1.) SOCKET IO

Documentation : https://socket.io/docs/v4/

Socket.IO is a library that enables low-latency, bidirectional and event-based communication between a client and a server.

a. Server
   - Once you have installed the Socket.IO server library, you can now init the server. Server initiation example can be found here : https://socket.io/docs/v4/server-initialization/
   - A Socket is the fundamental class for interacting with the client. It inherits all the methods of the Node.js EventEmitter, like emit, on, once or removeListener. https://socket.io/docs/v4/server-socket-instance/
   - On the server we always need to setup an event handler once a client is connected to the server. Example :

     ```
     io.on("connection", (socket) => {
       …
     });
     ```

b. Client
   - Once you have installed the Socket.IO client library, you can now init the client. Client initiation example can be found here : https://socket.io/docs/v4/client-initialization/

c. Events

   Sending and receiving message/data in Socket IO is done through event
   - Emiting Events

     Doc : https://socket.io/docs/v4/emitting-events/

     Basic Emit example :

     Server:

     ```
     io.on("connection", (socket) => {
       socket.emit("hello", "world");
     });
     ```

     Client:

     ```
     socket.on("hello", (arg) => {
       console.log(arg); // world
     });
     ```

     *) This works for both direction
   - Listening to Events

     Doc : https://socket.io/docs/v4/listening-to-events/

     Basic listener (EventEmitter methods) :

     socket.on(eventName, listener)

     example :

     ```
     socket.on("details", (args) => {
       console.log(args);
     });
     ```

2.) OpenLayers

Documentation : https://openlayers.org/doc/tutorials/concepts.html

Map

The core component of OpenLayers is the map (from the ol/Map module). It is rendered to a target container (e.g. a div element on the web page that contains the map). All map properties can either be configured at construction time, or by using setter methods, e.g. setTarget().

Example of generating map :

```
<div id="map" style="width: 100%, height: 400px"></div>
```

This is the html tag that contains the map

```
import Map from 'ol/Map.js';

const map = new Map({target: 'map'});
```

This script will generate a map on the target of html id 'map'

Layer

A layer is a visual representation of data from a source. OpenLayers has four basic types of layers:

- ol/layer/Tile - Renders sources that provide tiled images in grids that are organized by zoom levels for specific resolutions.
- ol/layer/Image - Renders sources that provide map images at arbitrary extents and resolutions.
- ol/layer/Vector - Renders vector data client-side.
- ol/layer/VectorTile - Renders data that is provided as vector tiles.

In this project we only implemented tile layer and vector layer. Tile layer is used to render map sources, while vector layer is used to render pointer gps icon.

Map layer setup example :

```
import TileLayer from 'ol/layer/Tile.js';

// ...

const layer = new TileLayer({source: source});

map.addLayer(layer);
```

For creating vector layer there are several components that we can adjust. We can make vector layer as following :

```
     var gpsLayer = new VectorLayer({

      source : gpsSource

    });
```

The vector layer needs a source which can be made by adding a feature :

```
    var gpsSource = new VectorSource({
      features: [gpsFeature]
    });
```

For making a feature :

```
    var gpsFeature = new Feature({
      geometry : new Point(fromLonLat([-6.5360378062373,63.6507991441]))
    });
```

Note that geometry is one of Feature properties. It can take Point function input which can take world coordinate with fromLonLat function

Next we can also add a style to the vector :

```
    gpsFeature.setStyle(new Style({
      image : new Icon(({
        src: 'assets/arrow.svg',
        imgSize: [600, 600],
        scale: 0.1,
        color: '#00FF2B'
      }))
    }));
```

Finally we can put all layer together as following :

```
this.map = new Map({
      target: 'map',
      layers: [
        new Tile({
          source: new OSM() // default map layer source
        }),
        gpsLayer
      ]
    });
```

Source

To get remote data for a layer, OpenLayers uses ol/source/Source subclasses. These are available for free and commercial map tile services like OpenStreetMap or Bing, for OGC sources like WMS or WMTS, and for vector data in formats like GeoJSON or KML.

View

The map is not responsible for things like center, zoom level and projection of the map. Instead, these are properties of a ol/View instance.

```
view: new View({
  center:fromLonLat([-6.5360378062373,63.65079914412625]),
  zoom: 10,
  enableRotation: false
})
```

center : this will be the initial point of view when the map is generated

zoom : this will set the map zoom view resolution to the center point

enablerotation : by default this properties is true and the map will be able to be rotated