GSOC 2022 project proposal

Yash Raj Bharti

Indian Institute of Technology (IIT-BHU), Varanasi

Lucknow, India

350h La Palma Volcano Eruption Tracking Tool

Table of Contents

350h La Palma Volcano Eruption Tracking Tool

Personal information

I am pursuing my bachelor's degree in Metallurgical Engineering at IIT-BHU, Varanasi. I did my schooling at City Montessori School (CMS) in Lucknow, India.

Project Development Experiences

Executive, Technical Team (website):

I was a part of the web development team responsible for creating the frontend of the **Kashiyatra'20 Website** (an annual socio-cultural festival of IIT-BHU) that uses the parallax.js javascript library to create a gyroscopic UI.

Club of Programmers (COPS), IIT-BHU:

Organized tasks and projects under COPS related to WebVR Virtual Tours, WebAR holograms and image targets made using the 8th Wall.

Project WebAR:

Built using AR.js developed an Augmented Reality Experience that lets users interact with 3D models and guides them through the whole workflow from advertising to purchasing, creating a unique WebAR shopping experience on the scan of a QR code. Github link | Deployed Site

Winter Internship (2021) at SupportGenie:

Using MindAR and 8thWall, I built two varieties of Annotation Creators that allow owners to annotate over real hardware devices and end-users to view those

annotations. Mind AR approach creates a target.mind file from an image of the actual hardware and annotations are created and viewed in AR. The 8th wall has a dedicated dashboard where users can upload the image targets; the annotations are made on that 2D image and viewed in AR. Both these approaches aim to improve the onboarding experience of users. **[Video Demonstration]**

Technical Team Head, Kashiyatra:

Head and Mentor for the **Kashiyatra'22** website, which uses the A-frame framework to build an animated and interactive 3D model in the website. Supervised the technical team executives to understand Three.js and A-frame and create stunning UI designs.

Programming languages and other technologies:

HTML, CSS, Javascript, basic Node.js (to establish a server), Java, Flutter (Dart), Three.js, A-frame, AR.js, Web Scraping, Twilio API.

[RESUME LINK]

Project Description

I plan to develop a web scraping tool made with the help of the **Dart native package** called **web_scraper 0.1.4** that collects all the data related to La Palma Volcano Eruption from various sites, such as:

- COPERNICUS
- GDACS
- GDACS | Affected Areas
- IGN.ES
- World 3D Map
- Comet.nerc
- NASA
- Volcano Discovery

  And many others.

This web scraping happens in our **Flutter app**, and then we convert the scraped JSON/String data to **KML format** and keep both formats. We show the data as readable text in the **Info tab** of our **Flutter app**. We **segregate the KML data** into the various tracks to get a more neat visualization of the different Map data, such as Lava flow, Historic flow, Earthquake Magnitude, Detected events, Located events, Affected areas, etc. And we list these categories in the **Tracks tab** of our Flutter app. Lastly, we write the code to **send KML data** from our Flutter app to the Liquid Galaxy to **visualize** this data on the Liquid Galaxy rig.

Beautiful Soup

Beautiful Soup is a Python library for pulling data out of HTML and XML files. This library makes it easy to **scrape information** from web pages. The one we shall use to pull down relevant data easily is the **Dart native package** inspired by this library called **web_scraper 0.1.4**. To know how it works, let's first look into some python code to **understand web scraping**.

A basic demonstration of **Beautiful Soup Python Library** is as follows:

1. Starting from scratch, we first cd into an empty folder and execute the following commands:

```
yashraj@Yashs-MacBook-Air ~ % cd /Users/yashraj/Desktop/Beautiful\ Soup
yashraj@Yashs-MacBook-Air Beautiful Soup % python3 -m venv venv
yashraj@Yashs-MacBook-Air Beautiful Soup % . venv/bin/activate
(venv) yashraj@Yashs-MacBook-Air Beautiful Soup %
```

2. Now we can install all the **dependencies** by running the following two commands:

```
(venv) yashraj@Yashs-MacBook-Air Beautiful Soup % pip3 install beautifulsoup4
Requirement already satisfied: beautifulsoup4 in ./venv/lib/python3.9/site-packa
ges (4.10.0)
Requirement already satisfied: soupsieve>1.2 in ./venv/lib/python3.9/site-packag
es (from beautifulsoup4) (2.3.1)
WARNING: You are using pip version 21.3.1; however, version 22.0.4 is available.
You should consider upgrading via the '/Users/yashraj/Desktop/Beautiful Soup/ven
v/bin/python3.9 -m pip install --upgrade pip' command.
(venv) yashraj@Yashs-MacBook-Air Beautiful Soup % pip3 install lxml
Requirement already satisfied: lxml in ./venv/lib/python3.9/site-packages (4.8.0
)
WARNING: You are using pip version 21.3.1; however, version 22.0.4 is available.
You should consider upgrading via the '/Users/yashraj/Desktop/Beautiful Soup/ven
v/bin/python3.9 -m pip install --upgrade pip' command.
(venv) yashraj@Yashs-MacBook-Air Beautiful Soup % pip3 install requests
Collecting requests
  Using cached requests-2.27.1-py2.py3-none-any.whl (63 kB)
Collecting urllib3<1.27,>=1.21.1
  Using cached urllib3-1.26.9-py2.py3-none-any.whl (138 kB)
```

3. The website that we shall be scraping is [the Global Volcanism Program](volcano.si.edu/volcano.cfm?vn=383010) which has a track of the Seismic activities of the Volcano in La Palma from 2017 to 2021.

Table 1. Precursor seismicity episodes at La Palma between October 2017 and late June 2021 were all located in the southern third of the island. Magnitude is reported by IGN as MbLg, or the magnitude from the amplitude of the Lg phase, similar to the local Richter magnitude. Data courtesy of IGN Noticias.

Date	Detected Events	Located Events	Magnitude Range (mbLg)	Depth Range (km)
07-09 Oct 2017	--	68	Less than 1.5-2.7	12-35
13-14 Oct 2017	352	44	Less than 1.5-2.1	15-22
10-14 Feb 2018	--	85	1.8-2.6	25-30
12 Feb 2019	--	7	0.7-1.1	15
24 Jul-02 Aug 2020	682	160	1.2-2.5	16-39
23-26 Dec 2020	602	126	1.3-2.3	30
31 Jan 2021	--	27	1.2-2.5	10-29
25 Jun 2021	80	12	Less than 2.2	18-34

We inspect the website and write an appropriate code that can scrape data from the website every time we want to update information.

4. We run the code by simply executing the command "**python3 filename.py**" and we get all the desired data stored in `.txt` files.

Web Scraper 0.1.4

The goal is to create the above tool to scrape relevant data from the web but **inside our Flutter app**, making everything happen on the tablet. We use **web_scraper 0.1.4,** a basic web scraper implementation to scrape HTML elements from a web page.

A simple usage example is as follows:

```
void fetchProducts() async {
  // Loads web page and downloads into local state of library
  if (await webScraper
      .loadWebPage('/test-sites/e-commerce/allinone/computers/laptops'))
{
    setState(() {
```

```
      // getElement takes the address of html tag/element and attributes
you want to scrape from website
      // it will return the attributes in the same order passed
      productNames = webScraper.getElement(
          'div.thumbnail > div.caption > h4 > a.title', ['href',
'title']);
      productDescriptions = webScraper.getElement(
          'div.thumbnail > div.caption > p.description', ['class']);
    });
  }
}
```

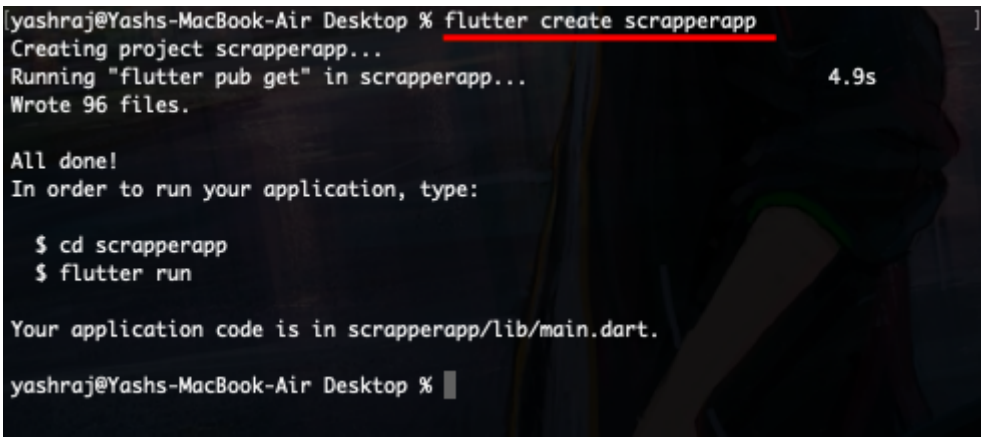
So, our **Web Scraping tool** is successfully set up and running. Now we move to show how to fetch this data through our **Flutter App** with a sample demo.

The Flutter App Demonstration

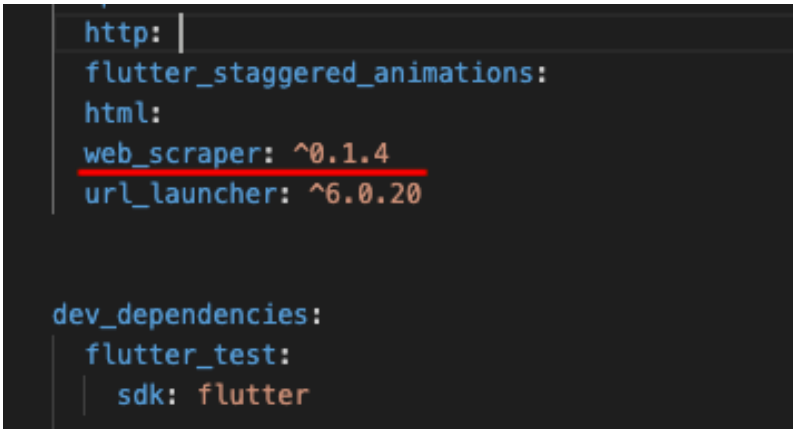
Flutter is an open-source **UI software development kit** created by Google.

We have shown a **sample use case** for collecting data from relevant sites with some code snippets. Now it's time we write a code for a **working demonstration of web scraping** with Dart.

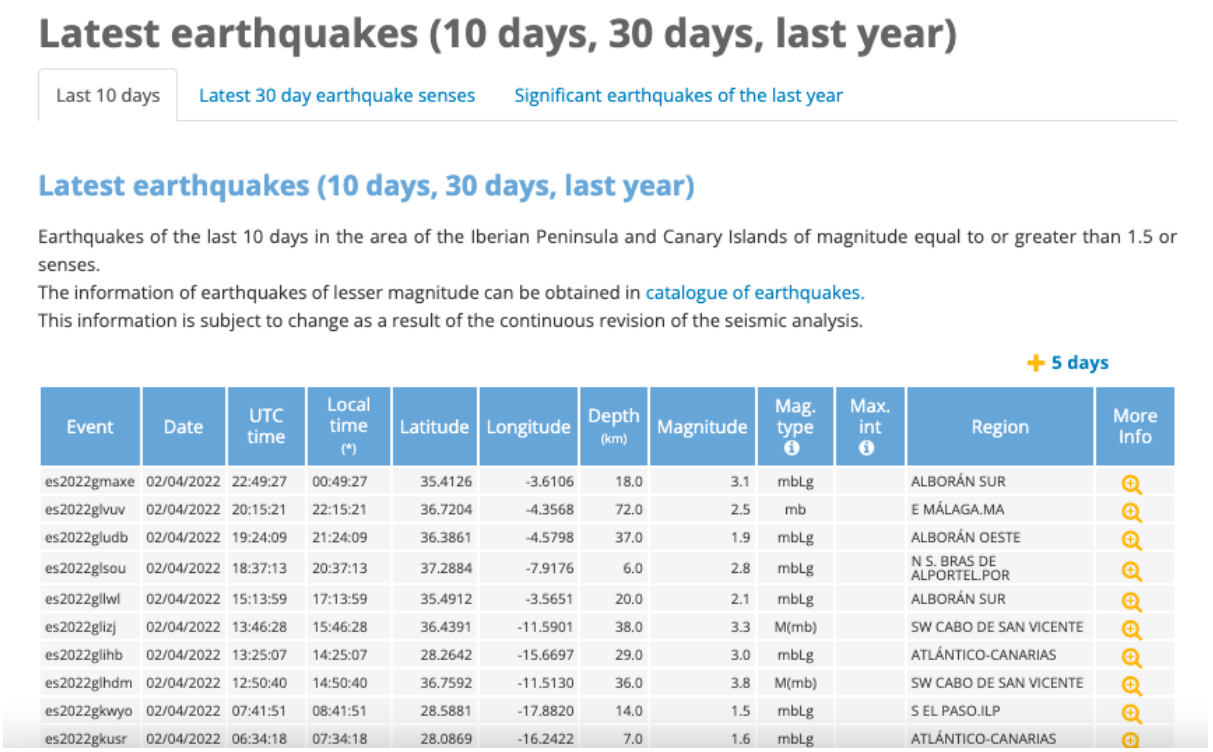
1. First we create an app using the following command:



2. We plugin the required dependencies in `pubspec.yaml` file.



3. We select a [suitable website] to scrape data from and inspect it to know the HTML tags that contain relevant data.



Latest earthquakes (10 days, 30 days, last year)

Last 10 days | Latest 30 day earthquake senses | Significant earthquakes of the last year

Latest earthquakes (10 days, 30 days, last year)

Earthquakes of the last 10 days in the area of the Iberian Peninsula and Canary Islands of magnitude equal to or greater than 1.5 or senses.
The information of earthquakes of lesser magnitude can be obtained in catalogue of earthquakes.
This information is subject to change as a result of the continuous revision of the seismic analysis.

+ 5 days

Event	Date	UTC time	Local time (*)	Latitude	Longitude	Depth (km)	Magnitude	Mag. type	Max. int	Region	More Info
es2022gmaxe	02/04/2022	22:49:27	00:49:27	35.4126	-3.6106	18.0	3.1	mbLg		ALBORÁN SUR	
es2022glvuv	02/04/2022	20:15:21	22:15:21	36.7204	-4.3568	72.0	2.5	mb		E MÁLAGA.MA	
es2022gludb	02/04/2022	19:24:09	21:24:09	36.3861	-4.5798	37.0	1.9	mbLg		ALBORÁN OESTE	
es2022glsou	02/04/2022	18:37:13	20:37:13	37.2884	-7.9176	6.0	2.8	mbLg		N S. BRAS DE ALPORTEL.POR	
es2022gllwl	02/04/2022	15:13:59	17:13:59	35.4912	-3.5651	20.0	2.1	mbLg		ALBORÁN SUR	
es2022glizj	02/04/2022	13:46:28	15:46:28	36.4391	-11.5901	38.0	3.3	M(mb)		SW CABO DE SAN VICENTE	
es2022glihb	02/04/2022	13:25:07	14:25:07	28.2642	-15.6697	29.0	3.0	mbLg		ATLÁNTICO-CANARIAS	
es2022glhdm	02/04/2022	12:50:40	14:50:40	36.7592	-11.5130	36.0	3.8	M(mb)		SW CABO DE SAN VICENTE	
es2022gkwyo	02/04/2022	07:41:51	08:41:51	28.5881	-17.8820	14.0	1.5	mbLg		S EL PASO.ILP	
es2022gkusr	02/04/2022	06:34:18	07:34:18	28.0869	-16.2422	7.0	1.6	mbLg		ATLÁNTICO-CANARIAS	

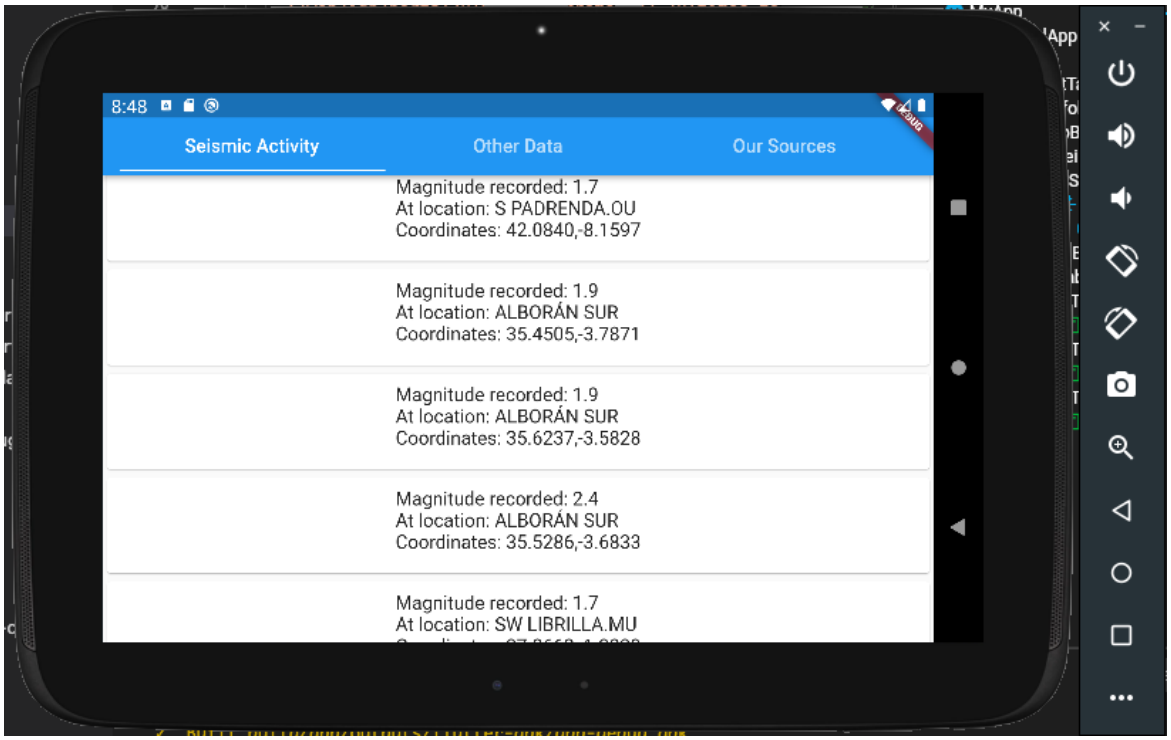
The above website shows the seismic activities as an early warning sign that something is going on underground with a **volcano**.

4. Write the appropriate code:

```
void fetchSeismicdata() async {
  if (await webScraper.loadWebPage(
      '/web/ign/portal/ultimos-terremotos/-/ultimos-terremotos')) {
    setState(() {
      volcanicData = webScraper.getElement('tr > td:nth-child(21)', []);
```

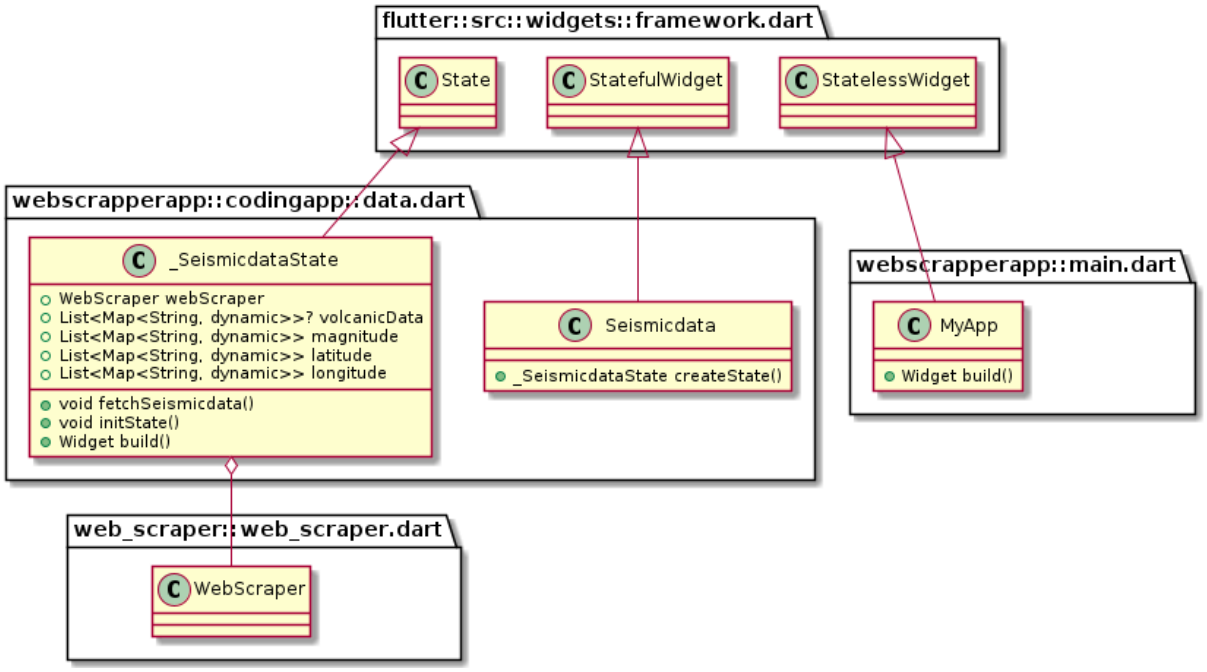
```
      magnitude = webScraper.getElement('tr > td:nth-child(15)', []);
      latitude = webScraper.getElement('tr > td:nth-child(9)', []);
      longitude = webScraper.getElement('tr > td:nth-child(11)', []);
    });
  }
}
@override
void initState() {
  super.initState();
  // Requesting to fetch before UI drawing starts
  fetchSeismicdata();
}
```

Expected Outcome:



Here is a [VIDEO Demonstration]() of the above code.

UML Class Diagram Representation for our Flutter Web Scraping app



We can see the complete Unified Modeling Language (UML) structure diagram here. We have the data where we want it. Next is to **visualize** the web scraped data into our Liquid Galaxy by handling the conversion to **KML data format**.

Converting JSON / String data to KML

We have fetched data from the Web 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 **visualize** it at a later stage.
To accomplish this, we modify our **Web Scraper app** as follows:

1. We scrape data from a source that provides coordinates of the volcanic activity in La Palma.

2. We convert our JSON / String data into **GeoJSON** using the following Dart code that uses the **geojson_vi** package.

3. We convert the String<Map> or JSON file to **KML data format** to visualize this data in the Liquid Galaxy.

To create KML from coordinates (Point Data):

```
class Point {
 double lat;
 double lng;

 Point(this.lat, this.lng);

 generateTag(){
   return '''
     <Point>
       <gx:drawOrder>1</gx:drawOrder>
       <coordinates>${this.lng},${this.lat}</coordinates>
     </Point>
   ''';
 }

 toMap() {
   return {
     "lat": lat,
     "lng": lng,
   };
 }

 static fromMap(dynamic map) {
   return Point(map['lat'], map['lng']);
 }

 @override
 String toString() {
```

```
    return super.toString();
  }
}
```

Create KML instructions to Fly to a particular coordinate:

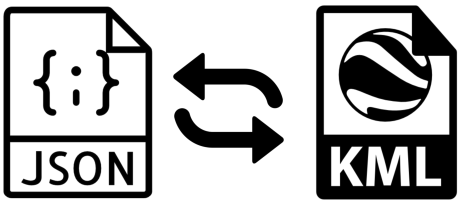
```
import 'package:ras/models/kml/LookAt.dart';

class Flyto {
  LookAt lookAt;

  Flyto(this.lookAt);

  generateFlyto() {
    return 'flytoview=${this.lookAt}';
  }
}
```

We convert our web scraped 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 visualize 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:

1. Create a reference to the file location:

```
Future<File> get _localFile async {
 final path = await _localPath;
 return File('$path/counter.kml');
}
```

2. **Write data** to the file:

```
Future<File> writeCounter(int counter) async {
 final file = await _localFile;
 // Write the file
 return file.writeAsString('$counter');
}
```

3. **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;
 }
}
```

4. To **save the KML file** in the downloads directory.

```dart
import 'dart:io';
// ignore: import_of_legacy_library_into_null_safe
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);
   }
 }
}
```

Connect the Flutter app to the LG rig in a VM setup

To connect our Flutter app to the LG rig, we need to set up the Liquid Galaxy first, as I have demonstrated in [this video](#).

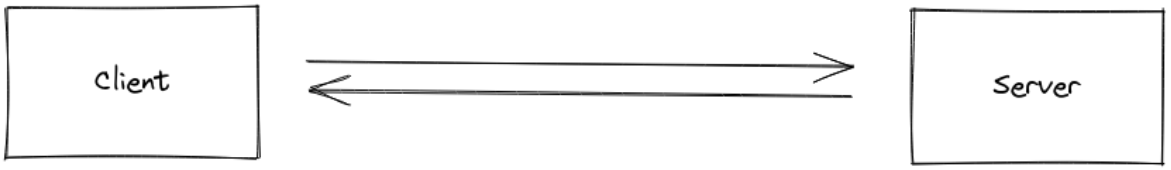
Then, in a Virtual Machine setup, we enable **Port Forwarding,** as shown below:

..	TCP	127.0.0.1	2222	192.168.22.4	22
..	TCP	127.0.0.1	12345	192.168.22.4	45678
..	TCP	127.0.0.1	8080	192.168.22.4	80
..	TCP	127.0.0.1	4444	192.168.22.4	3000

Port number 45678 is the one Liquid Galaxy uses. Then we establish the connection through **web sockets**.

Socket.io

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.**



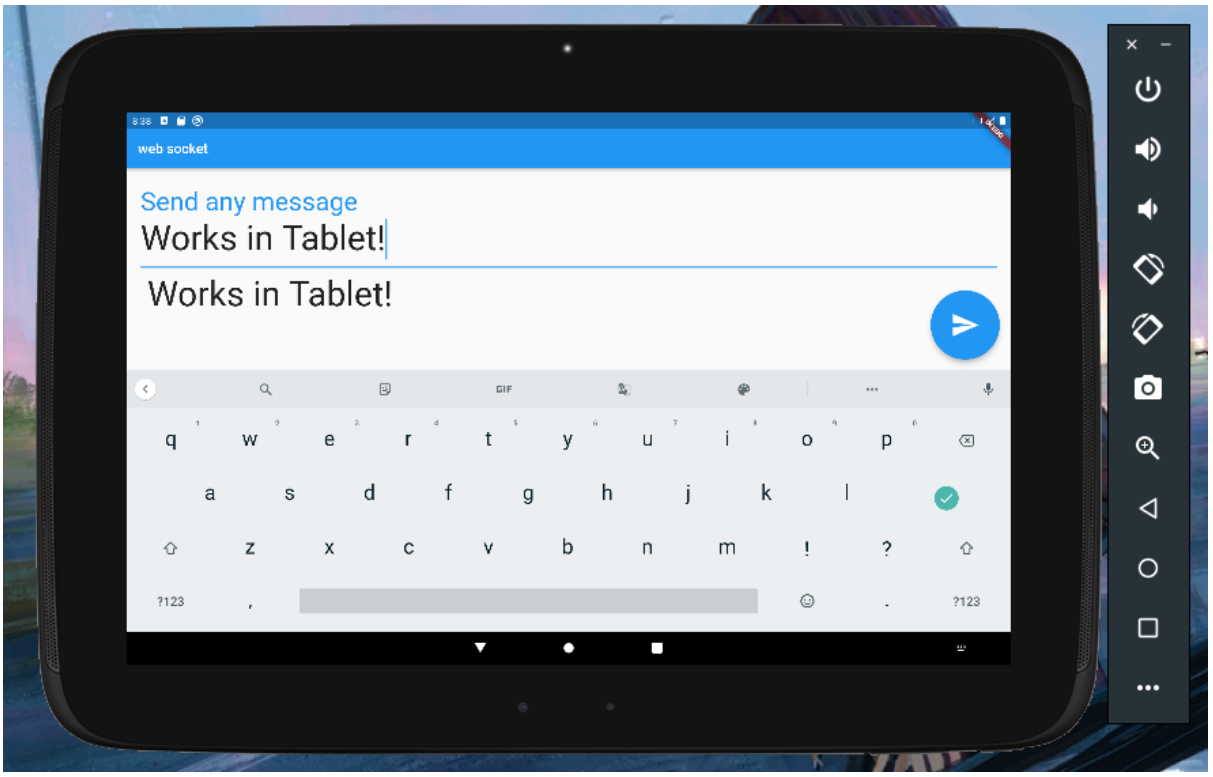
A basic code that helped me understand **how web socket works** is as follows:

```
     home: MyHomePage(
       channel:
           IOWebSocketChannel.connect("wss://echo.websocket.events"), //
server
     ),
   );
 }
}
```

```
         StreamBuilder(
             stream: widget.channel.stream,
             builder: (BuildContext context, AsyncSnapshot<dynamic>
snapshot) {
               return Padding(
                 padding: const EdgeInsets.all(10.0),
                 child: Text(snapshot.hasData ? '${snapshot.data}' : ''),
               );
             })
       ]),
     ),
     floatingActionButton: FloatingActionButton(
       child: const Icon(Icons.send),
       onPressed: _sendMyMessage,
     ),
   );
 }
```

The above code allows us to connect to a test server provided by **websocket.org** and the server sends back the same message we send to it.

Expected Outcome (on a Nexus 10.05’ inch tablet)**:**



Code to establish a connection between our Flutter app and Master Machine:

The following code uses the **ssh 0.0.7** 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;
    });
  }
}
```

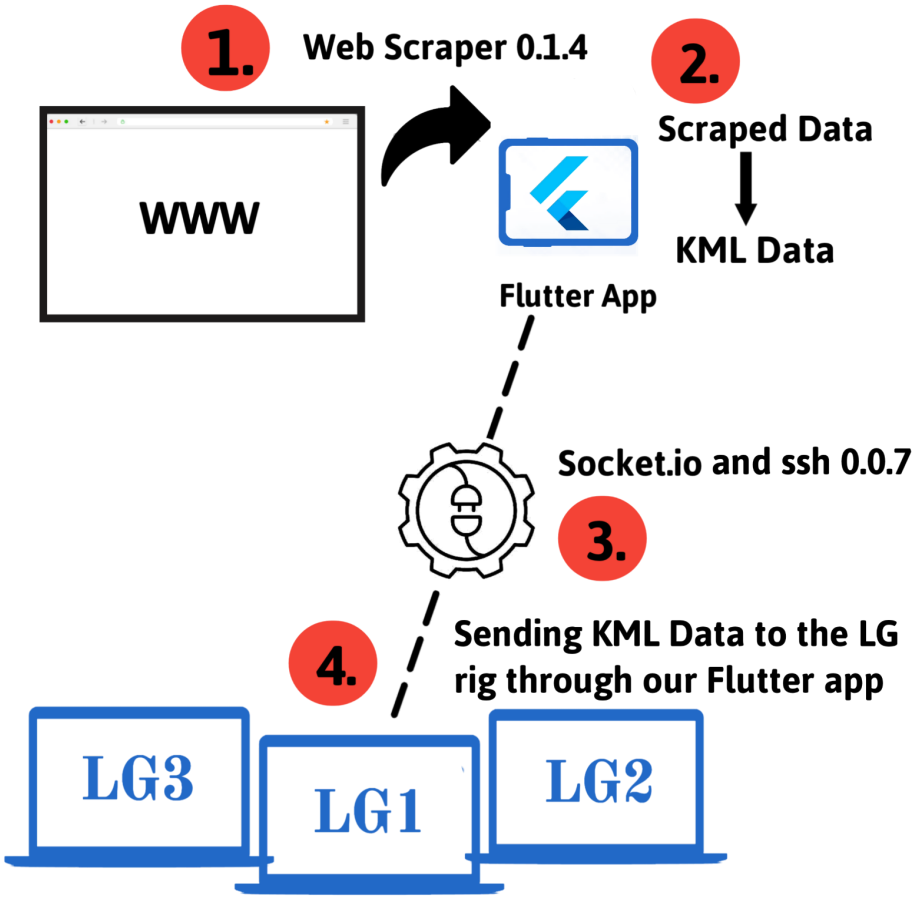
Sending KML Data to the Liquid Galaxy

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:

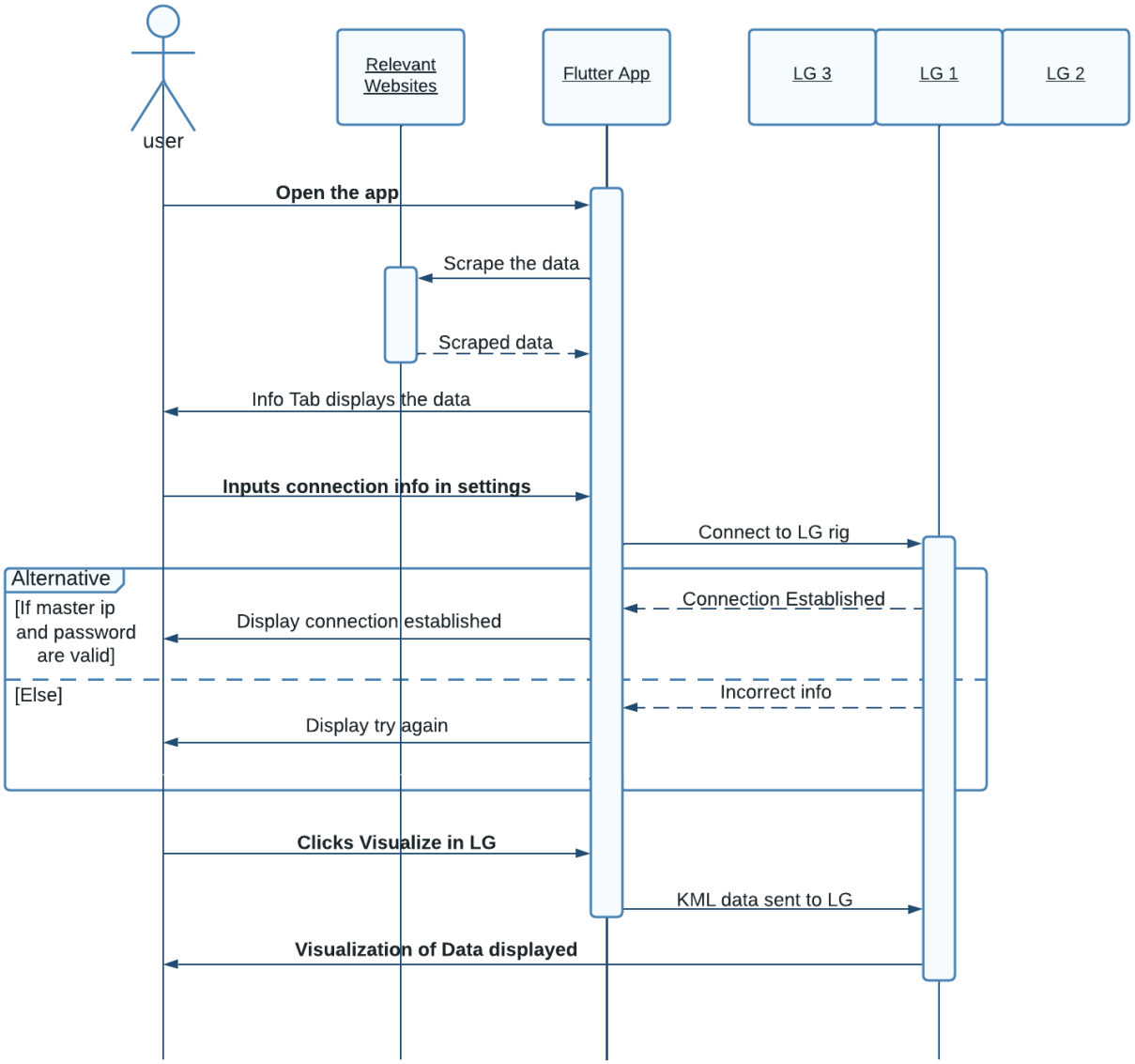
```
 launchToLG(ProjectViewArgs args) {
   Project? p = args.project;
   // create kml based on geodata attribute
   String content = KML.buildKMLContent(args.project.geodata.markers,
       args.project.geodata.areaPolygon,
args.project.geodata.landingPoint);
   KML kml = KML(args.project.projectName, 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');
   });
 }
```

A **breakdown of the approach** can be illustrated as follows:

1. We first web scrape all the relevant data available on the web with the help of the **Dart native package** called **web_scraper 0.1.4.**
2. Then we convert the web scraped information into both **GeoJSON format KML data formats**. We store them in the Android Tablet's memory.
3. We establish a connection between the Flutter app and our Liquid Galaxy rig with the help of **Socket.io** and **ssh 0.0.7**.
4. We send the **KML Data to the Liquid Galaxy**, as shown above, to get the visualization.



A UML Sequence Diagram to illustrate the whole workflow is as follows:



Below are the mockup screens for our Flutter app.

The first screen consists of all the collected data segregated into various maps that we can select and hit the **Visualize in LG** button to load them.

The second screen UI shows a **map view of La Palma** that we can develop with the Google Maps API.

The third screen has the **raw JSON / Map <String, dynamic>** information that we scraped from the web and converted to a readable text format. It also has an **Update Info** button allowing the users to **update the web scraped information** when needed manually.

The fourth screen shows the Connection manager needed to connect to the **Master machine of the Liquid Galaxy.**

The fifth screen shows the **LG tasks**, such as shutdown, relaunch, Clean KML, etc., that we can access from the hamburger menu.

The last screen shows the menu options such as help, about, Connection Manager, and LG Tasks.

In the UX mockups for the LG rig:

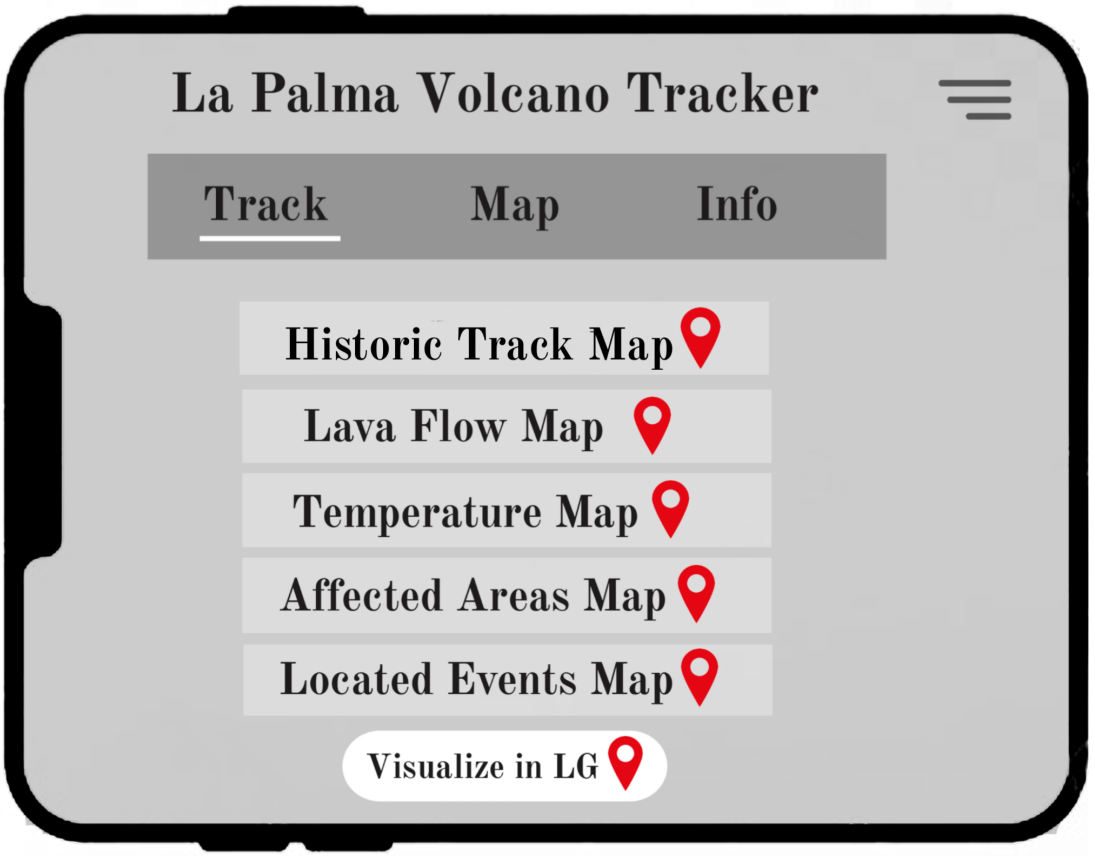
The first Big screen Mockup shows the Lava flow activity over time in the **Historic track** option. This information can be web scraped from sites such as Volcano.si.edu | Eruptive History or World 3D Map to visualize data over a long-range or short-range of time, respectively.

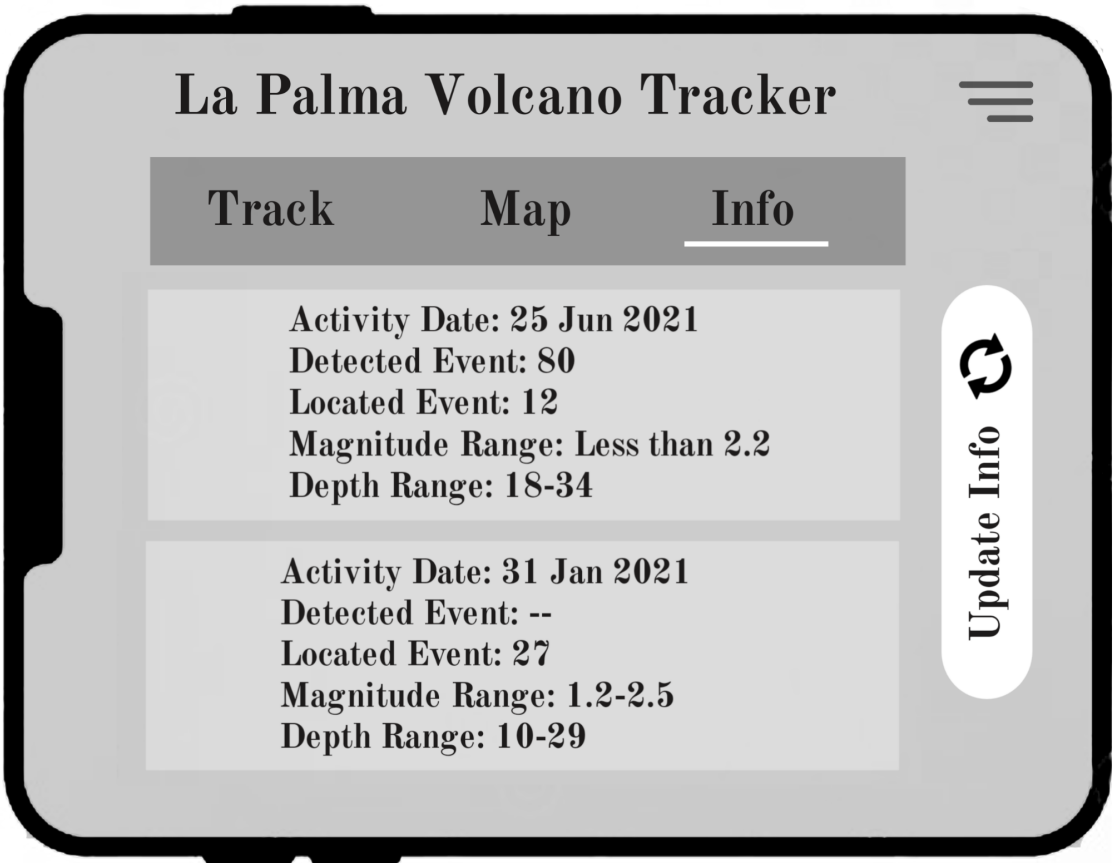
The second Big screen mockup gives a visualization of the **thermal conditions** of La Palmas due to the Volcanic activity.

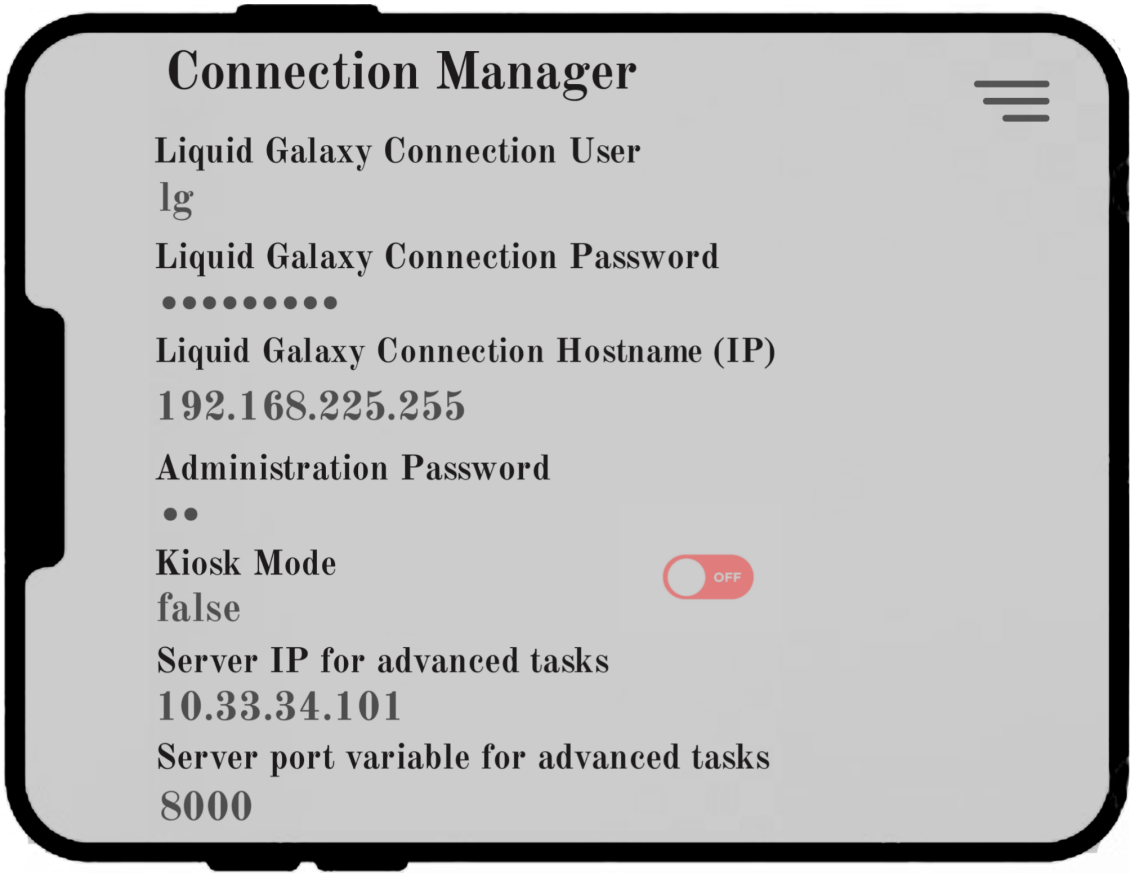
The third Big screen mockup shows the buildings, roads, and **areas affected** by Volcanic activity. We can scrape this data from the Gdacs website.

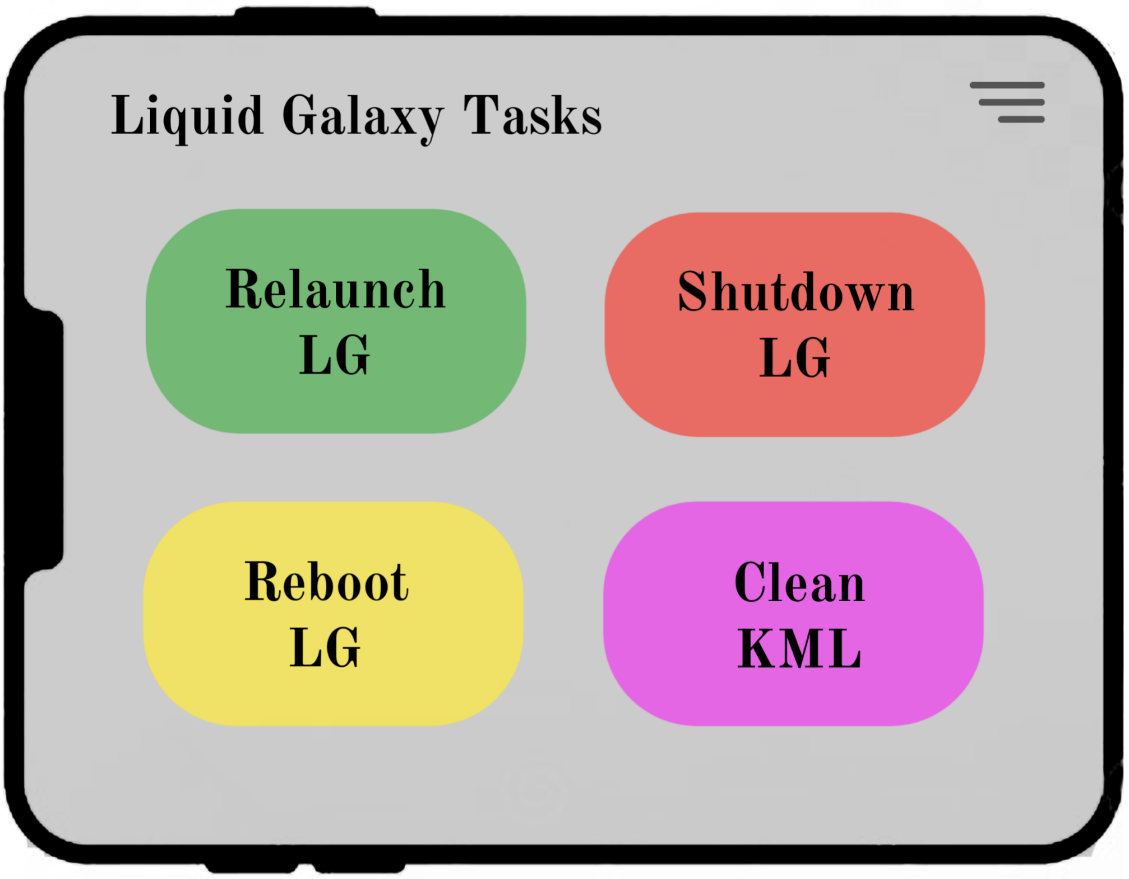
I also plan to have Visualizations of tracks of Change in Landscape and Sulfur Dioxide emissions due to the Volcanic eruptions, depending on the data available on the internet. We can have the menu items of the Tracks tab **scrollable** to add more such options.

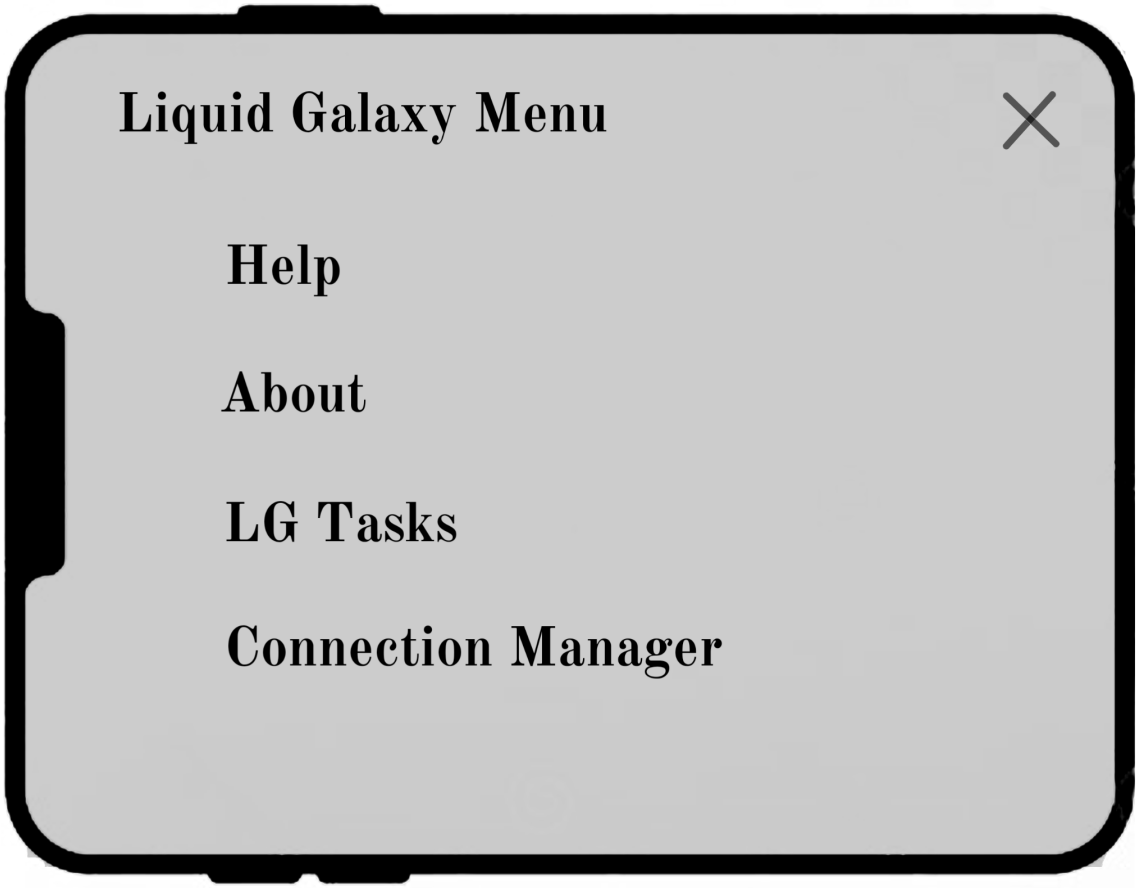
MOCKUP UX LAYOUT OF THE SCREENS

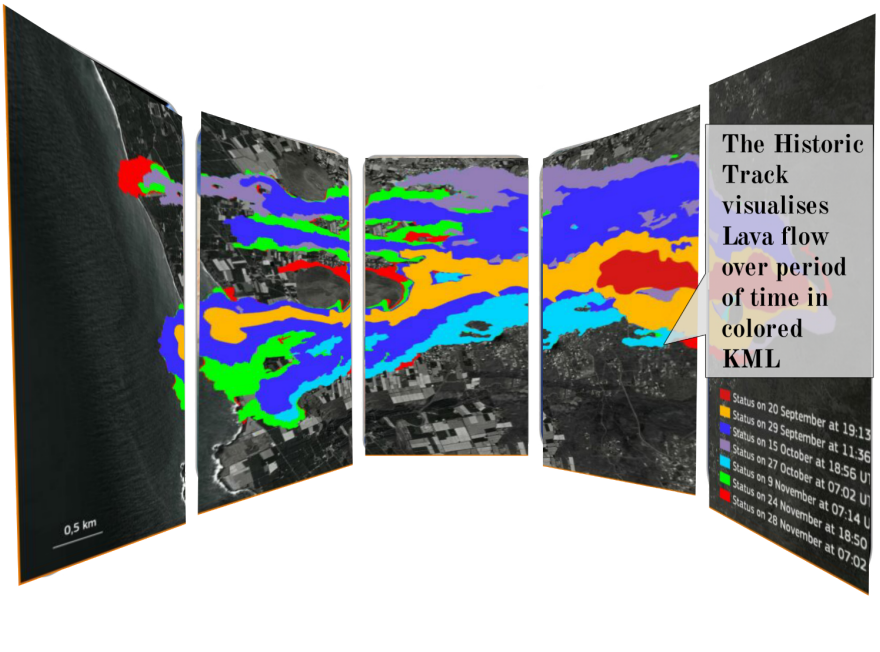


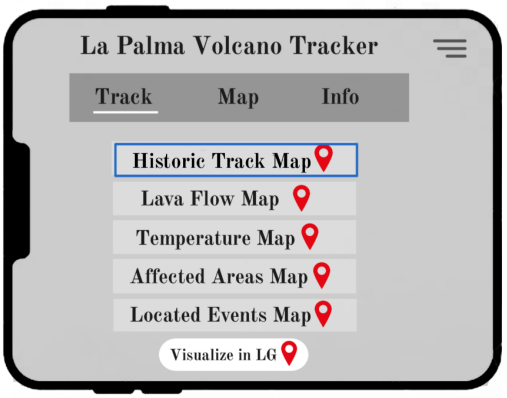

La Palma Volcano Tracker
Track
Map
Info
Activity Date: 25 Jun 2021
Detected Event: 80
Located Event: 12
Magnitude Range: Less than 2.2
Depth Range: 18-34
Activity Date: 31 Jan 2021
Detected Event: --
Located Event: 27
Magnitude Range: 1.2-2.5
Depth Range: 10-29
Update Info


Connection Manager
Liquid Galaxy Connection User
lg
Liquid Galaxy Connection Password
•••••••••
Liquid Galaxy Connection Hostname (IP)
192.168.225.255
Administration Password
••
Kiosk Mode
OFF
false
Server IP for advanced tasks
10.33.34.101
Server port variable for advanced tasks
8000

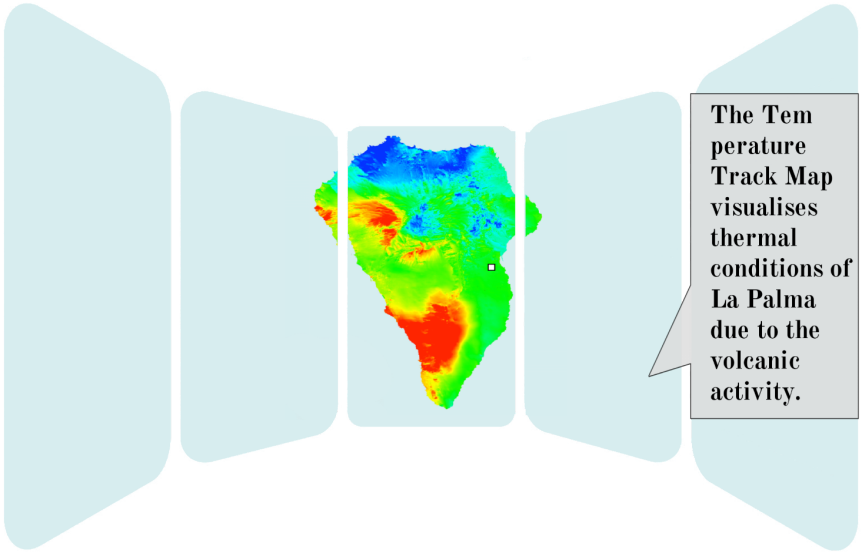

Liquid Galaxy Tasks
Relaunch LG
Shutdown LG
Reboot LG
Clean KML

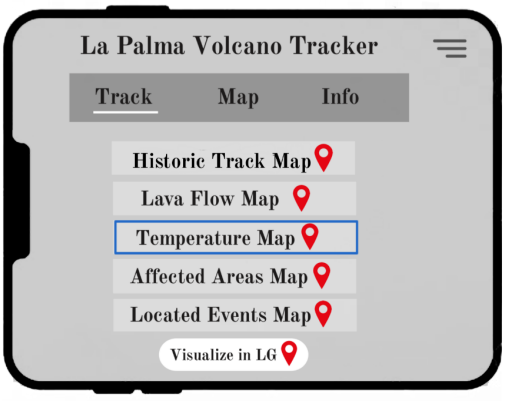

Liquid Galaxy Menu
Help
About
LG Tasks
Connection Manager



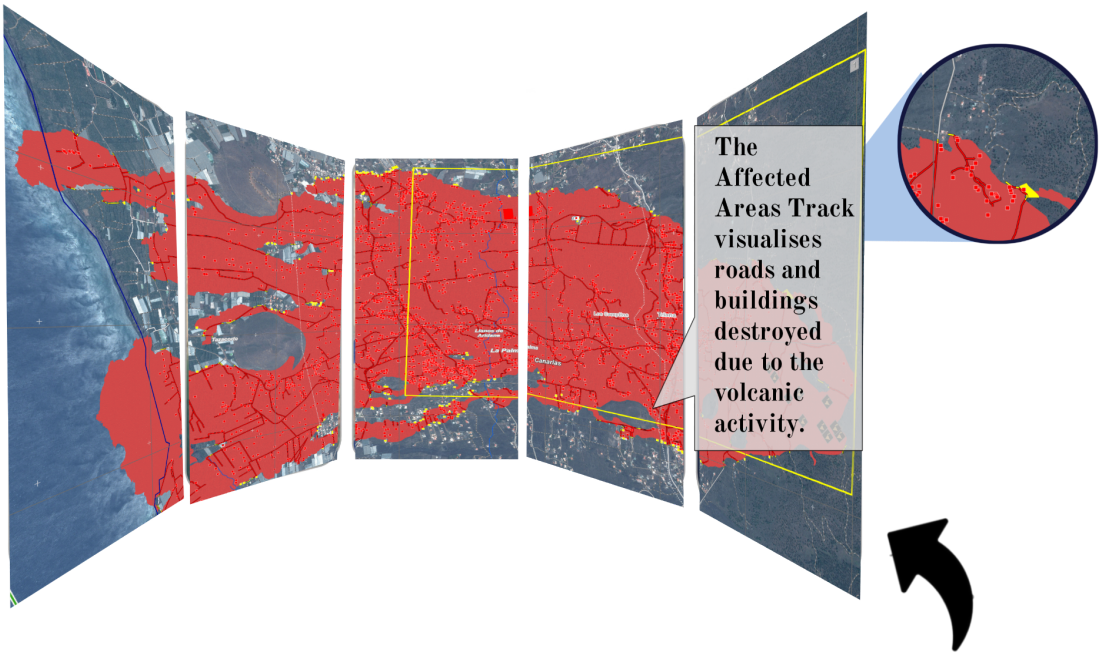


La Palma Volcano Tracker app sends KML data to the Liquid Galaxy for Visualization of Historic Track data, which is similar to Lava flow Track but over a range of time.

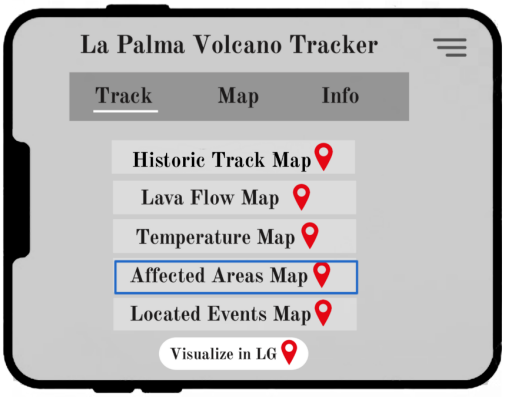


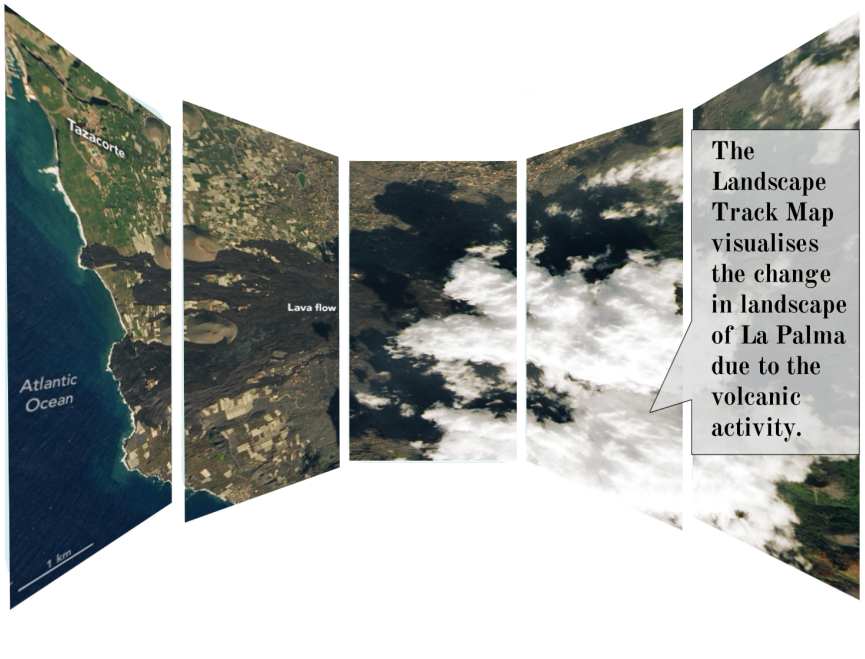


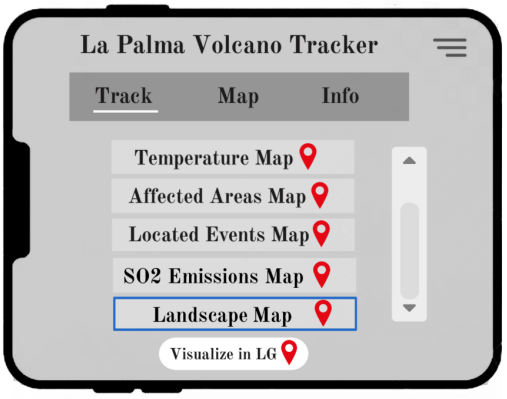
La Palma Volcano Tracker app sends KML data to the Liquid Galaxy for Visualization of Thermal Data.



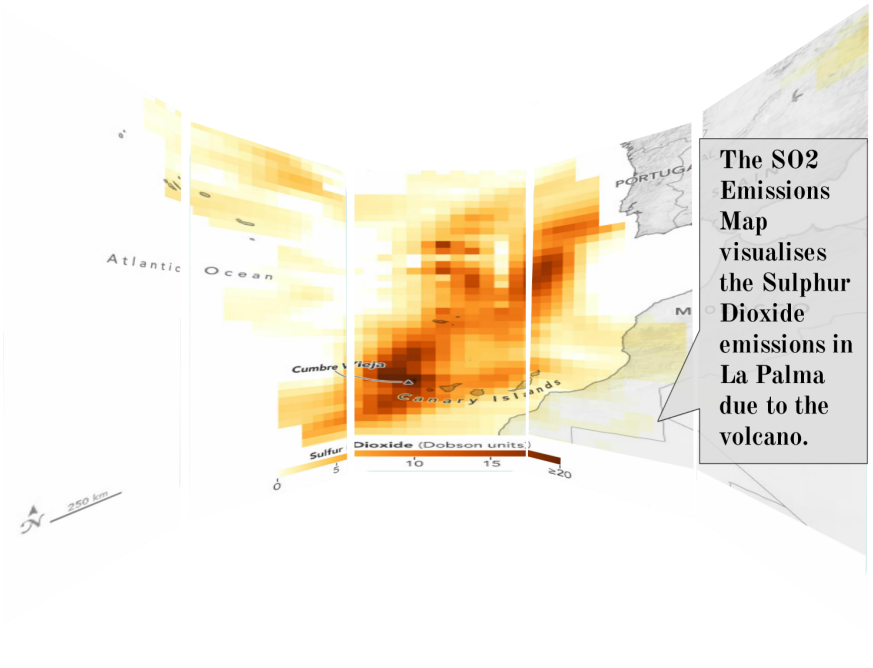
La Palma Volcano Tracker app sends KML data to the Liquid Galaxy for Visualization of Affected Areas. Red color shows Roads and Buildings destroyed, Yellow color shows Roads and Buildings possibly damaged.

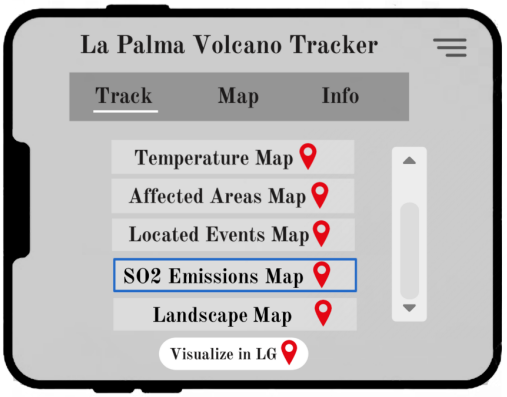






La Palma Volcano Tracker app sends KML data to the Liquid Galaxy for Visualization of Change in landscape before and after the volcanic eruption.





La Palma Volcano Tracker app sends KML data to the Liquid Galaxy for Visualization of Harmful Sulphur Dioxide emissions due to the volcanic eruption.

Use Cases

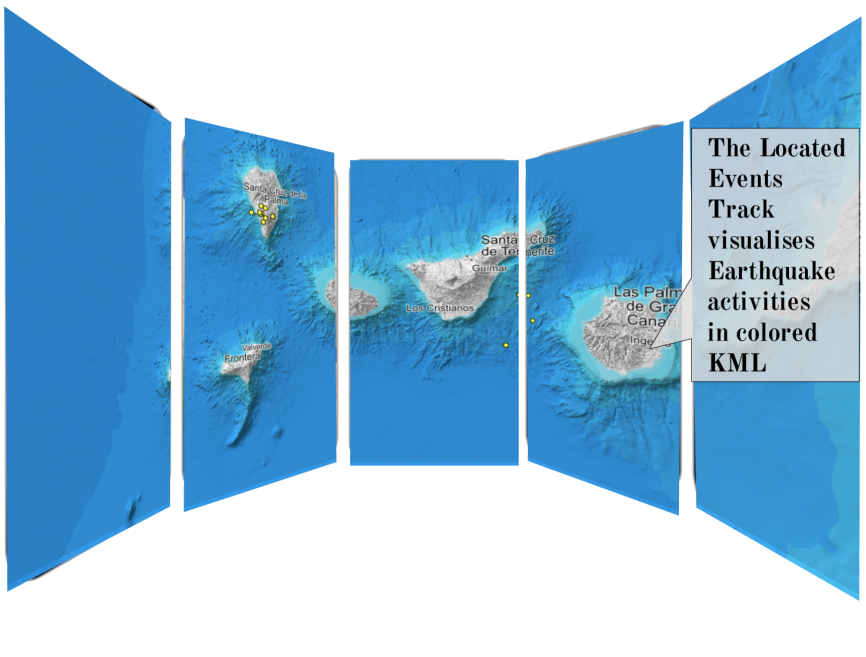
Disaster Management:

We can use our application to keep track of volcanic activities and evacuate civilians from the potentially affected areas. Liquid Galaxy can visually show **the avoiding areas** downwind and river valleys downstream from the volcano. The app keeps itself updated on the lava flow, and it serves as a **user guide** to know about catastrophes and conditions users with advice on disaster preparedness.

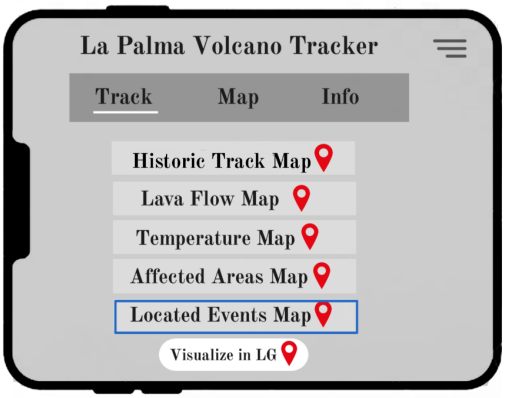
La Palma Volcano Tracker App sends KML data to the Liquid Galaxy for Visualization of Lava Tracks.

Geography:

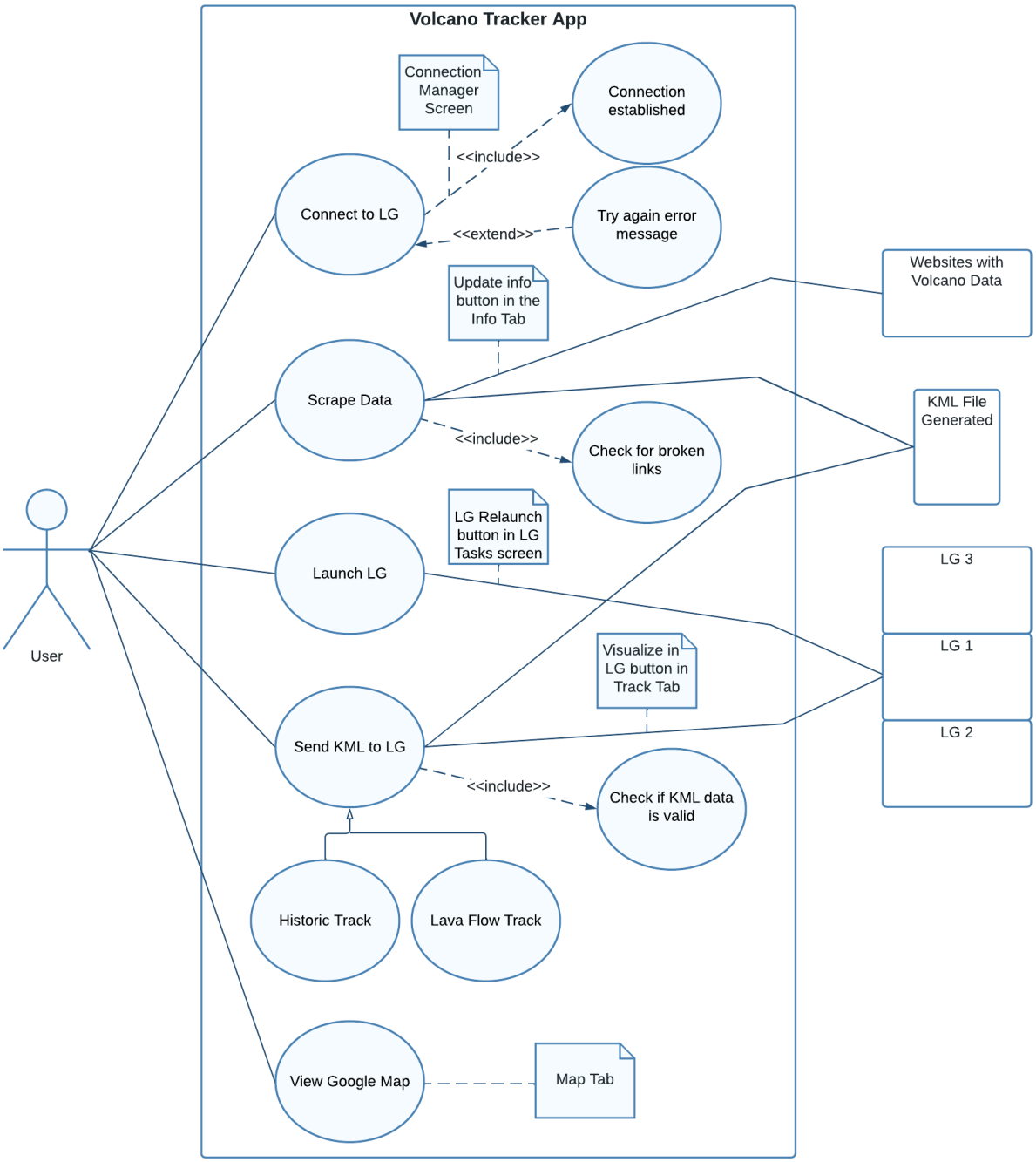
We can also use our application for **educational purposes**, such as studying natural disasters, volcanic activities, and catastrophes from the destruction caused by La Palma Volcano. Teachers can show that Seismic activities such as **Earthquakes** are an early warning sign that something is going on underground with a **volcano** and visualize the data in the **Liquid Galaxy** for a clearer understanding.



La Palma Volcano Tracker App sends KML data to the Liquid Galaxy for Visualization of Located Earthquake Events.



A detailed description of every tablet interface option through the **UML Use Case diagram**.



We can explain the above diagram as follows:

- The user must first **connect the app to the LG rig**, so he goes to the **Connection Manager** and fills in the required information. The user can either successfully connect or get an error message to try again.

- After the user has successfully connected, he goes to the **Info Tab** and hits the **Update Info** button to **scrape** the volcanic data into our tablet. Doing this will generate our needed **KML file** and store it in the Downloads directory with the help of the **path_provider** plugin that allows reading and writing files in our flutter app.

- Then the user will launch our LG using the **LG relaunch button** to confirm we have successfully connected to the rig and we can pass instructions.

- Then we **send** the generated **KML data** to the LG rig to make the visualization possible.

- Also, the user gets various track options to choose from, so if he selects another track, the app automatically **cleans the data** of the previous one.

- The user also has an interface to La Palmas Map in the **Maps tab**.

- Lastly, the hamburger menu provides the user with **about and help** options to know more about the app or get help with specific commands.

Linked Technologies

geojson_vi 2.0.7:

The GeoJSON package allows us to create, read, search, update and delete the geospatial data interchange format. We will use it to convert the **web scraped data** into **GeoJSON format**, which might be helpful as **future support** as it is easy to read, easy to use, and generally smaller in size than KML.

Google Maps API:

I plan to make the Flutter app more user-friendly by adding an interactive **Google Map** of La Palma.

We first need to add the dependency in the **pubspec.yaml** file.

```
dependencies:
 ...
 google_maps_flutter: ^0.4.0
```

A basic code to add the **Google Map Widget** is as follows:

```
class _MyAppState extends State<MyApp> {

Completer<GoogleMapController> _controller = Completer();

static const LatLng _center = const LatLng(28.7099744,-17.8984565);

void _onMapCreated(GoogleMapController controller) {

  _controller.complete(controller);
```

```
}

@override

Widget build(BuildContext context) {

  return MaterialApp(

    home: Scaffold(

      appBar: AppBar(

        title: Text('Maps Sample App'),

        backgroundColor: Colors.green[700],

      ),

      body: GoogleMap(

        onMapCreated: _onMapCreated,

        initialCameraPosition: CameraPosition(

          target: _center,

          zoom: 11.0,

        ),

      ),

    ),

  );

}

}
```

We can modify the above code to get the Google Map widget inside the **Map tab** of our Flutter app, as shown below:

We can bring this functionality of the google maps widget as an add-on to our Flutter app. In contrast, the app's primary purpose remains the same, i.e., to **visualize all the collected data in the Liquid Galaxy**.

Timeline

Bonding Period (May 20 - June 12)

Week 1 (May 20 - May 27): 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.
Week 2 (May 28 - June 4): I will prepare a list of all the websites from which we can scrape relevant data. I will web-scrape a few of them using the already built Web Scraper Flutter code to **test our code and check loading time**.
Week 3 (June 5 - June 12): For the Flutter app, I will look into the code needed to establish a **connection between the app and the Master machine**.

First Working Period (June 13 - July 24)

Week 4 & 5 (June 13 - June 26): I will start with the **basic layouts** needed for our Flutter app and begin with the connection settings integration.
Week 6 & 7 (June 27 - July 11): I plan to complete the **connection settings** page during this period, so later on, it's easier to check if the **Visualize in LG** button works when we send KML data to the Liquid Galaxy.
Week 8 & 9 (July 12 - July 24): I will create the **Web Scraper** for our Flutter app using **web_scraper_0.1.4,** which scrapes data when the user clicks the **Update Info button** in the **Info tab**. On the **Info tab**, I plan to show the JSON / Map <String, dynamic> data our app obtains, converted into simple text format to make the information more readable.

Second Working Period (July 25 - September 4)

Week 10 & 11 (July 25 - August 7): 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.

Week 12 & 13 (August 8 - August 21): I plan to make a few options in the **Track tab**, such as Lava Flow Map, Historic Map, Temperature Map, Affected Areas Map,

Located Events Map, Detected Events Map, etc. These options will have the KML data **segregated into relevant Maps**, so the Visualization looks neat.
Week 14 & 15 (August 22 - September 4): I plan to develop an interface with the code to send **KML data** to the **Liquid Galaxy** using the options available in the **Track tab**, such that the user can select any of the tracks and click the **Visualize in LG button** in our Flutter app to visualize the data in the Liquid Galaxy. The Flutter app will send the **KML data that is uploaded or changed** in the Liquid Galaxy, making the Visualization possible.

Third Working Period (September 5 - September 12)

I will debug any errors and create the **Maps tab** that contains the **La Palma google maps widget** to make our app interactive and user-friendly.

Closing and Finalization (September 12 - September 19)

I will run a few final checks. And try with several tracks of different Map data (Lava Flow, Temperature, Affected Areas, Located Events and Historic Track), check if the Visualize in LG button is working in our Flutter app and the Info tab has the scraped information updating. Finally, we can publish our Flutter app in the Play Store.

Conclusion:

In Conclusion, I will always be ready for debugging and updating code since Flutter is in the active development stage. So for any changes in their codebase that might reflect an error in our App, I'll be happy to modify and update them.