International Space Station Tracker

- Let's build iss tracker only using vanilla javascript.

Inorder to create this web app, we need to know about API and Javascript promise.

API

API - Application Programming Interface

Interface - what is interface. if you worked with java , you'll know. In OOPS concept, we will be using Interface in order to achieve **Abstraction**

- Suppose we need to build the weather app, we don't need to be in that field to build the application, we can take advantage of some third party services to get us weather details instead of creating it from scratch.
- So the third party services, allow us to use the services by providing interface to access the data only without needing to know the implementation details.
- APIs are access points not Database or web server!!

Almost every top products has API to allow the developers to use them without making them spending time on implementation details.

Example : Spotify API, Youtube API,Twitter API

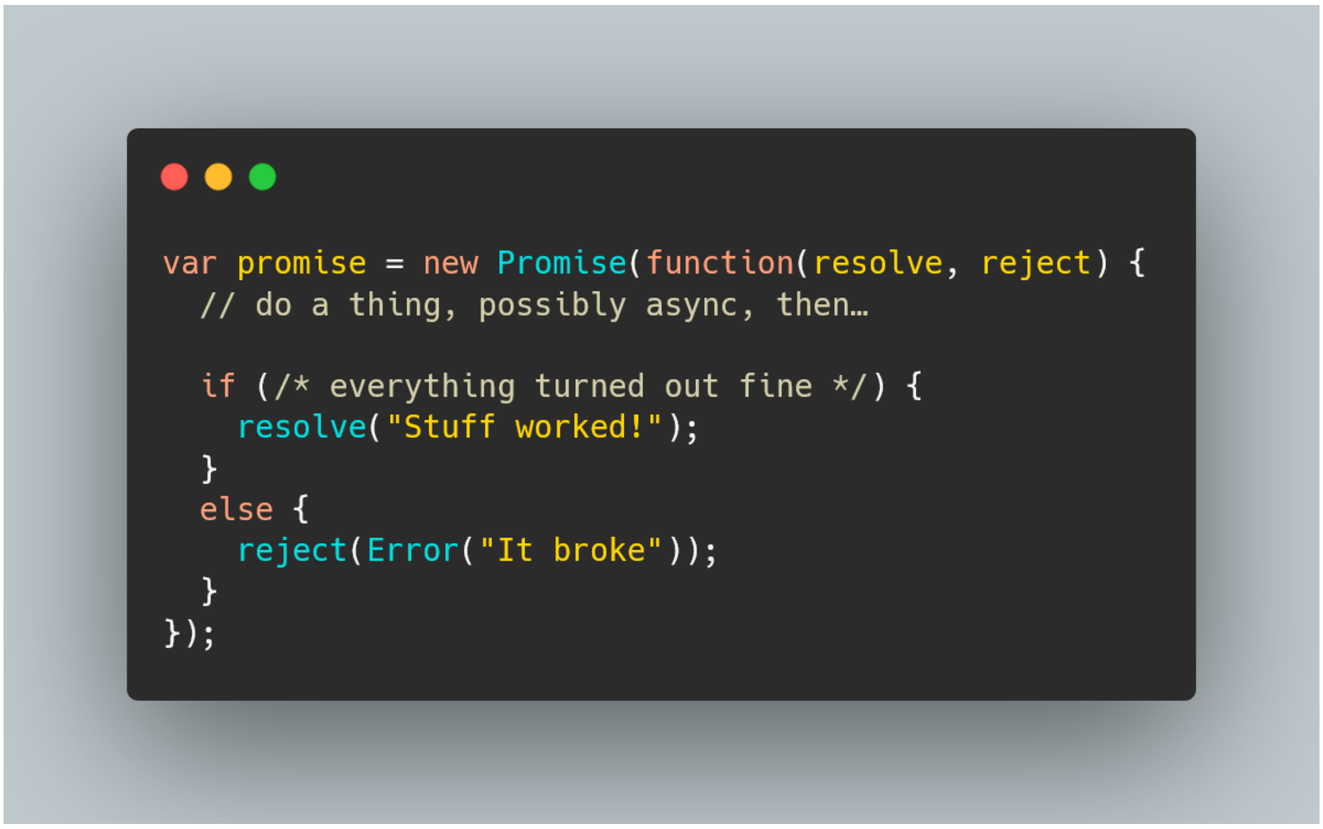
Even spotify uses musixmatch(acts as third-party service) to show lyrics while playing the songs.

In this application, we are gonna use Open Notify API and Maptiler API.

- Open Notify API to get Co-ordinates of ISS.
- Maptiler API to display map on our web app.

Promise and Fetch

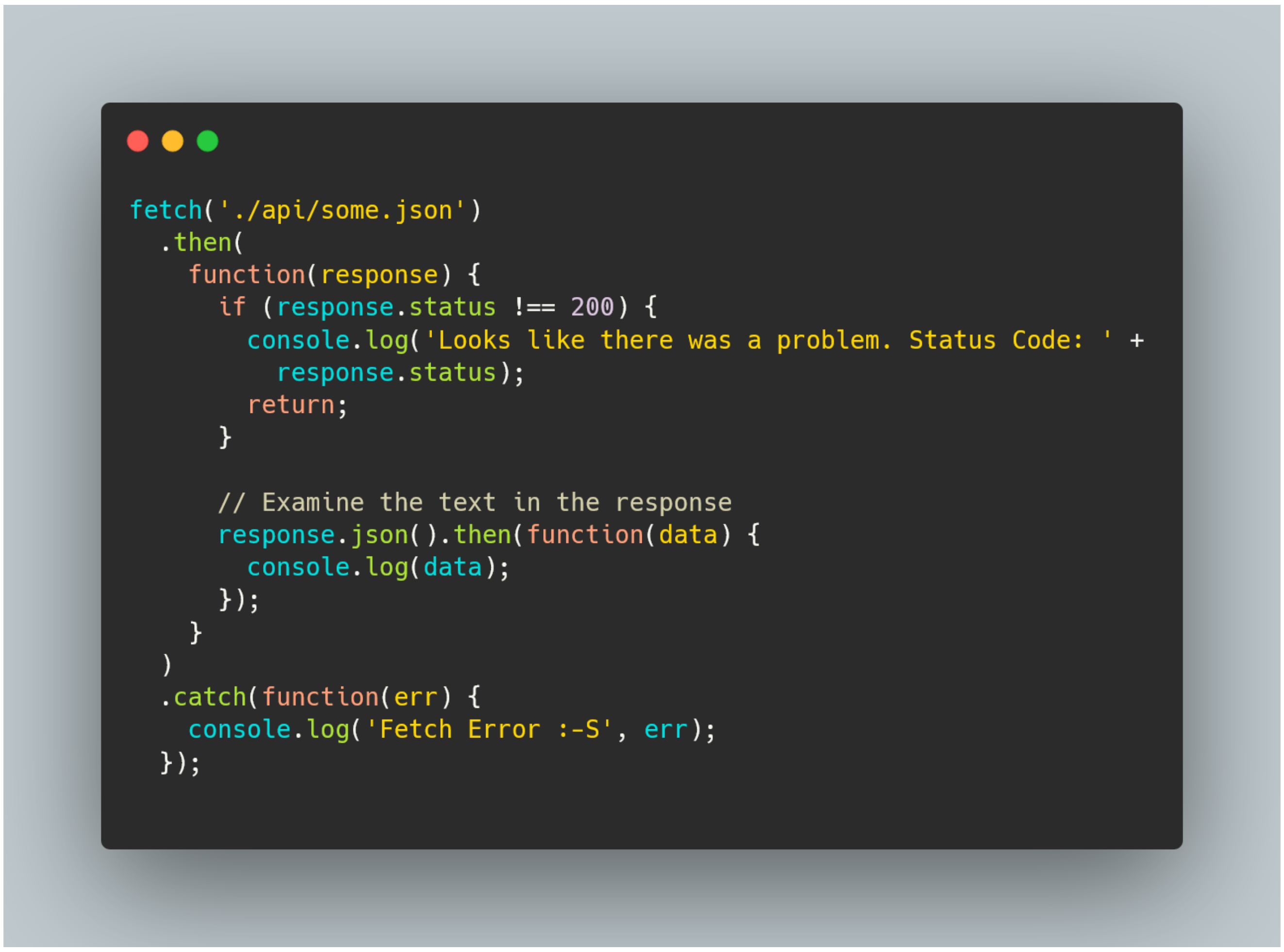
- Promise are the async function of javascript that provides Non-blocking state.
- Promise has two states.
  - Resolve (in case promise is sucessfull)
  - Reject (when promise is broken)



```
var promise = new Promise(function(resolve, reject) {
  // do a thing, possibly async, then…

  if (/* everything turned out fine */) {
    resolve("Stuff worked!");
  }
  else {
    reject(Error("It broke"));
  }
});
```

- Nextup, we are going to see Fetch in javascript.
- Fetch returns Promise.
- Instead of old way of using XMLHttpRequest, the modern way to fetch data from the api is to use **Fetch**.



```
fetch('./api/some.json')
  .then(
    function(response) {
      if (response.status !== 200) {
        console.log('Looks like there was a problem. Status Code: ' +
          response.status);
        return;
      }

      // Examine the text in the response
      response.json().then(function(data) {
        console.log(data);
      });
    }
  )
  .catch(function(err) {
    console.log('Fetch Error :-S', err);
  });
```

ok enough concepts, let's get into coding part!

Boilerplate for html

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>ISS Tracker</title>
</head>
<body>
</body>
</html>
```

Next up, create a div container in order to display the map in the html body and center the div using CSS.
Set map width height using CSS.

```html
    <div id="mapid"></div>
```

```css
 #mapid{
        height: 600px;
        width: 800px;
    }
    body{
        background-color: #1e212d;
    }
    div{
        position:absolute;
        top:0;
        bottom: 0;
        left: 0;
        right: 0;
        margin:auto;
    }
```

Importing Leaflet library in order to customize the map displayed on the web browser. copy and paste in the head section.

```html
<link rel="stylesheet" href="https://unpkg.com/leaflet@1.7.1/dist/leaflet.css"
/>
<script src="https://unpkg.com/leaflet@1.7.1/dist/leaflet.js"></script>
```

Now let's show the map on the html file using javascript code in the body section using script tag.

- Leaflet library is imported and it is generally refered by L (this is like in THREEjs)
- We are gonna create map and set it with container id **mapid** and setting view with lat and long when it is loaded and setting up zoom level

```
<script>
// L.map(MAPID).setView([lat,long],zoom-level)
var map = L.map('mapid').setView([51.505, -0.09], 7);
</script>
```

Next up, we need a map to display real map, so we are gonna take advantage of MapTiler API to get the tile for the map.

- what is tile?
  The tile is like grid-like structure that is placed in the group of pixels. Tile are used mostly in gaming development.

  here we have to specify the tile layer for leaflet to show the map, go to the google search and type "Tiles Maptiler".
  and select the tile of the choice, copy the raster tile with (PNG File) to display map.

  once Copied, paste it as first argument in tileLayer method.

```
L.tileLayer(MAP_URL_HERE, {
    tileSize: 512,
    zoomOffset: -1,
    minZoom: 2,
    attribution: '&copy; <a
href="https://www.openstreetmap.org/copyright">OpenStreetMap</a> contributors'
}).addTo(map);
```

tileSize : can be 512 or
minZoom : value denotes that the minimum zoom value we can perform on the map is 2
attribution : giving Credit to the team
zoomOffset: 0

and add all that to Map.

Nextup, we need somthing like marker in order to know the location of ISS, so we are gonna customize the marker as icon.

For that,
we are gonna use icon method of Leaflet.

```
var iss_icon = L.icon({
 iconUrl:'ISSIcon.png',
 shadowUrl:'ISSIcon_shadow.png',
 iconSize: [50,30],
 shadowSize: [60, 40],
 shadowAnchor: [30, 15],
 iconAnchor: [25,15]
});
```

iconSize , shadowSize - dimension of the PNGs
iconAnchor , shadowAnchor - this denotes the point in the PNG where we have to move with respect to. Mostly, center point will yield much persentation.

Adding custom icon to the marker

```
var iss = L.marker([51.505,-0.09],{icon:iss_icon}).addTo(map);
```

Next, creating circle around custom marker in order to improve the presentation.

```
var circle_icon = L.circle([51.505,-0.09],200e3,
{color:"#c22",opacity:0.3,weight:1,fillColor:"#c22",fillOpacity:0.1}).addTo(ma
p);
```

We displayed map with all necessary details, now we just need to fetch from Open Notify API, we are gonna make use of fetch
that is easier to understand.

```
function moveISS()
{
  fetch('http://api.open-notify.org/iss-now.json')
      .then(
          function(response)
          {
              if(response.status !==200 )
              {
                  console.log('Looks like there was problem with Open Notify
API' + response.status);
                  return;
              }
              response.json().then(function(data)
              {
                  var lat = data.iss_position.latitude;
                  var long = data.iss_position.longitude;
                  console.log(lat,long);
                  iss.setLatLng([lat,long]);
                  circle_icon.setLatLng([lat,long]);
                  map.panTo([lat,long],animate=true);
  global_iss = setTimeout(moveISS(),5000);
              });
          }
      ).catch(function(err)
      {
          console.log('Fetch Error ',err);
      });
}
moveISS();
```

settimeout : inorder to call (get Request) for every 5 seconds from our web app.

setLatLng : function to set the lat and long for the custom marker. In this case, we set that as iss icon.

panTo : similar to setView method (in fact they are same)