SENG202 – Software Engineering Project Workshop

2018

Handout - Resources for working with Interactive Map API's

Last updated: 10th September 2018

1. Overview

The purpose of this handout is to provide you with the necessary information required to begin implementation of a dynamic Map that displays route/path information between two or more markers. The map will be displayed within a JavaFX WebView and utilise HTML, Javascript and Java to perform the desired functionality.

Before we begin, an important note:
In this document, we'll be signing up to a free trial of Google Maps. The free trial gives $300 of credit to use over 12 months (more than enough for SENG202). However, you are required to provide billing details.

Google state that they will never auto-charge your account after the free trial ends so there is no risk. However, if you are uncomfortable with providing account details, Nokia (https://here.com) provide an alternative solution that works similarly (actually, almost identical) to the instructions in this document.

At the end of this tutorial, I'll explain how you can set up an account with Nokia (no card details necessary) and show you how similar the two API's are. It will still be worthwhile you reading through the Google tutorial and sample code though so that you get a feel for how it will work with Nokia.

2. Google API Key

To access Google API services, you must first obtain an API key. This is easily accomplished by navigating to the Google Maps API website located here (https://developers.google.com/maps/documentation/javascript/tutorial).

Follow these steps:

1. On the top menu bar, click 'Get Started.' A popup will ask which product(s) you want to enable. Choose the first 'Maps' option, and click 'Continue'

2. For the next step, select 'Create a new project' from the dropdown box. Give your project a name and click 'Next' (Note: I already have a developer account with Google. You may need to sign up yourself at developers.google.com)

3. You'll next be asked to provide billing details. This is still a free account, you will get $300 credit for free and Google state that they won't charge your card after this trial period until you upgrade to a paid account. Make sure you update your 'Account type' to state that you are an individual and not a business.

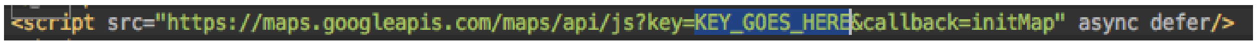
4. You'll then be prompted to enable your API's. Clicking 'Next' does this automatically and you'll finally be provided with your API key. Keep this for later.

This key will be required whenever your application sends requests to the Google API server and therefore it is important to protect from unwanted access. Google provides some useful information on this topic here - however, for this course these steps will not be necessary.

3. Download Project Files

You will find contained within the project two files named Controller.java and map.html. This is where the majority of the application logic is located. These two files are where you should be focusing your attention to understand how to bridge the gap between the html/javascript web page and the Java backend (not to be confused with a backend server). Feel free to explore the additional files to see how the rest of the application is formed.

You will need to enter your API key that you obtained earlier into the placeholder text between "...key=" and "&callback...", shown in Figure 1. This is located at the bottom of the map.html file.



```
<script src="https://maps.googleapis.com/maps/api/js?key=KEY_GOES_HERE&callback=initMap" async defer/>
```

Figure 1: API key placement.

4. Using the Application

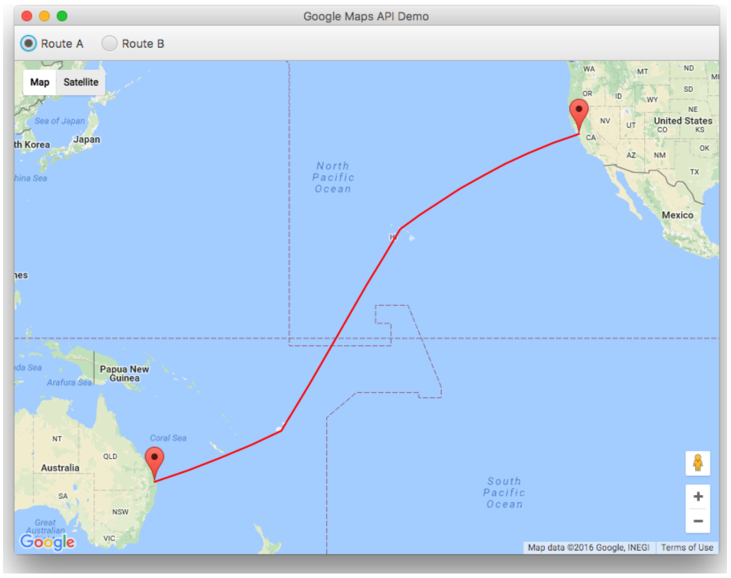


Figure 2: Map Application in all its glory.

Have a play around with the application. It will behave as a regular Google Map (as you might see on a website) - the important feature is the radio buttons (Route A & Route B) at the top of the window.

5. JavaFX WebView

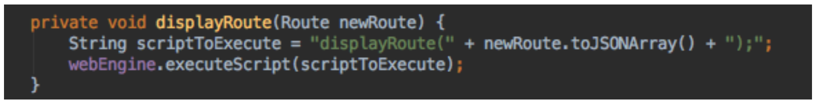
A WebView is an embedded lightweight browser with the ability to render HTML content from local or remote URLs. The WebView is just like any other node in JavaFX (e.g. BorderPane, HBox, VBox). In addition to this, the WebView contains a WebEngine object that provides the basic web page functionality

supporting user interactions such as navigating links and submitting HTML forms. The WebEngine handles one web page at a time and provides access to the Document Object Model (DOM) which is beneficial when executing Javascript commands.
While we are aware that you have **not** covered HTML or Javascript yet (SENG365), the code in the example should be fairly straight forward and if you are having any difficulty please contact the tutors.

Furthermore, the Google API documentation contains lots of useful information and sample code - it can be found here.

6. Understanding the Java/Javascript Bridge

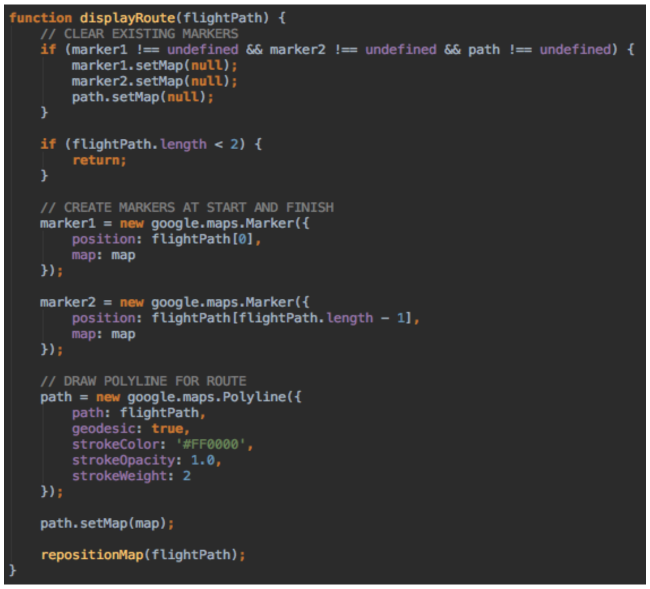
Within the method "displayRoute" located within Controller.java, there are two important points to note.



```
private void displayRoute(Route newRoute) {
    String scriptToExecute = "displayRoute(" + newRoute.toJSONArray() + ");";
    webEngine.executeScript(scriptToExecute);
}
```

Figure 3: Java display route function.

Firstly, the "scriptToExecute" defines the exact Javascript code that will be executed, in this case it will call the "displayRoute" Javascript function (contained within map.html) shown in Figure 3.



```
function displayRoute(flightPath) {
    // CLEAR EXISTING MARKERS
    if (marker1 !== undefined && marker2 !== undefined && path !== undefined) {
        marker1.setMap(null);
        marker2.setMap(null);
        path.setMap(null);
    }

    if (flightPath.length < 2) {
        return;
    }

    // CREATE MARKERS AT START AND FINISH
    marker1 = new google.maps.Marker({
        position: flightPath[0],
        map: map
    });

    marker2 = new google.maps.Marker({
        position: flightPath[flightPath.length - 1],
        map: map
    });

    // DRAW POLYLINE FOR ROUTE
    path = new google.maps.Polyline({
        path: flightPath,
        geodesic: true,
        strokeColor: '#FF0000',
        strokeOpacity: 1.0,
        strokeWeight: 2
    });

    path.setMap(map);

    repositionMap(flightPath);
}
```

Figure 4: Javascript display route function.

Secondly, "webEngine.executeScript(scriptToExecute)" is the method call that executes the script within the WebEngine, and therefore changes the content shown within the WebView.

7. Don’t want to provide billing information?

No worries, this is how you can set up an account with here.com:

1. Navigate to https://developer.here.com and click the ‘GET STARTED FOR FREE’ button.
2. Sign up for an account, and confirm your email address etc.
3. Log in to your new account
4. Again, click the ‘GET STARTED FOR FREE’ button.
5. After your project has been automatically created, under the ‘JavaScript/REST’ section, click the ‘Generate App ID and App Code’ button. You’ll want to save these for later.

That’s it. You’re ready to code.

Like Google, here has some very nice documentation. Everything you need can be found at:
https://developer.here.com/documentation/maps/topics/overview.html

8. Similarities between Google and Here

Navigate to the ‘Quick Start’ tab. If you followed along with the Google tutorial above, then it should look fairly familiar. By simply changing a few lines in the map.html file, we can change from using Google to using Here:

1. First we need to edit the head section of the HTML document. The head section is where we keep the pages meta-information. All we have to do here is:

- Change the title from ‘Google Map Demo’ to ‘Here Map Demo’ (although this isn’t compulsory)
- Import two scripts from the internet (see below)

Google

```
<head>
    <title>Google Map Demo</title>
    <meta name="viewport"
content="initial-scale=1.0">
    <meta charset="utf-8">
    <style>
        html, body {
            height: 100%;
            margin: 0;
            padding: 0;
        }
        #map {
            height: 100%;
        }
    </style>
</head>
```

Here

```
<head>
    <title>Here Map Demo</title>
    <meta name="viewport" content="initial-scale=1.0">
    <meta charset="utf-8">
    <style>
        html, body {
            height: 100%;
            margin: 0;
            padding: 0;
        }
        #map {
            height: 100%;
        }
    </style>
    <script src="http://js.api.here.com/v3/3.0/mapsjs-
core.js" type="text/javascript" charset="utf-8">
</script>
    <script src="http://js.api.here.com/v3/3.0/mapsjs-
service.js" type="text/javascript" charset="utf-8">
</script>
</head>
```

2. Remove the script import where we added our Google API key

```
<script src="https://maps.googleapis.com/maps/api/js?key=YOUR_KEY_HERE&callback=initMap"
async defer></script>
```

3. Initialise the Map. In the previous step, the script that we removed included a call back function, this states which function will be called when the page is first initialised. Don't worry about what this means, that's for the SENG365 team next year to explain, all you need to know is that Here doesn't provide a callback option, so our initialisation goes into the root of the script. We'll break down the initialisation into steps:

a) Delete the initMap() function.

b) Declare a new 'platform' variable. This is where you enter your API keys:

```
var platform = new H.service.Platform({
  'app_id': 'your app id',
  'app_code': 'your app code'
});
```

c) Initialise the maps object (this looks very similar to how Google does it).

```
// Obtain the default map types from the platform object:
var defaultLayers = platform.createDefaultLayers();

// Instantiate (and display) a map object:
var map = new H.Map(
  document.getElementById('map'),
  defaultLayers.normal.map,
  {
    center: {lat: 39.144684, lng: -84.510079}, zoom: 15
  });
```

This is what all the changes look like together:

Google

```
function initMap() {
  map = new google.maps.Map(
    document.getElementById('map'),
    {
      center: {
        lat: 39.144684,
        lng: -84.510079
      },
      zoom: 15
  });

  new google.maps.Marker({
    position: {
      lat: 39.144684, lng: -84.510079
    },
    map: map, title: 'some title’
  });
}
```

Here

```
var platform = new H.service.Platform({
  'app_id': 'X1MlU7yHYG0XMR7n91TJ',
  'app_code': '9IS-gs0nr1WwTese3Z1QAQ'
});

// Obtain the default map types from the platform
object:
var defaultLayers = platform.createDefaultLayers();

// Instantiate (and display) a map object:
var map = new H.Map(
  document.getElementById('map'),
    defaultLayers.normal.map,
    {
      center: {lat: 39.144684, lng: -84.510079},
      zoom: 15
    }
);
```

Now when running your application, you will be presented with the Here map instead of the Google map.

We won’t cover the rest of the application here (mapping the flight paths) because the logic is the same, the documentation will show you how to plot a Here Polyline instead of a Google Polyline, and a Here marker over a Google marker. If you have any problems, contact a tutor and we’ll be happy to help.