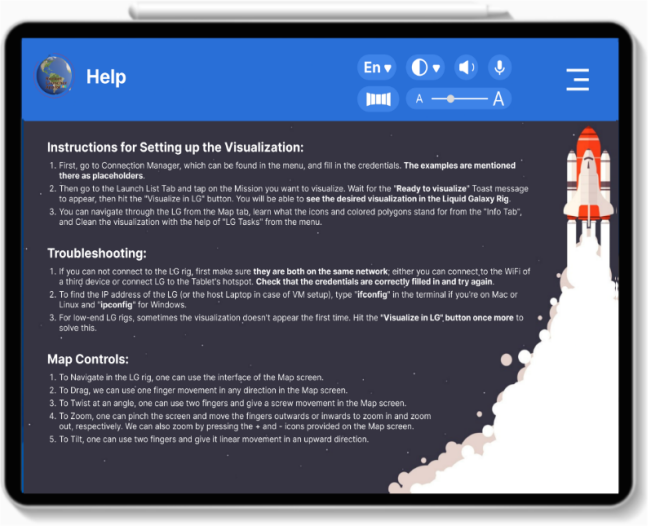


This is **how the menu bar** will look when tapped on it. It will contain a list of buttons named "**Home**": Will redirect to the rocket list page. "**About**": Will redirect to a page containing information about the app and the organisation. "**Help**": Contains instructions and assistance to understand how the app works. "**LG Tasks**": Tasks to do on LG from the application itself. "**Connection Manager**": It will redirect to the connection manager screen to connect to LG. "**Settings**": This will redirect to the settings page of the app. "**Take a tour**": This will give a walk through of the app to the user. "**Inventory**": This will redirect to an inventory screen consisting of the collection of rockets and satellites of space agencies. "**Exit App**": The app will be closed.

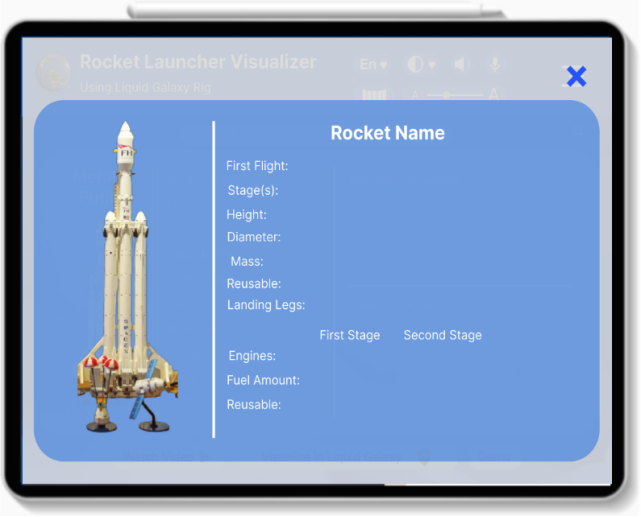
This is the settings screen. "**Language**": This will contain a language list from which users can select their preferred language. "**Colour Theme**": This will contain five modes namely light **mode, dark mode, red filter mode, green filter mode, blue filter mode**. "**Audio**": There will be two options, one will be system audio meaning animation music that has been inserted in the app, and other is **text to speech**, whether to enable it or not. "**Font Size**": It will help users to select optimal font size for them while using the app, they can adjust size by sliding the range selector as shown. "**Voice Commands**": Users can either choose to enable or disable voice commands using this setting option.



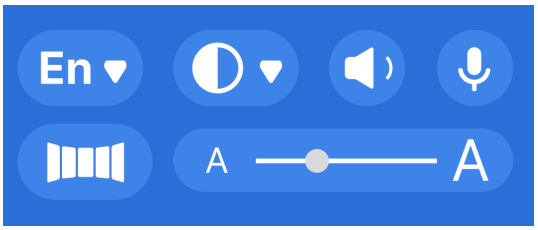
This screen is for Liquid Galaxy Tasks, meaning users can give commands or control liquid galaxy using this screen by choosing among the options provided that are namely "**Clean Logo**": This option will remove the default LG logo being shown on the left most screen of the LG Rig. "**Clean KML**": This will remove/delete the visualisation that is being displayed on LG Rig. "**Restart LG**": This will restart LG and reset it to its initial state. "**Reload LG**": This will reload LG. "**Shut Down LG**": This will shut down LG Rig.



This is the help screen with instructions written on how to use the app and also if encountered with any general problem with connectivity then how to troubleshoot it. It has step wise instructions and important things will be highlighted by making them bold text.



This screen is a pop up screen which will be shown when the user clicks on the rocket then this more information pop up of the rocket will be shown. This will also include the 3D model of the rocket as shown.



Let's go through this multi option accessibility panel on the home screen of the app. Following is the explanation of each icon in the panel:

This icon is for selecting languages.

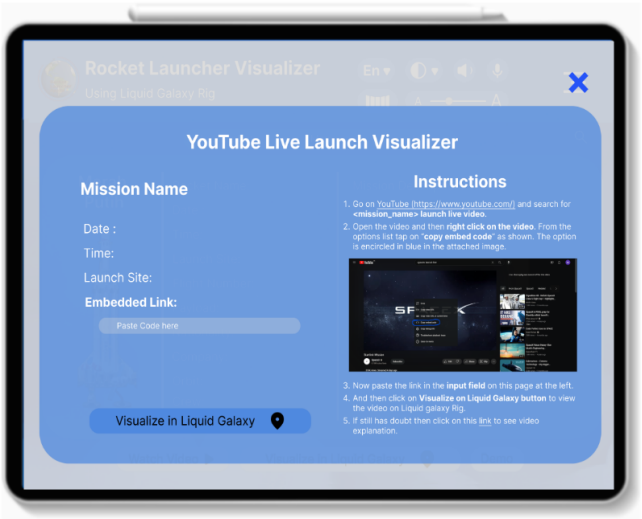
This icon is for selecting themes.

This icon is to turn on/off text to speech features in the app.

This icon is to enable/disable voice commands to control or navigate through the app.

This icon is to either connect or disconnect from the LG Rig. If not connected it will redirect the user to the connection manager screen.

This icon is to select font size of the content displayed on screen



This is the view of the YouTube Launch Visualizer dialog box. On the left we have some detail of the missions and launch and an input **field where embed code is to be pasted** which will be then **sent with KML placemarker** file to launch YouTube Live video on Liquid Galaxy itself. And on the right side there will be **instructions on how to paste embed code in a step wise manner**. And for further convenience a **video explanation link will also be there**.

For the app for different functionality cases in the app, like for toast messages for showing status of connection when trying to connect or selecting date range from calendar to access custom data set. For the error message I would like to add some **GIFs and animation** with these messages to make it more live. Also there will be **transition animation of different types** to make the UI/UX more live and user interactive.

This is the inventory screen of the app using horizontal scroll view where users can assess information about **the rockets and satellites inventory** of the space agencies. Here users will have the option to **search by name**, and many filter options as well for rockets and satellites such as results **only from SpaceX, NASA, others or all** and for satellites apart from agencies such as **Starlink satellite or Geocentric satellite**. Tapping on any tab of result will show a detailed view of the selected object as already depicted earlier.

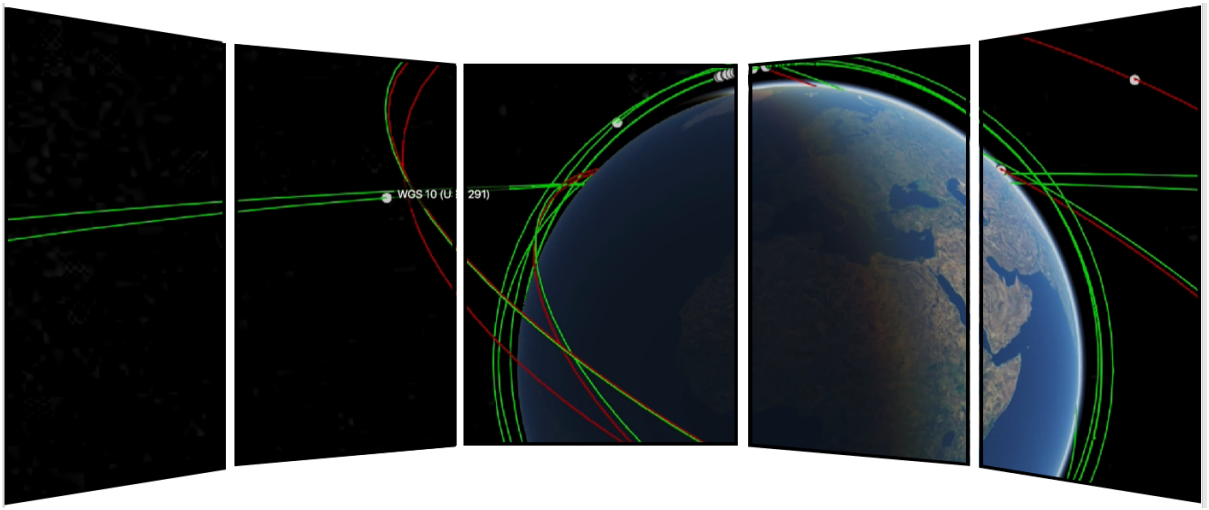
A demo of user interaction flow with the app and a walkthrough of the UI/UX of the app has been attached [here] as mentioned above. In the demo I have screen recorded how the user will interact with the app and basic UX of the app.

Mockups of Liquid Galaxy Rig:

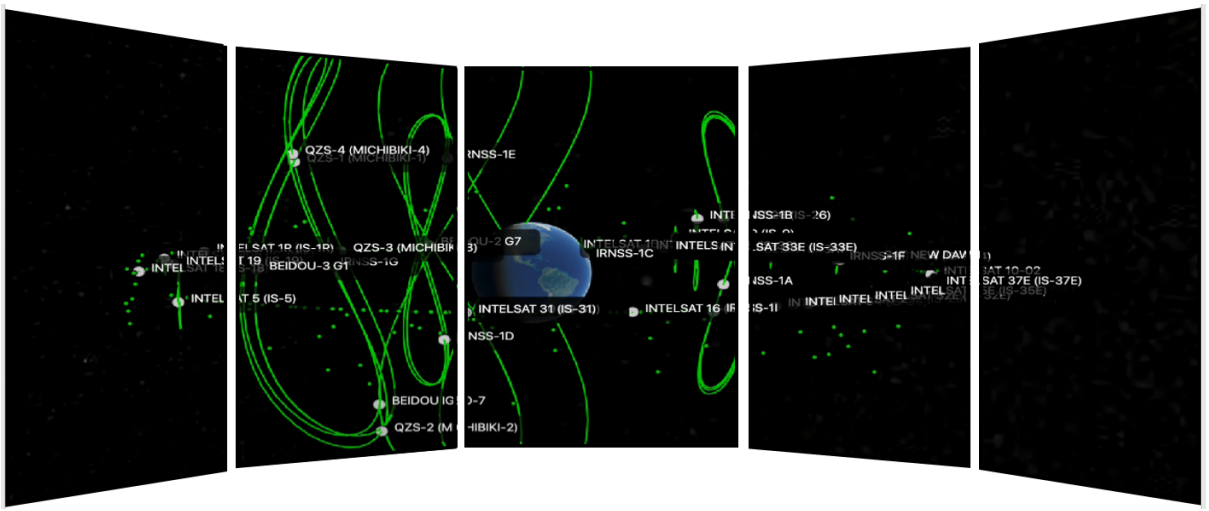
1. Visualizing Launch Pads across the globe and **filter the data based on location or country or agency**. With a custom dialog box showing details about them like **number of launches, name, location, latest mission name, upcoming mission if any, and operational since**. User can select any site and he will have more details about it. As the SpaceX API doesn't provide all data I will **either web scrape them using the "web_scraper" package or will add them locally** by name which will be displayed together with the data from SpaceX API.

2. Visualizing Landing Pads across the globe and **filter the data based on location**, these are generally or mostly of SpaceX hence no agency filter is required. With a custom dialog box showing details about them like **number of launches, name, location, latest mission name, upcoming mission if any**. User can select any site and he will have more details about it.

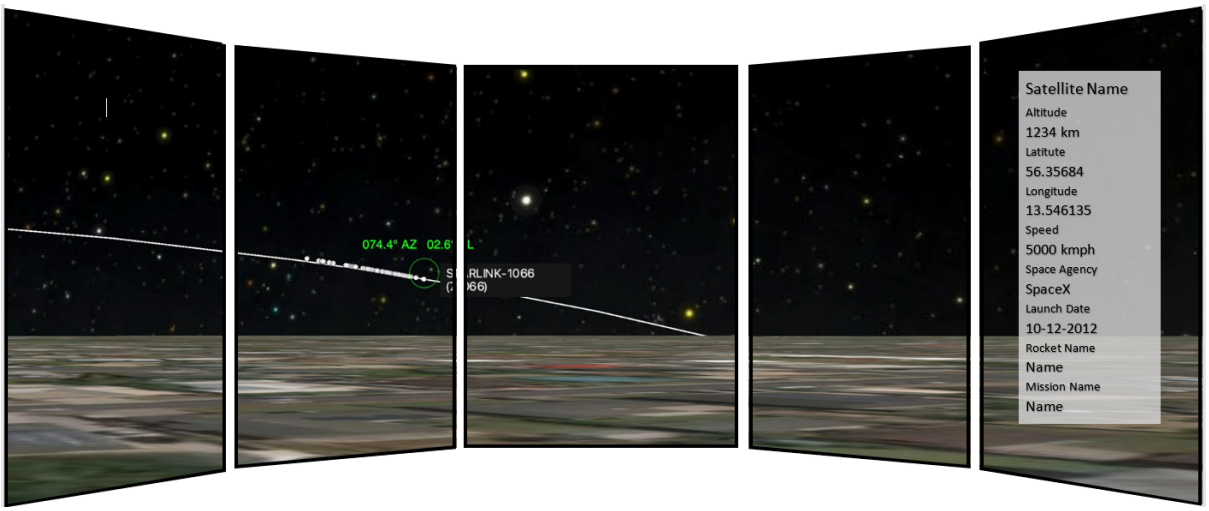
3. Visualising **starlink satellites using SpaceX API** query method for satellites in **v4** with a dialog box beside them showing details about them like **epoch time, speed, altitude, name, mission name**, and other satellites also using Celestrack API or other APIs. One can visualise the launch which had deployed these satellites in orbit. Users can select any satellite to view details about the specific satellite. **OPTIONAL FEATURES** I'm trying to add is **visualising the orbit path of the satellite** around the Earth and if possible an **earth surface point of view of the satellite** as seen from any specific location on earth when the satellite is passing by above that point.



4. There will be a special filter for "**Geocentric Satellites**". These satellites are those satellites which have a revolution speed of one revolution per day around the earth keeping it above one point of the earth at all the times. Which people generally thinks but **IS NOT THE CASE** and I want to specifically represent this case using **Celestrack API** on Liquid Galaxy Rig and it is expected to look as follows. And the people will know the actual movement of the geocentric satellites, which is not very intuitive to them.



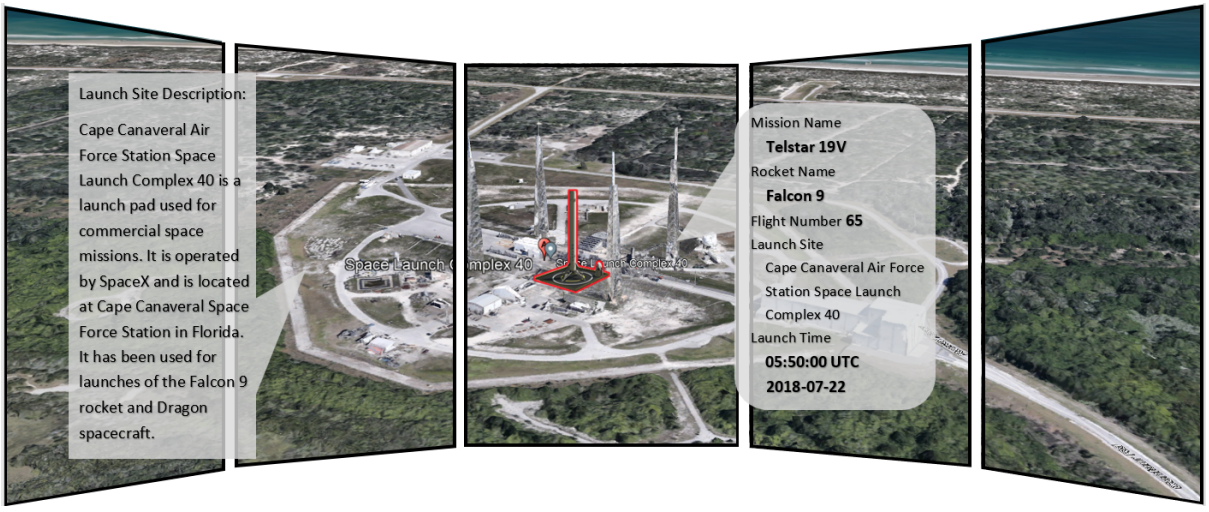
5. As I have mentioned about the **earth point of view** of the satellite visualisation when it is passing by that location. For this feature I would set the view from the earth surface that is a street **view feature of google earth** and will **disable the atmosphere** so to have a better view of the satellite as it passes by. A dialog box will be there displaying the details of the satellite such as **epoch time, speed, altitude, name, mission name**.



Use Cases

Space Science Education

The Rocket Launcher Visualizer or SpaceX Launch Visualizer app can be a formidable tool in space science education for school students when deployed on Liquid Galaxy. This app can allow students to visualise the launch site, its associated information, and trajectory. Through this immersive experience, students can learn about the location and environmental factors that affect rocket launches. They can also gain knowledge about the physics behind space travel and the features of the Falcon 9 rocket, which is frequently used by SpaceX for its launches. When used with Liquid Galaxy, the app can provide an even more captivating experience and stimulate the curiosity of the next generation of space explorers.



Here in the attached image we can see the launch site with details related to things shown on screen.

Space Science Awareness and Exploration

The Rocket Launcher Visualizer or SpaceX Launch Visualizer app can be an instrumental tool in creating awareness about space science and promoting exploration. When used with Liquid Galaxy, the app can provide a captivating experience that allows users to visualise the launch site, its relevant information, and trajectory. This can help to spark interest and curiosity in space science and inspire more people to learn about the physics and engineering behind space travel. Ultimately, the Rocket Launcher Visualizer or SpaceX

Launch Visualizer app can contribute significantly to raising awareness about space science and inspiring the next generation of space explorers.

This is another image of another launch site, users can opt whether to show details or not.

Rocket launch live visualisation

There will be a launch animation depicting the expected trajectory of the rocket. The link for the **animation demo** is attached here.

The above animation will be facilitated by **sending KML files with little time intervals**, maybe 4-5 frames per second, with **changing data for visualisation of the rocket and surrounding** (zooming out simultaneously) to make it more realistic. Then continuous rendering of these changing frames at some interval will make it look like live animation. And also it will show live launch video of the rocket from YouTube on the right side of the screen **using url embed technique in KML files**, and flight information on the left side of the screen such as **altitude, speed, time elapsed after launch**.

Linked Technologies

Retrofit (retrofit):

It provides an easy-to-use, type-safe API for defining API endpoints and expected response types, as well as supporting features such as **request interception, URL manipulation, and response caching**.

```
// for installation

import 'package:retrofit/retrofit.dart';

// adding this in pubspec.yaml
dependencies:
  retrofit: ^4.0.1
```

Here is the example of how we will use it to get info from Rest API.

Google Maps API (google_maps_flutter):

The Google Maps API allows users to easily **integrate Google Maps into their applications**. With the Google Maps API, one can **display maps, add markers, and draw routes**. One of the key features of the API is the ability to access **live location data and directions**, which can be used to provide real-time information to users.

```
// for installation

import 'package:google_maps_flutter/google_maps_flutter.dart';

// adding this in pubspec.yaml
dependencies:
  google_maps_flutter: ^2.2.5
```

Here is the example of how we add a google maps widget in our app. To use this package we need to made following changes in our project:

1. Set the minSdkVersion in **android/app/build.gradle**:

```
android {
    defaultConfig {
        minSdkVersion 20
    }
}
```

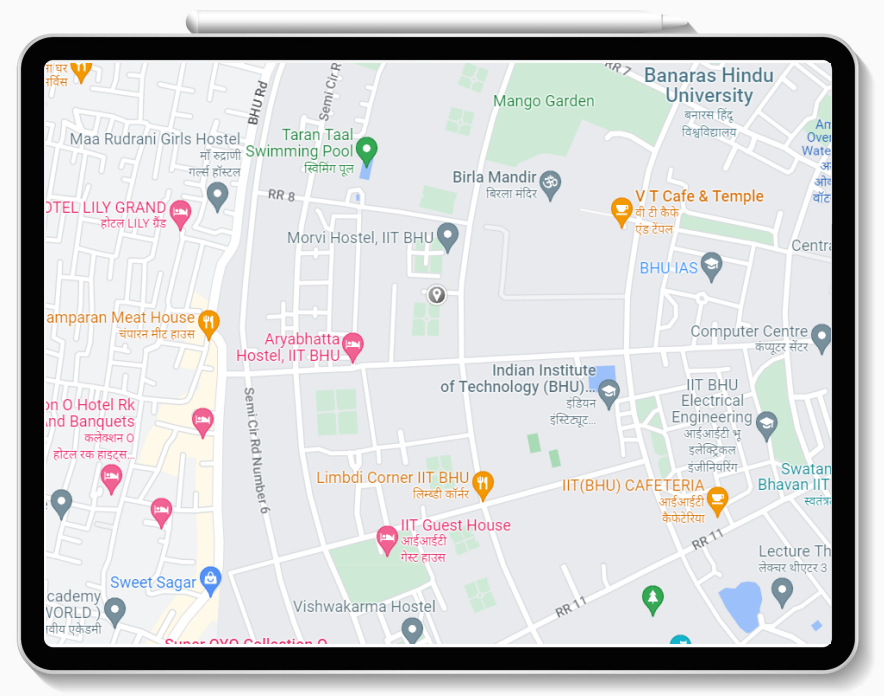
2. Specify your API key in the application manifest for Android **android/app/src/main/AndroidManifest.xml**:

```
<manifest ...
  <application ...
    <meta-data android:name="com.google.android.geo.API_KEY"
               android:value="YOUR KEY HERE"/>
```

3. To set up, specify your API key in the application delegate for iOS **ios/Runner/AppDelegate.m**:

```
#include "AppDelegate.h"
#include "GeneratedPluginRegistrant.h"
#import "GoogleMaps/GoogleMaps.h"

@implementation AppDelegate

- (BOOL)application:(UIApplication *)application
    didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {
  [GMSServices provideAPIKey:@"YOUR KEY HERE"];
  [GeneratedPluginRegistrant registerWithRegistry:self];
  return [super application:application
didFinishLaunchingWithOptions:launchOptions];
}
@end
```

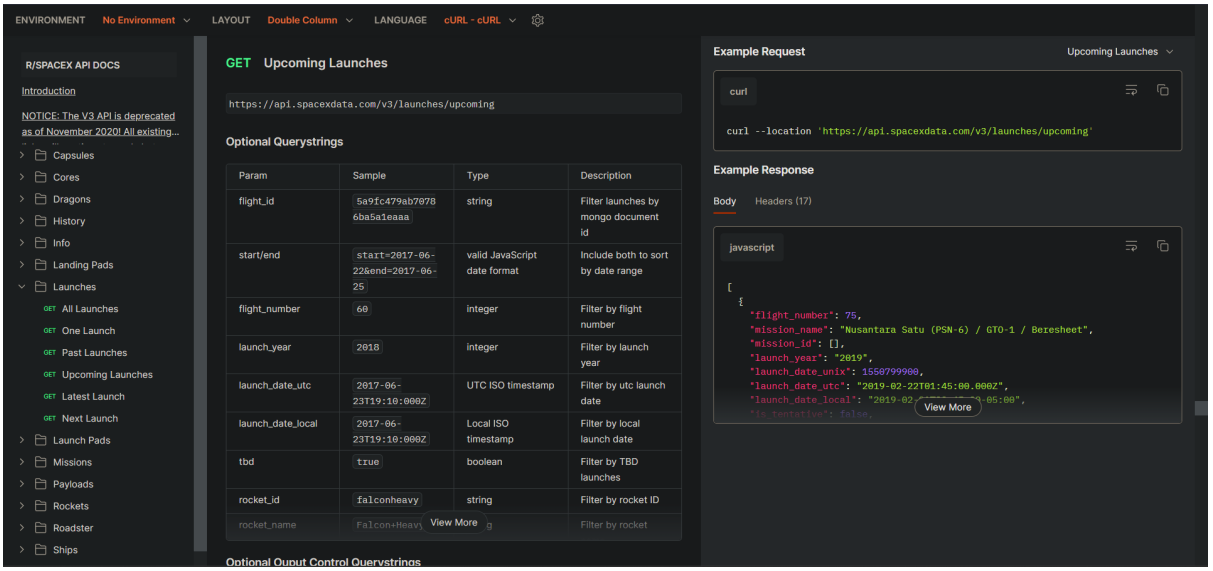
Here is the screenshot attached.



SpaceX API:

API: https://docs.spacexdata.com/

The SpaceX API is a **free, public REST API** that provides access to data about SpaceX's launches, rockets, capsules, and other related data. With the SpaceX API, one can retrieve detailed information about each **launch, including launch date, launch site, mission details, and rocket specifications**.



Speech recognition and text to speech (flutter_tts, speech_to_text):

The **speech_to_text** package enables developers to easily integrate speech recognition into their applications, allowing users to **dictate text or commands using speech**. The **flutter_tts** package, on the other hand, allows users to have **text read aloud to them**. It supports a variety of languages and allows developers to **customise the speed, pitch, and volume of the generated speech**.

```
// for installation

import 'package:flutter_tts/flutter_tts.dart';
import 'package:speech_to_text/speech_to_text.dart';

// adding this in pubspec.yaml
dependencies:
  speech_to_text: ^6.1.1
  flutter_tts: ^3.6.3
```

Example app for both is as follows:

flutter_tts: Here is the example code.

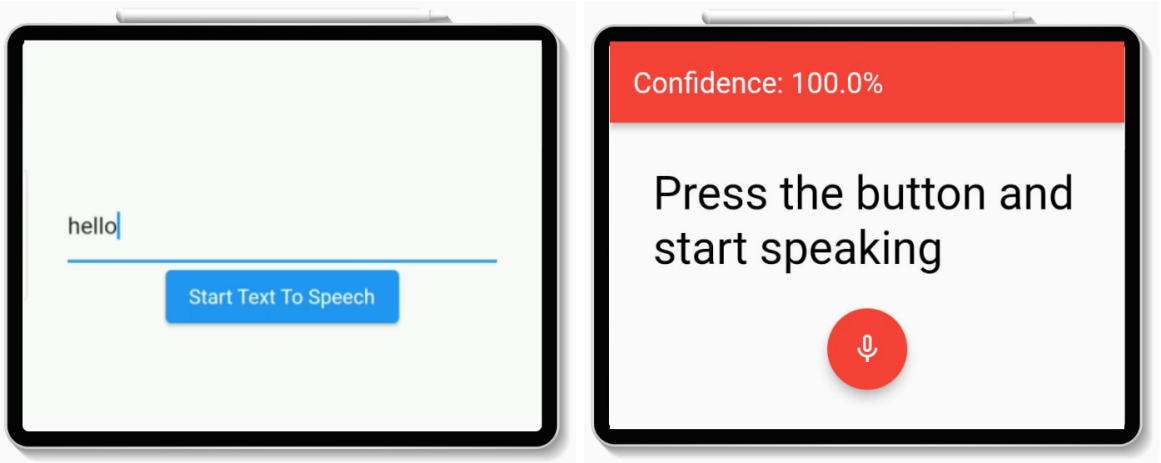
speech_to_text:

We also need to add permission in **android/app/src/main/AndroidManifest.xml**:

```
<uses-permission android:name="android.permission.RECORD_AUDIO"/>
<uses-permission android:name="android.permission.INTERNET"/>
```

Here is an example code and attached below is the screenshot of the demo app with a textfield and a button to say out loud whatever is written in the textfield.

Following are the video links of demonstrations of both the packages in action. Click on these links to view the video demonstration. flutter_tts, speech_to_text.



GeoJSON format (geojson_vi):

GeoJSON is an open standard format used for **encoding geographic data structures**. It is based on the JSON format and is used to represent **geographical features such as points, lines, and polygons, as well as their associated properties**. In Flutter, the **geojson_vi** package is a useful tool for working with GeoJSON data.

```
// for installation

import 'package:geojson_vi/geojson_vi.dart';

// adding this in pubspec.yaml
dependencies:
  geojson_vi: ^2.0.7
```

Here is an example code of implementation.

Port Forwarding and Socket.IO (socket_io_client):

Port forwarding is a technique used to allow **external devices to connect to a device on a local network**, such as a server or computer. Socket.IO is a library that enables **real-time, bidirectional communication between the client and the server**, and it includes **support for WebSocket, HTTP, and other transport protocols**.

```
// for installation

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

```
// adding this in pubspec.yaml
dependencies:
  socket_io_client: ^2.0.1
```

SSH (ssh2):

SSH client is a protocol used for **secure remote access and communication with a server**. These packages provide the functionality to execute **remote commands, transfer files, and manage SSH connections**.

```
// for installation

import 'package:ssh2/ssh2.dart';

// adding this in pubspec.yaml
dependencies:
  ssh2: ^2.2.3
```

Here is an example code of implementation.

Timeline

Bonding Period (4 May - 25 May):

Week 1 (4 May - 11 May)	I plan to thoroughly understand the open-source code available to create and send **KML data from the Flutter app to the Liquid Galaxy Rig** and also will call a few APIs with already built in code to **test run time and scope for optimization**. I will explore different types of data that can be sent through KML like **3D modes, animation, data fields, and url embedding**.
Week 2 (12 May - 18 May)	For the Flutter app, I will look into the code needed to establish a connection between the app and the Master machine and will **optimise already built code for establishing connection**. Will try for smooth running of applications even in slow speed data connection.
Week 3 (19 May - 25 May)	In this period I will **start to code the UI/UX of the Flutter App** and will test it for different use case scenarios with local and hard coded data.

First Working Period (26 May - 9 July):

Week 4 - 6 (26 May - 15 June)	In this period I will **complete code the UI/UX of the Flutter App** and will test it for different use case scenarios with local and hard coded data. Will first code the main screens of the app and then will code the **drawer, pop ups, snackbar, prompts and error view GIFs and animation**. And will make its **UI responsive** for different sizes of tablets.
Week 7 - 9 (16 June - 9 July)	Once the frontend will be ready I will start integrating the **SpaceX API** as well to make the **data flow pipeline fully functional**. By the end of this week I will try to make the app functional with all features except for the accessibility features.

Midterm evaluation (10 July - 14 July)

Second Working Period (14 July - 20 Aug):

Week 10 - 12 (14 July - 27 July)	I will begin with and complete the Flutter code to **convert the JSON / String data we get into GeoJSON and KML formats** in our Flutter App and Store them in the Downloads directory and **send data to Liquid Galaxy**. I will make the **template for different types of data** that we will be sending to Liquid Galaxy and then will just need to **insert the required parameters to generate those files** hence optimising the app run time.
Week 13 - 15 (28 July - 20 Aug)	In this period I will **enable the accessibility features** of the app such as text to speech, voice commands, colour theme, dynamic font size and then the **other functionality such as search, filter, sort** for better surfing through launch data. And test the app for different types of use cases and implement some extra features as they come up later or as suggested by the mentor.

Final Week (21 Aug - 28 Aug):

Final Week (21 Aug - 28 Aug)	I will run **final checks for all functionality** from calling APIs, displaying data on Flutter app then visualising it on in-app google maps and on Liquid Galaxy, bugs in UX/UX, accessibility and search functionality. Will **write full documentation of the app and contribution guide**. Finally, we can publish our Flutter app in the Play Store.

Post GSoC

As a dedicated contributor to the Liquid Galaxy organisation's GSoC’23 project named “Rocket Launcher Visualizer / SpaceX Rocket Visualizer”, **my commitment to the project would not end with GSoC'23**. I believe that an open-source project's success depends on its community's contributions and continuous improvement. Hence, **I would like to continue to contribute to the project even after GSoC by staying in touch with the community**, attending meetings, and collaborating with other contributors. I would ensure that the **codebase remains up-to-date** with the latest technologies, fix bugs, and add new features as per the project's needs. My goal would be to keep the app functional and user-friendly, while also ensuring that it adheres to industry standards and best practices. To accomplish this, **I would work closely with the project's maintainers and other contributors, providing guidance and support wherever needed**. Additionally, I would be available to update, debug, and maintain the code of the app as required, ensuring that it remains robust and reliable.

In summary, my commitment to the Liquid Galaxy organisation's GSoC’23 project named “Rocket Launcher Visualizer / SpaceX Rocket Visualizer” would be long-term, and I would continue to contribute actively and collaboratively even after GSoC'23.

Conclusion

In conclusion, the Liquid Galaxy Flutter app for project “Rocket Launcher Visualizer / SpaceX Rocket Visualizer”, will successfully achieve its goal of visualising rocket launches on the Liquid Galaxy Rig by end of this project duration. The app, built using Flutter and integrated with the SpaceX API, provides users with a comprehensive and user-friendly experience. With the implementation of various Dart packages and Flutter plugins, the app has been optimised for performance and accessibility, ensuring a smooth experience for all users. Additionally, the app features a wide range of customization options, allowing users to tailor their experience to their specific needs. Also I’m willing for a long-term commitment to ensure that the app remains up-to-date with the latest technologies and adheres to industry standards and best practices.

Overall, the Liquid Galaxy Flutter app project represents a significant contribution to the field of space and rocket launch visualisation and serves as an example of the collaborative nature of open-source collaboration and will be immensely useful for educational purposes and will ignite a light of interest and enthusiasm towards space technology and liquid galaxy in students.