Programming Manual

CREATING AN 3 DIMENSIONAL
MAP VIEWER
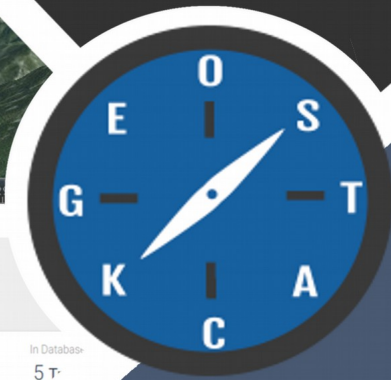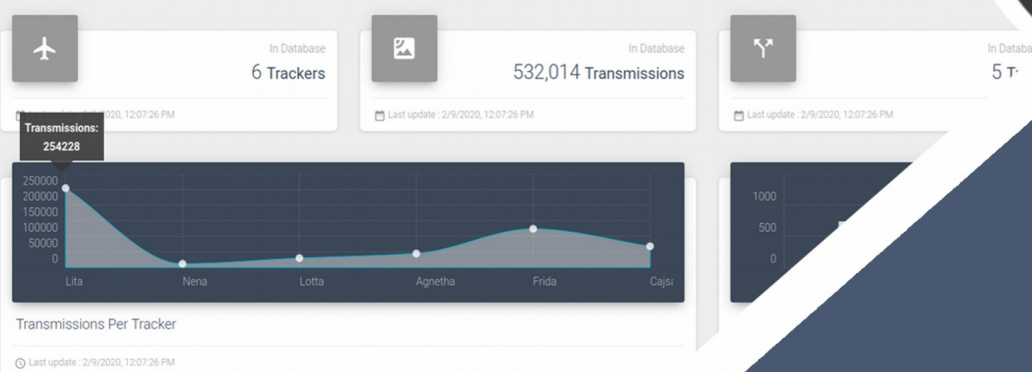
Version : 1.0

Date : 08-04-2020

Author : The GeoStack Project

Purpose of this document

This programming manual serves as an extension for the following documents:

1) **Cookbook: Creating the GeoStack Course VM:**

    The datastores, tools and libraries used during this programming manual are installed and created in the cookbook: Creating the GeoStack Course VM.

2) **Cookbook: Creating a basic web application:**

    The base application of this Dataset Dashboard has been created during the cookbook: Creating a basic web application.

3) **Cookbook: Data modeling in MongoDB using MongoEngine:**

    The data used during this cookbook, is modeled, indexed and imported in the cookbook: Data modeling in MongoDB using MongoEngine.

4) **Programming manual: Creating the Python-Flask web application:**

    The middleware that will be used during this programming manual is created in the programming manual: Creating the Python Flask web application.

If you have not read these documents yet, please do so before reading this document.

The purpose of this programming manual is to create an 3D map viewer application using the AngularJS JavaScript framework and the JavaScript framework Cesium. This application is an extension of our Angular base application and the 2D Map Viewer.

The reason this application is an extension of the 2D Map Viewer is because the 3D Map Viewer uses the same functions in order to add and remove items, add and remove layers and selecting item data. We don't want to write the same code as we did in the 2D Map Viewer so we are going to remove the unused code and edit the existing code according to the needs of the 3D Map Viewer.

The Angular apps will perform API calls to our Flask application and our Flask application will then retrieve the requested data via queries, performed on our datastores. The results are then returned to our Angular applications.

This programming manual serves as a guideline for the steps you have to perform to create a 3D Map Viewer using Cesium and visualize the data retrieved by the Flask-API.

During this programming manual the code is explained using the inline comments in the source code located in the folder: "POC". It's highly recommended to use the source code provided in this folder when creating the web application yourself.

NOTE: Sometimes you will notice that in the code which you have to create some functions do not exist yet. Don't worry about this since they will be added later on during the programming manual!

Table of Contents

1.Introduction

1.1 Getting ready

This application is basically the same as the 2D Map viewer so we can copy this application and start creating the 3D map viewer from there. After we copied the 2d-map-viewer folder, we need to change some names and titles. We start by changing the name of the folder we just copied from 2d-map-viewer to 3d-map-viewer, as shown in the image below.
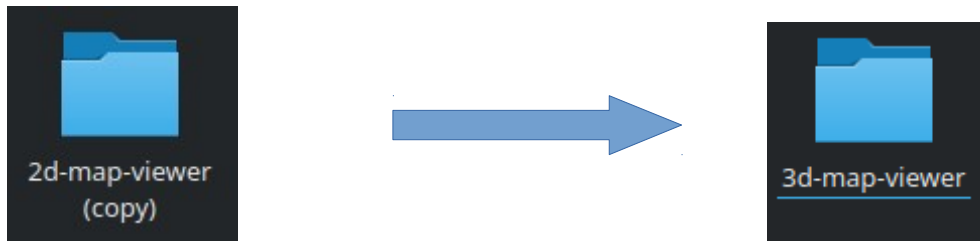

We also need to edit the project name: "2d-map-viewer" to "3d-map-viewer". If you are using the code-editor Atom, this is done by performing the following steps:

1) In the edit press the keys Ctrl + shift + f in the Atom editor.

2) In the screen that pops up enter: "2d-map-viewer" in the find section and "3d-map-viewer" in the replace section, as shown in the illustration below. Then click on find all.
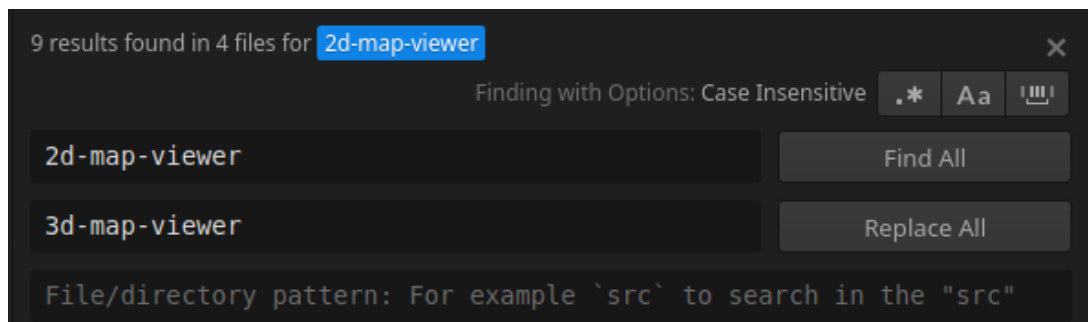
3) Click on replace all and on the save button in the screen that pops up.
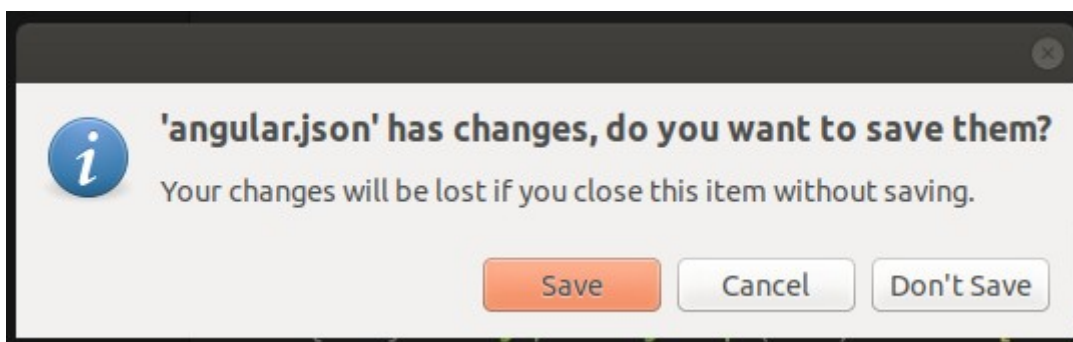
4) In the file: index.html located in the folder 3d-map-viewer/src, replace the title from 2D Map Viewer to 3D Map Viewer.

5) In the file: sidebar.component.html, located in the folder src/app/components/sidebar/, change the text: "3D Map Viewer" to "3D Map Viewer".

6) In the file: sidebar.component.ts, located in the folder src/app/components/sidebar/, change the title of the route related to the 2D Map Viewer from 2D Map OpenLayers to 3D Map Cesium as shown in the illustration below:

```
export const ROUTES: RouteInfo[] = [
    { path: '/map-page', title: '3D Map Cesium', icon: 'map', class: '' },
    { path: '/base-page', title: 'Base Page', icon: 'map', class: '' },
];
```

1.2 Adding the JavaScript framework Cesium

Since we don't need the geospatial JavaScript framework OpenLayers anymore we can remove the OpenLayers folder which can be found in the folder: "3d-map-viewer/src/assets/geospatial-frameworks/".

Deleting the OpenLayers framework can done by opening the 3d-map-viewer folder in Atom, finding the OpenLayers Folder, right clicking the folder and selecting delete as shown in the illustration below:
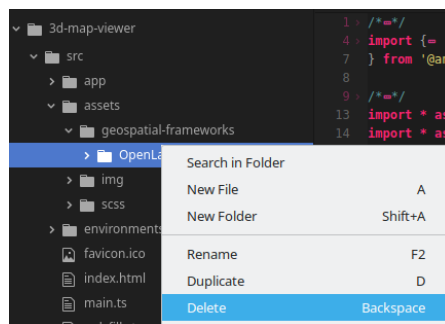
Now that we have removed the JavaScript Framework: "OpenLayers" we should add the JavaScript Framework:"Cesium".

Just like with OpenLayers; Adding the geospatial framework to our Angular application can be done in 2 ways which are as follows:

1) **Installing the NPM Package: "cesium":**

    This is the first technique which you can use to install Cesium in your application. During this programming manual we will not be using this technique. If you want to read up on using this technique you should visit the following URL: https://cesium.com/blog/2018/03/12/cesium-and-angular/

2) **Downloading the Cesium source code:**

    During this programming manual we are going to use this technique. We do this because, from the version control point of view, this method is the best method since there are no files added to the Node_Modules folder of the application. Using this technique we are going to add the geospatial framework as static files in the assets folder of our application. This enables us to easily switch to a newer or older version of the geospatial framework.

First we want to download the Cesium source code from the Cesium website which is located on the following URL:

https://cesium.com/downloads/

When navigating to the URL mentioned above you should be greeted with a green download button as shown in the illustration below:
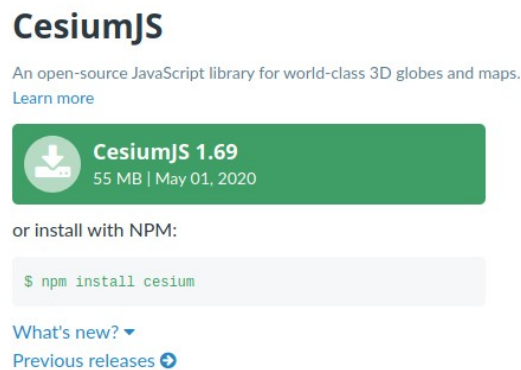
Click on the download button and then select Save File and click on Ok as shown in the illustration below:
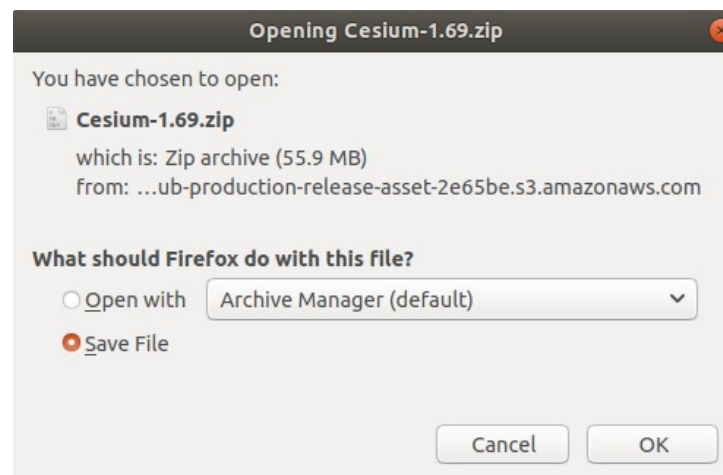
After a few seconds you will end up with a ZIP folder in your downloads folder as shown in the illustration below:


Open the ZIP file. This ZIP contains all the files related to the JavaScript framework Cesium. Since we only need the Cesium source code we should click on the build folder as shown in the illustration below:
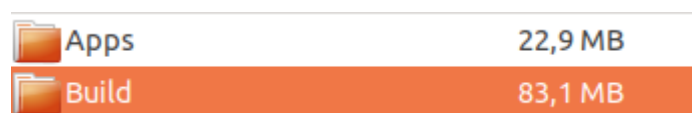
Once we are in the build folder we now have to copy the folder called: "Cesium" to the geospatial-frameworks folder in our 3d-map-viewer application ("3d-map-viewer/src/assets/geospatial-frameworks/")

This can be done by dragging and dropping the Cesium folder as shown in the illustration below:
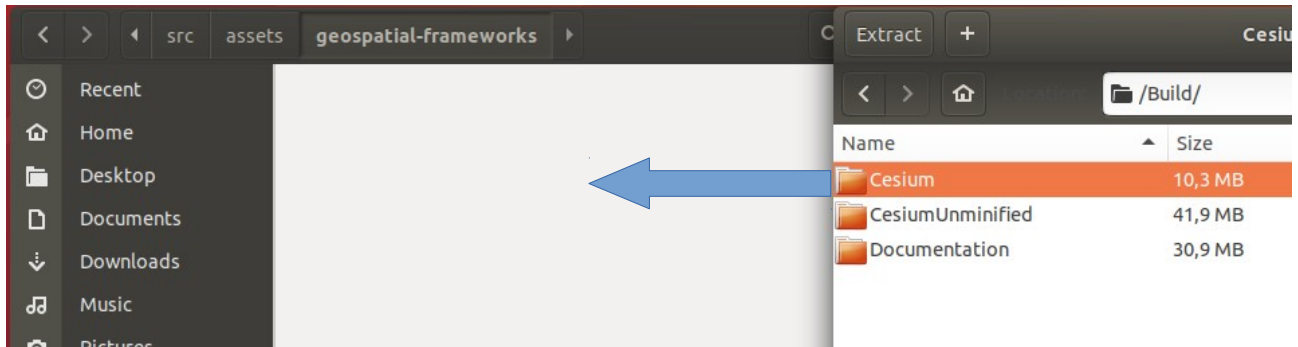
Now that we have the Cesium source code in our Angular application we need to edit the index.html file of our application so that the application knows where Cesium is located.

So let's open the index.html file which is located in the folder: "3d-map-viewer/src/".

We currently have the references to the OpenLayers framework in the <head> tag of the index.html. So let's change it according to the illustration shown below:

```
<!--Here we add the reference to the OpenLayers style sheet-->
<link rel="stylesheet" href="/assets/geospatial-frameworks/OpenLayers/ol.css"/>

<!--Here we add the reference to the OpenLayers javascript code-->
<script src="/assets/geospatial-frameworks/OpenLayers/ol.js"></script>
```

```
<!--Here we add the reference to the Cesium style sheet-->
<link rel="stylesheet" href="/assets/geospatial-frameworks/Cesium/Widgets/widgets.css"/>

<!--Here we add the reference to the OpenLayers javascript code-->
<script src="/assets/geospatial-frameworks/Cesium/Cesium.js"></script>
```

That's it! Now we can use the JavaScript framework Cesium throughout our application. In the next section you will learn how to create a free Cesium ION account. This step is optional but recommended since it will extend the functionalities which you can add to your 3D Map Viewer later

 In the next section we are going to cleanup the map.component.ts file to remove the code from the 2D Map Viewer application which we don't need for the 3D Map viewer application.

1.3 Creating a free Cesium ION account

This section describes why and how you can create a Cesium ION account. As mentioned above; this step is optional and is not required to create the 3D Map Viewer application. Creating a Cesium ION account will however extend the base functionalities of the 3D Map Viewer application such as using the Cesium World Terrain Layer and the Bing Aerial satellite images.

"Cesium ion is a robust, scalable, and secure platform for 3D geospatial data. Upload your content and Cesium ion will optimize and tile it for the web, serve it up in the cloud, and stream it to any device.

With Cesium ion, you'll have access to our curated 3D content including Cesium World Terrain and Bing Maps imagery. Combine these assets with your own data to provide more context or see it on a 3D map of the world.

The Cesium ION tools are built for 3D geospatial—and tied to precise map coordinates. We first built Cesium to track objects in space, and this incredible precision is baked into our platform." Source: "www.cesium.com, 2020"

To find more information related to the capabilities of Cesium ION you should visit the following URL:

https://cesium.com/cesium-ion/

On this website you will also find the option to register an account as shown in the illustration below:
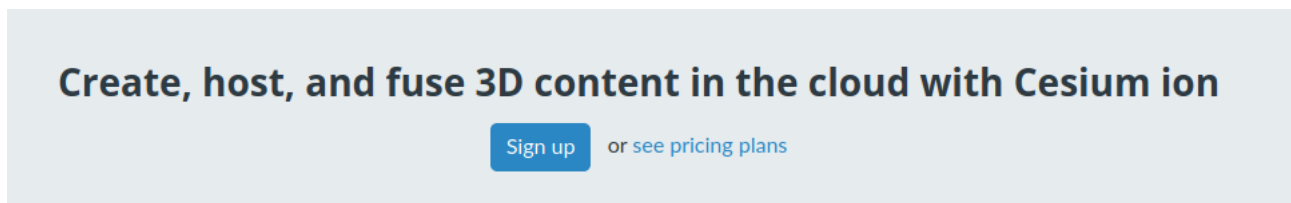
Click on the Sign up button and fill in your information as shown in the illustration below:
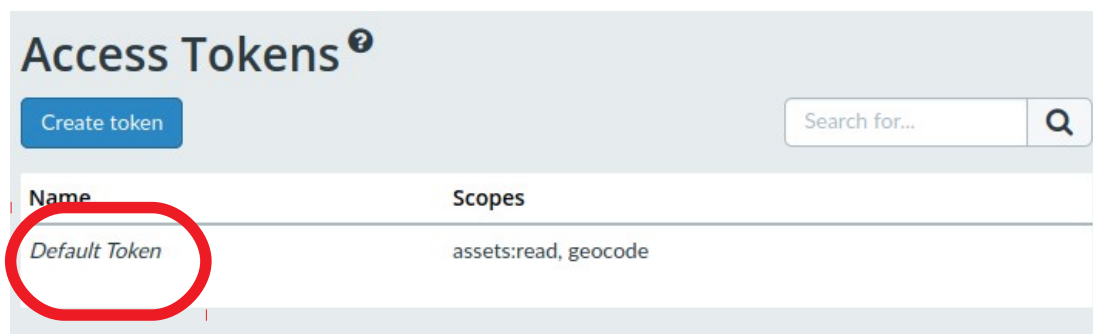
NOTE: Make sure to fill in you own information!

Once you are registered and logged in you should be greeted with the same screen as shown in the illustration below. Click on the button: "Access Tokens" encircled in red.
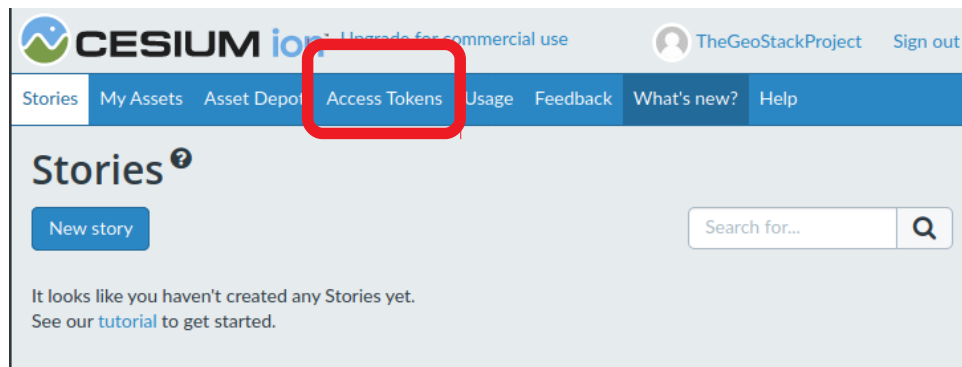
In the next window that shows you should click on the default token entry as shown in the illustration below:

Now copy the token by clicking on the button encircled in red in the illustration below. Save the token somewhere in a text file. We are going to use it later!

NOTE: In the illustration below the token is blurred for privacy reasons!
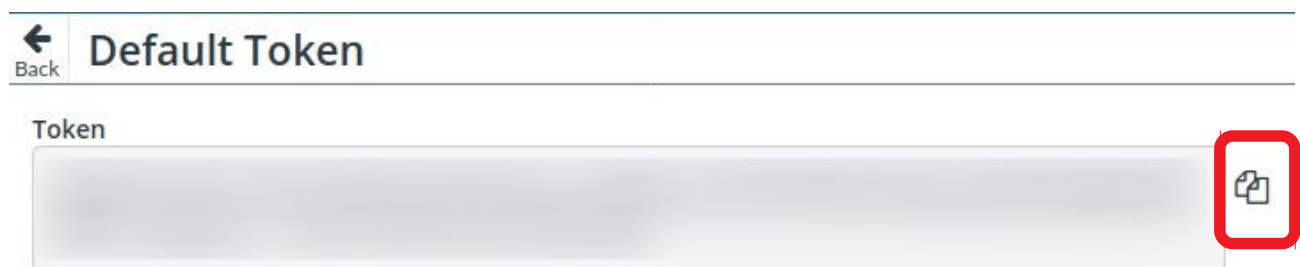
That's it! Now you have registered a free Cesium ION account. NOTE: The free Cesium ION account cannot be used for Commercial purposes as shown in the illustration below:

For more information visit this URL: https://cesium.com/pricing/

In the next section we are going to cleanup the map.component.ts file to remove the code from the 2D Map Viewer application which we don't need for the 3D Map viewer application.

2 Cleaning up the map.component.ts file

Because the 3D Map Viewer is an extension of the 2D Map Viewer we can keep most of the code which we created during the manual: "Creating an 2 Dimensional Map Viewer".

This section describes what files and code you should remove and keep in order to turn the 2D Map Viewer into the 3D Map Viewer. So let's open the map.component.ts file located in the folder: "3d-map-viewer/src/app/page/map-page/".

To make it easier to remove code we can fold the code so that only to first lines of the module imports, functions etc. are shown.

This is done by pressing Ctrl+ Alt + Shift + [on your keyboard. This will result in the following map.component.ts file:

```
1  > /*=*/
4  > import {=
7    } from '@angular/core';
8
9  > /*=*/
13   import * as Chartist from 'chartist';
14   import * as tooltip from 'chartist-plugin-tooltips'
15
16 > /*=*/
21 > import {=
23   } from 'src/app/services/map.service'
24 > import {=
26   } from 'src/app/services/crane.service'
27 > import {=
29   } from 'src/app/services/port.service'
30
31 > /*=*/
36   declare const ol: any;
37
38 > /*=*/
42 > export class Item {=};
83
84 > /*=*/
93 > @Component({=})
98 > export class MapComponent implements OnInit {=};
21
```

2.1 Cleaning the unused imports
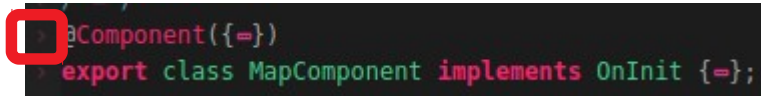
We want to start of by removing the module imports which we don't need anymore. The following module imports have to be removed:

➔ The Chartist module and Tooltip modules, since we don't need an elevationProfile in our 3D Map Viewer.

➔ The PortService import, since we don't need the World Port Index to be displayed in our 3D Map Viewer application.

So let's remove the imports by selecting the code and pressing backspace as shown in the illustration below:

```
/*=*/
import * as Chartist from 'chartist';
import * as tooltip from 'chartist-plugin-tooltips'
```

The next thing we need to do is removing the PortService from the providers entry in our MapComponent metdata. So open the Component metadata by clicking on the arrow (encircled in Red) next to the line @Component({}) as shown in the illustration below:
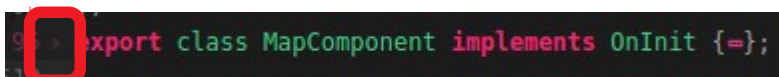
Now remove the PortService from the providers list as shown in the illustration below:

The last thing we need to do in order to remove the unused imports is removing the PortService from the MapComponent class constructor. So click on the arrow (encircled in red) next to the line : "export class MapComponent implements OnInit" to unfold the MapComponent class coe as shown in the illustration below:

Now find the MapComponent class constructor and edit it according to the illustration below:

That's it! Now we have removed the unused imports from our 3D Map Viewer map.component.ts file. In the next section we are going to remove the unused global variables.

2.2 Cleaning the unused global variables

Now let's remove some global variables which we are not going to need for our 3D Map Viewer application. The global variables which we are going to remove are as follows:

➔ MapLayer, since we are going to create a new Map Layer which is usable with Cesium;

➔ SeaLayer, since we do not need the OpenSeaMap Layer in the 3D Map Viewer application;

➔ LayerStyles, since layer styling works differently in Cesium;

➔ colorList, widthList, lineTypeList and StyleDict, again since layer styling works differently in Cesium.;

➔ elevationProfile, since we don't need the elevation profile in the 3D Map Viewer application;

➔ elevationProfileOpen, again since we don't need the elevation profile in the 3D Map Viewer;

➔ portLayer, since we are not going to use the World Port Index dataset in the 3D Map viewer.

Now that you know what global variables we are going to remove we should start removing them. In the illustrations below the global variables encircled in red need to be removed:

```
/*= */
public map: any;

/*= */
public mapProviders: Map < any, any > = new Map();

/*= */
public mapLayer: any = new ol.layer.Tile({=});

/*= */
public seaLayer: any = new ol.layer.Tile({=});

/*= */
public items: Item[] = [];

/*= */
public selectedItems: Item[] = [];

/*= */
public activeItem: Item = new Item();

/*= */
public layerStyles: any = {=};

/*= */
public dateRange: any = [0, 0];
```

```
/*= */
public countryList: Map < string, Number[][] > = new Map([=]);

/*= */
public colorList: Map < string, string > = new Map([=]);

/*= */
public widthList: Map < string, number > = new Map([=]);

/*= */
public lineTypeList: Map < string, number[] > = new Map([=]);

/*= */
public styleDict: any = {=};

/*= */
public elevationProfile: any;

/*= */
public elevationProfileOpen: boolean;

/*= */
public portLayer: any;
```

2.3 Cleaning unused functions

Now that we have removed the unused global variables we can start removing the unused functions. The functions we are going to remove are as follows:

➔ createOpenLayersMap(), since we are going to create a new function which creates a Cesium map;

➔ setMapProvider(), since we are going to create a new function which changes map providers;

➔ zoomToLocation(), since we are going to create a new function which zooms to the start location of an item;

➔ addOverlays(), since we are do not have to create overlays in our 3D Map Viewer application;

➔ setDynamicOverlays() and setStaticOverlays(), since we are not going to use overlays;

➔ toggleLayer() and toggleOverlay(), since we don't need layer toggling in our 3D Map Viewer;

➔ setLayerStyle(), since we are not going to use layerStyling in the 3D Map Viewer;

➔ animateRoute() and clearAnimation(), since animating items is done differently in Cesium;

➔ createElevationProfile() and loadElevationData(), since we don't need an elevation profile in our 3D Map Viewer;

➔ createPortLayer(), since we are not going to use the World Port Index dataset in our 3D Map Viewer.

Now that you know which functions are going to be removed we can start removing them. In the illustrations below the functions encircled in red need to be removed:

```
/*= */
constructor(private _MapService: MapService,=) {}

/*= */
ngOnInit() {=};

/*= */
getMapProviders(): void {=};

/*= */
createOpenLayersMap(): void {=};

/*= */
setMapProvider(providerKey): void {=};

/*= */
addItem(itemId, itemName, itemType, itemRouteLength, itemTimeColumn, itemDTG): void {=};
```

```
/*= */
getItems(): void {=};

/*= */
timeConverter(timestamp): string {=};

/*= */
selectItem(item: Item): void {=};

/*= */
getInitalItemData(item: Item): void {=};

/*= */
loadItemData(data: any[]): void {=};

/*= */
zoomToLocation(): void {=};

/*= */
addLayerGroup(item: Item): void {=};

/*= */
setLayerGroup(groupKey: string): void {=};

/*= */
addOverlays(): any[] {=};

/*= */
setDynamicOverlays(item: Item): void {=};

/*= */
setStaticOverlays(item: Item): void {=};

/*= */
removeItem(item: Item): void {=};

/*= */
getItemDataByDTG(item: Item, dtg_s, dtg_e): void {=};

/*= */
getDTGEvent(id: string, $event): void {=};
```

```
/*= */
removeLayerGroup(layerGroupKey: string): void {=};

/*= */
getItemDataByAmount(item: Item, amount): void {=};

/*= */
getItemDataByCountry(item: Item, coords: Number[][]): void {=};

/*= */
toggleLayer(layerType: string): void {=};

/*= */
toggleOverlay(overlayType: string): void {=};

/*= */
setLayerStyle(layerType: string): void {=};

/*= */
animateRoute(): void {=};

/*= */
clearAnimation(): void {=};

/*= */
createElevationProfile(): void {=};

/*= */
loadElevationData(): void {=};

/*= */
createPortLayer(ports){=};
```

The last thing we need to do before this section is finished is clearing the code inside the functions: "addLayerGroup()" and "setLayerGroup()". So go to these functions in the map.component.ts file and remove all the code inside these functions so that only the following remains:

```
addLayerGroup(item: Item): void {
};

setLayerGroup(groupKey: string): void {
};
```

That's it! Now we have removed all the functions which we are not going to need anymore. In the next section we are going to edit some existing functions according to the needs of our application.

2.4 Editing the existing code

Now that we have removed the unused imports, variables and functions we need to update some code and functions. Let's start off by editing the constant which was declared at the top of the map.component.ts file. At this point we have declared a constant called:"ol" which was used to be able to use build-in OpenLayers functions in our 2D Map Viewer application. We want to use Cesium instead of OpenLayers so let's edit the constant according to the illustrations below:
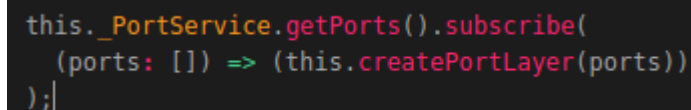

Now we can use the syntax: "Cesium.{a build in function}" to use the functionalities of the JavaScript framework Cesium.

Next up is editing the function: "ngOnInit()". We need to remove the line that triggers the function: createOpenLayersMap() since we removed that function earlier. So let's edit the function according to the illustrations below:
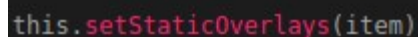
Next up is editing the function: "getItems()". We have to remove the code which was used to obtain the World Port Index data. We do this because we are not going to create a PortLayer in the 3D Map Viewer application. So let's remove the code shown in the illustration below:

Next up is editing the function: "selectItem()". We want to remove the line of code which was used to set the Static Overlays in our 2D Map Viewer application. We do this because we are not going to use overlays in the 3D Map Viewer. So let's remove the line shown in the illustration below:

The next function we are going to edit is the function: "loadItemData()". We want to edit one line related to converting coordinates in a format which was understandable for OpenLayers.

The line we want to edit is found in the foreach loop which loops through all the datapoints from the list of data that is passed on the function call.

So let's edit the line according to the illustrations below:

```
item.coordinateList.push(
  ol.proj.fromLonLat(row.geometry.coord.coordinates)
);
```

```
item.coordinateList.push(row.geometry.coord.coordinates);
```

We also want to remove the 2 lines related to setting the static overlays and creating the elevation profile. This is done by removing the lines of code shown in the illustration below:

```
this.setStaticOverlays(item)

this.createElevationProfile();
```

Next up is the function: "removeItem()". We need to remove the lines of code which we used to clear the running animation, toggle the overlays of and setting the static overlays. We start of by removing the line of code related to clearing the animation. This is done by editing the first line in the function:"removeItem()" according to the illustrations below:

```
this.activeItem.id == item.id ? (this.clearAnimation(),
    this.selectItem(this.selectedItems.values().next().value)) :
  null;
```

```
this.activeItem.id == item.id ? (
    this.selectItem(this.selectedItems.values().next().value)) :
  null;
```

We also need to remove a few lines of code which were used to remove the layers from the OpenLayers map in our 2D Map Viewer. The lines which you have to remove are shown in the illustration below:

```
item.layerGroups.forEach(layerGroup => {
  for (let [key, value] of Object.entries(layerGroup)) {
    this.map.removeLayer(value['layer'])
  }
});
```

The last line we need to remove the last line of code in the function: "removeItem()". The line which you have to remove is shown in the illustration below:

```
this.selectedItems.length == 0 ? this.toggleOverlay('all') :
    this.setStaticOverlays(this.activeItem)
```

The last function we need to edit is the function: "removeLayerGroup()". In this function we also need to remove the line related to clearing any running animations. The line of code which you have to remove is shown in the illustration below:

```
this.clearAnimation()
```

The last thing we need to remove in the function:"removeLayerGroup()" is the code related to removing all the layers from the OpenLayers map in our 2D Map Viewer. The lines that you have to remove are shown in the illustration below:

```
for (let [key, value] of Object.entries(groupToRemove)) {
    this.map.removeLayer(value['layer'])
}
```

That's it! Now you have removed the unused code and edited the functions which we are going to use in the 3D Map Viewer application. If you run the application you should not encounter any errors.

To run the the application you should navigate to the root directory of the 3D Map Viewer by running the following command:

cd ~/Geostack/angular-apps/3d-map-viewer

And then start the Angular Live Development server by running the following command:

sudo npm start

If you run the application you should not encounter any errors as shown in the illustration below:

```
10% building 3/3 modules 0 active  ｢wds｣: Project is running at http://localhost:4200/webp
i ｢wds｣: webpack output is served from /
i ｢wds｣: 404s will fallback to //index.html

chunk {main} main.js, main.js.map (main) 147 kB [initial] [rendered]
chunk {polyfills} polyfills.js, polyfills.js.map (polyfills) 268 kB [initial] [rendered]
chunk {runtime} runtime.js, runtime.js.map (runtime) 6.15 kB [entry] [rendered]
chunk {styles} styles.js, styles.js.map (styles) 2.31 MB [initial] [rendered]
chunk {vendor} vendor.js, vendor.js.map (vendor) 4.87 MB [initial] [rendered]
Date: 2020-05-27T15:18:13.034Z - Hash: c2f4929c06eaa57308c7 - Time: 11793ms
** Angular Live Development Server is listening on localhost:4200, open your browser on ht
i ｢wdm｣: Compiled successfully.
```

When opening the application in your browser you will be greeted with the page shown in the illustration below:
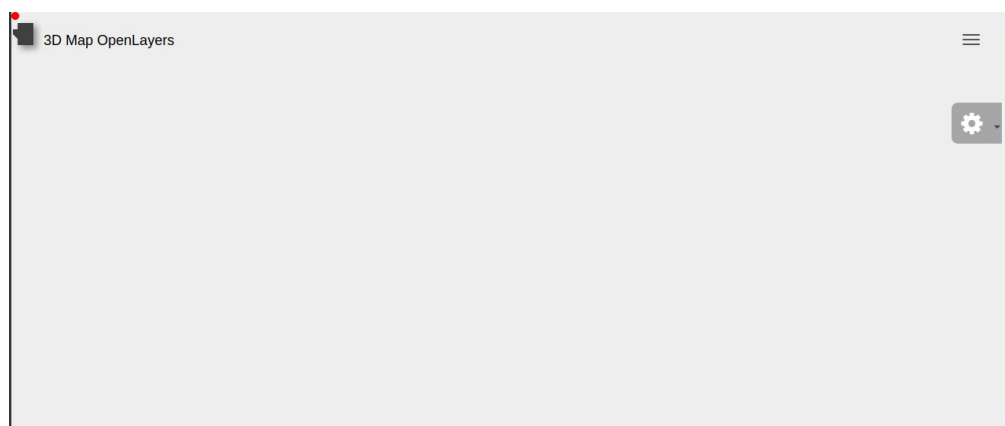
3 Cleaning up the map.component.html file

Now that we have cleaned the map.component.ts file we also need to remove some code from the map.component.html file. We do this because we will not be using settings such as changing layer styles, animation routes, toggling overlays etc. So let's open the HTML file in Atom.

Now let's fold the code like we did in the map.component.ts file. Folding code is done by pressing the key combination Ctrl + Alt + Shift + [on your keyboard. After you folded the code the map.component.html file will look the same as shown in the illustration below.

NOTE: You will probably do not have to code comments (the green text) shown in the illustration. The code in the folder: "POC" contains these comments.

```html
<!-- Here we add the empty div elements related to the overlays. -->
<div id="geomarker" style="background-color: red; height: 10px; width: 10px; border-radius: 100px;"></div>
<div id="geomarkerInfo" class="hint--no-animate hint--right hint--always" data-hint=""></div>
<div id="startmarkerInfo" class="hint--no-animate hint--right hint--always" data-hint=""></div>
<div id="endmarkerInfo" class="hint--no-animate hint--left hint--always " data-hint=""></div>


<!--
Here we create a div element to which the Cesium map will be assigned
We give the div a height of 100vh (Full screen height) and a width of 100%
which is the full width of the screen. -->
<div id="map" style="height: 100vh; width: 100%;"></div>


<!--
In this div element all the logic related to the map settings is added.
The class: "fixed-plugin" make sure the settings menu is shown on the web page.
-->
<div class="fixed-plugin" id="fixed-plugin">⋯
</div>



<!-- Here we definean angular if statement which determines if the length
of the selectedItems list is bigger than 0. If this is the case the following
HTML code will be executed. -->
<div *ngIf="selectedItems.length> 0">⋯
</div>
```
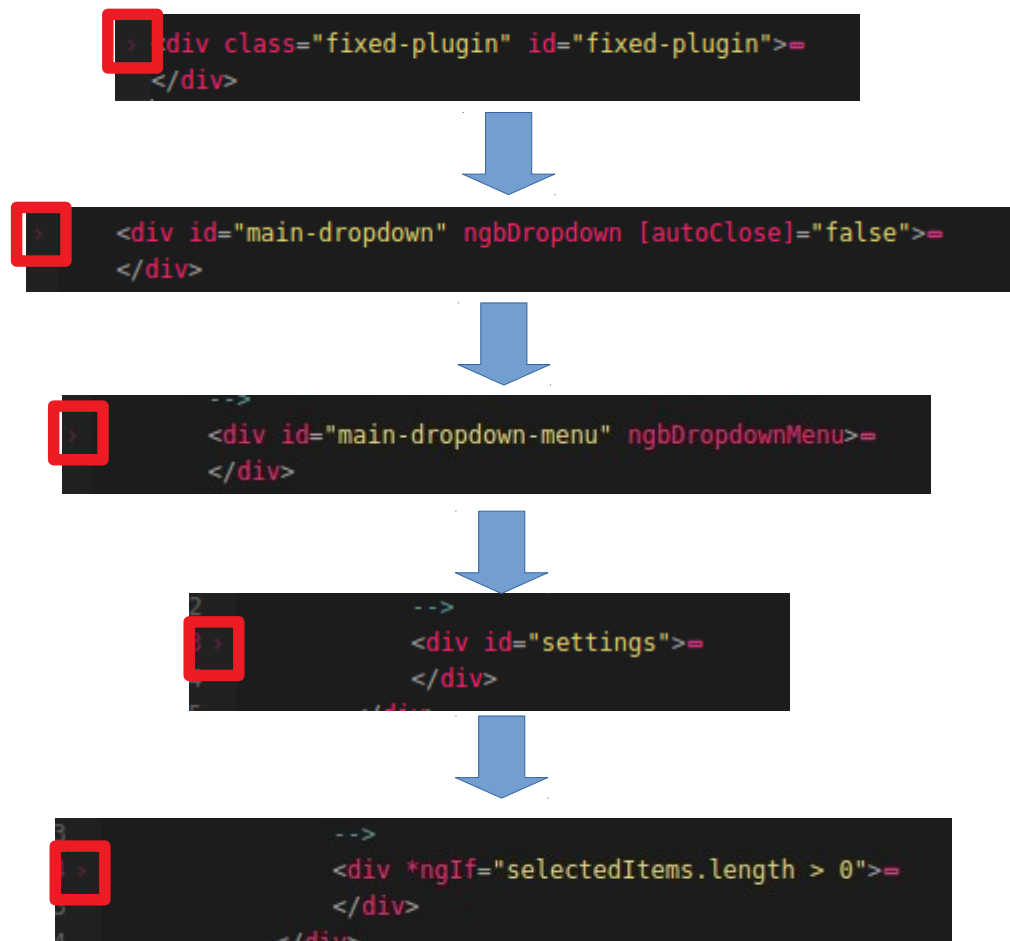
Now let's start of by removing the 4 div elements at the top of the file. These div elements were used for the overlays in the 2D Map Viewer. We remove them because we are not going to use overlays in the 3D Map Viewer. The code that you have to remove is shown in the illustration below:

```html
<!-- Here we add the empty div elements related to the overlays. -->
<div id="geomarker" style="background-color: red; height: 10px; width: 10px; border-radius: 100px;"></div>
<div id="geomarkerInfo" class="hint--no-animate hint--right hint--always" data-hint=""></div>
<div id="startmarkerInfo" class="hint--no-animate hint--right hint--always" data-hint=""></div>
<div id="endmarkerInfo" class="hint--no-animate hint--left hint--always " data-hint=""></div>
```

The next code that we want to remove is the last div element (related to the elevation profile) in the HTML page. The code which you have to remove is shown in the illustration below:

```html
<!-- Here we definean angular if statement which determines if the length
of the selectedItems list is bigger than 0. If this is the case the following
HTML code will be executed. -->
<div *ngIf="selectedItems.length> 0">⋯
</div>
```

The last thing we need to remove in the map.component.html file are the div elements related to the layer toggling, layer styling and the animation. So let's find these div elements by clicking on the red arrows (to unfold the code) next to the line of code shown in the illustrations below:

```
<div class="fixed-plugin" id="fixed-plugin">=
    </div>
```

```
<div id="main-dropdown" ngbDropdown [autoClose]="false">=
    </div>
```

```
    -->
        <div id="main-dropdown-menu" ngbDropdownMenu>=
        </div>
```

```
    -->
            <div id="settings">=
            </div>
```

```
    -->
            <div *ngIf="selectedItems.length > 0">=
            </div>
        </div>
```

Now find and remove (by highlighting them and pressing delete on you keyboard) the div elements (and the code inside the elements) shown in the illustration below:

NOTE: You will probably do not have to code comments (the green text) shown in the illustration. The code in the folder: "POC" contains these comments.

```
<div id="toggleSelection">=
</div>

<!--
In this div element we add the logic related to changing layer styles
-->
<div id="styleSelection">=
</div>

<!--
In this div element we add the logic related to animating the route.
-->
<div id="animationSelection"  style="margin-top:55px;">=
</div>
```

That's it! Now when you refresh the application, open the settings menu and select an item you should have see the same as shown in the illustration below:
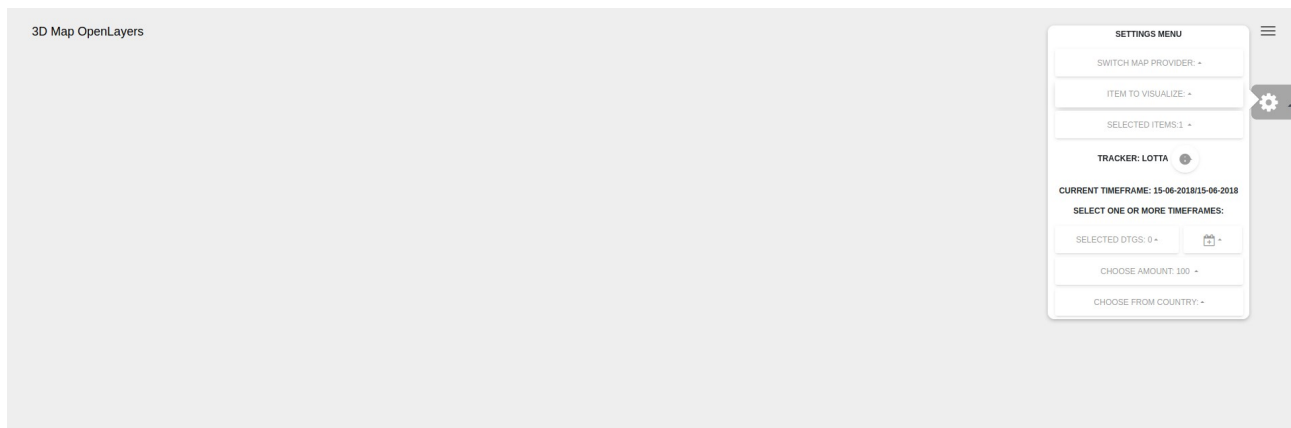
At this point we are finally done with removing and editing existing code. Now we are ready to start adding the functionalities in order to create the 3D Map Viewer.

4 Creating the Map Component functionalities

At this point we are finally done with removing and editing existing code. Now we are ready to start adding the functionalities in order to create the 3D Map Viewer.

We are going to start of by creating the Cesium Map instance and adding the base map, which was created in section 5.5.3: "Installing the TileStache Tileserver", and the elevation maps which were created in section 5.5.4.2: "Rendering Digital Terrain Models for Cesium" of the cookbook: "Creating the GeoStack Course VM".

How this is done is shown in the next section.

4.1 Creating the Cesium Map instance (Cesium Viewer)

Before creating the Cesium Map instance we need to add 2 global variables to our map.component.ts file. These global variables are as follows:

1) mapTileLayer, which is the layer that is going to contain the OpenStreetMap tiles served by our TileStache Tileserver.

   This is the variable to which the baseLayer containing the tiles of the map is assigned. We assign the elevationMaps by creating a new Cesium OpenStreetMapImageryProvider and passing the location of the TileStache Tileserver running behind the NGINX webserver as URL.

   We set the local OpenStreetMap tiles as default WMS. So when the application is loaded, the local OpenStreetMap tiles are loaded as well.

2) MapTerrainLayer, which is the layer that is going to contain the Cesium Terrain Files (Elevation map) served by our Cesium Terrain Server.

   This is the variable to which the baselayer containing the elevationMaps is assigned. We assign the elevationMaps by creating a new CesiumTerrainProvider and passing the location of the Cesium Terrain Server running behind the NGINX webserver as URL.

   We set the local Hamert DSM files as default Terrain files. So when the application is loaded, the local (DTM) terrain files of the Hamert are loaded as well.

Let's start of with creating the mapTileLayer. This is done by adding the following code below the global variable:"map" and above the global variable:"mapProviders":

```
public mapTileLayer:any = new Cesium.OpenStreetMapImageryProvider({
  url : 'http://localhost/tiles/openstreetmap-local/'
});
```

Now let's add the mapTerrainLayer. This is done by adding the following code below the global variable: "mapTileLayer" which we created above:

```
public mapTerrainLayer:any =  new Cesium.CesiumTerrainProvider({
  url : 'http://localhost/terrain/M_52EZ2'
});
```

Now let's add a function called: "createCesiumMap()". This function is used to create the Cesium Map instance which is also known as the Cesium Viewer. The function is triggered in the function: "ngOnInit()" to make sure that the Cesium Map (Viewer) is created when the MapComponent is loaded.

The map instance will be created in the HTML div element with the id:'map'. This div element is defined in the HTML layout of the MapComponent which can be found in the file: "map.component.html".

The following steps are executed when the function is triggered:

1) Trigger the function:"getMapProviders()" which is used to obtain all the available WMS's (WebMapServers) which are defined in our TileStache configuration file.

2) Optional: Assign your Cesium ION Token key to the Cesium instance. (This is the token which you copied in section 1.3 of this document)

3) Create a new Cesium Map instance (Viewer) to which we assing the global variable:"mapTileLayer" as imageryProvider (tile layer) and the global variable:"mapTerrainLayer" as terrainProvider (Elevation map layer)

4) set allowDataSourcesToSuspendAnimation to False to make sure that when an animation is started it will not stop when a Terrain file or Tile is loaded.

Now that you know what the function is used for we can start coding it. This is done by adding the following code below the function: "getMapProviders()":

NOTE: This function is described using 2 illustrations. The last line of each illustration is the first line of the next illustration, you don't need to add this line!

```
createCesiumMap():void{
  // Here we trigger the function:"getMapProviders()" to obtain all the
  // available WMS's.
  this.getMapProviders()

  // Here we assign the Cesium ION Token to the Cesium instance.
  Cesium.Ion.defaultAccessToken = 'Paste Your Cesium token';

  // Here we create a new Cesium Map instance (Viewer). We pass the id ('map')
```

```
// Here we create a new Cesium Map instance (Viewer). We pass the id ('map')
// of the div element in the HTML layout in which the Viewer will be added.
this.map = new Cesium.Viewer('map',
 {
     // Here we assign the mapTileLayer as imageryProvider.
     imageryProvider: this.mapTileLayer,
     // Here we assign the mapTerrainLayer as terrainProvider.
     terrainProvider: this.mapTerrainLayer,
  });

  // Here we make sure that animations are not stopped when a data object
  // is loaded.
  this.map.allowDataSourcesToSuspendAnimation = false;
};
```

Now we need to add the function: "createCesiumMap()" to our ngOninit function to make sure that the Cesium Map instance (Viewer) is created when the application is loaded. So edit the function according to the illustration below:

```
ngOnInit() {
  this.getItems();
};
```

```
ngOnInit() {
  this.getItems();
  this.createCesiumMap();
};
```

Now start the Flask-API, TileStache Server, MemCached Cache and the Cesium Terrain Server By clicking on the Desktop Icons shown in the illustrations below (from left to right):

Start-Flask-API   Start-Tileserver   Start-Memcached   Start-Cesium-Server
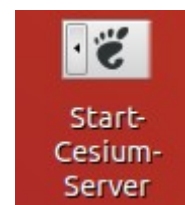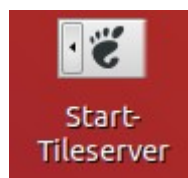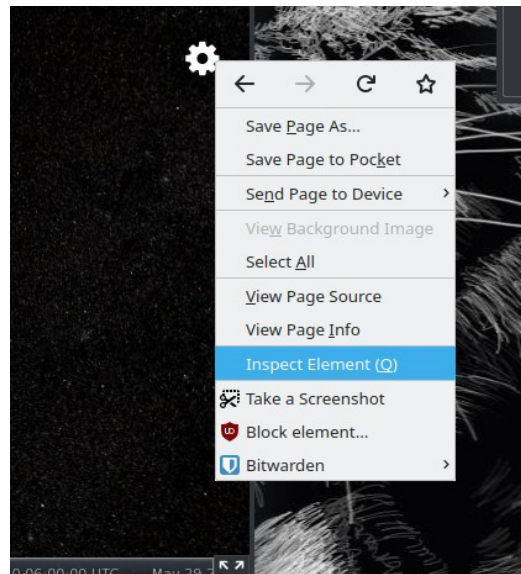
Now when we reload the 3D Map Viewer application you should be greeted with the following screen:


As you may have noticed, you only see a blue sphere which represents the globe. The problem is the map is not showing up yet.

The reason for this can by right clicking on the settings icon and clicking on inspect element as shown in the illustration below:

This will open the Firefox developer options. Now click on the console entry as shown in the illustration below (encircled in red).
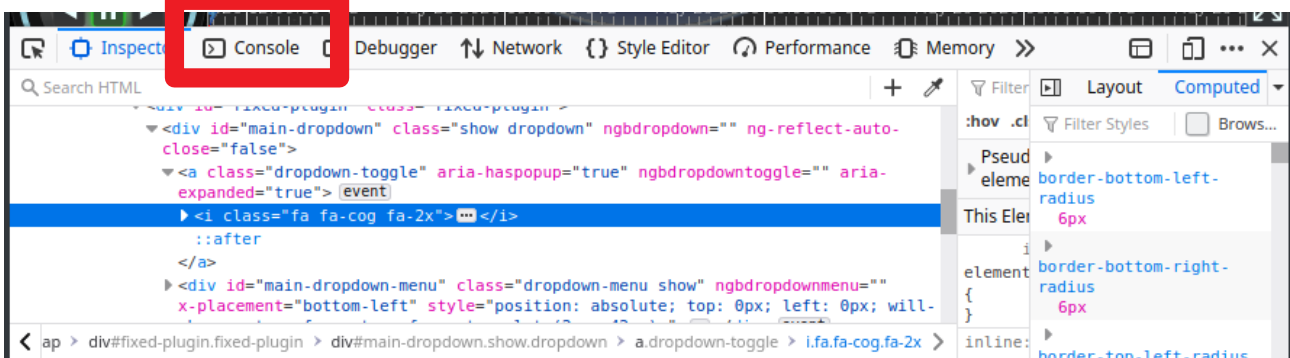
This will open the Firefox developer console. As you can see the problem that occurs is that the Cross-Origin requests are blocked as shown in the illustration below. These requests are made to our TileStache Tileserver from which we obtain the OpenStreetMap tiles.

These error's occur when trying to obtain resources from multiple web servers at the same time. In the example above this error occurred when trying to obtain OpenStreetMap Tiles from the TileStache tileserver (running on "localhost:8081") in the 3D Map Viewer running on localhost:4200.

For more information related to the specifics of CORS you should read the following URL:

https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS

4.1.1 Allow Cross-Origin Resource Sharing (CORS)

To solve this problem we need to install a Firefox extension called: "Allow CORS". This is done by performing the following steps:

1) Navigate to the Addons website of Firefox by entering the following URL in your browser: https://addons.mozilla.org/en-US/firefox/

2) Type: "Allow CORS" in the search bar in the top right of the page and click on the first option that pops up, as shown in the illustration below:
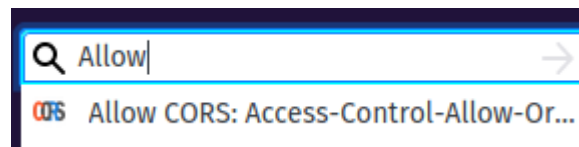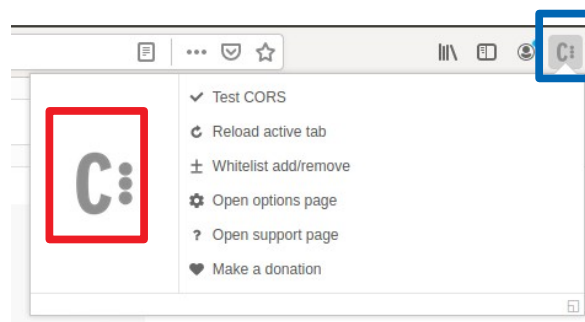

3) On the next screen select: "Add to Firefox" as shown in the illustration below:

4) Then select: "Add" in the popup window as shown in the illustration below:
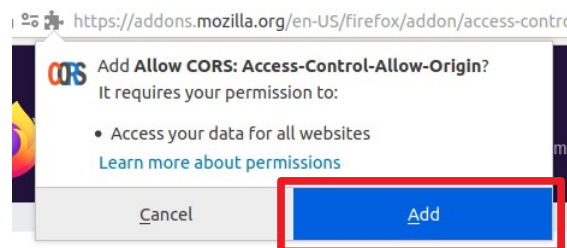
5) After the extension is add a new icon will pop up in the top right of your Firefox window (Blue), click it and Click on the allow CORS icon as shown in the illustration below (Red) :

That's it! Now when you reload the application again you should be greeted with the screen shown in the illustration below:
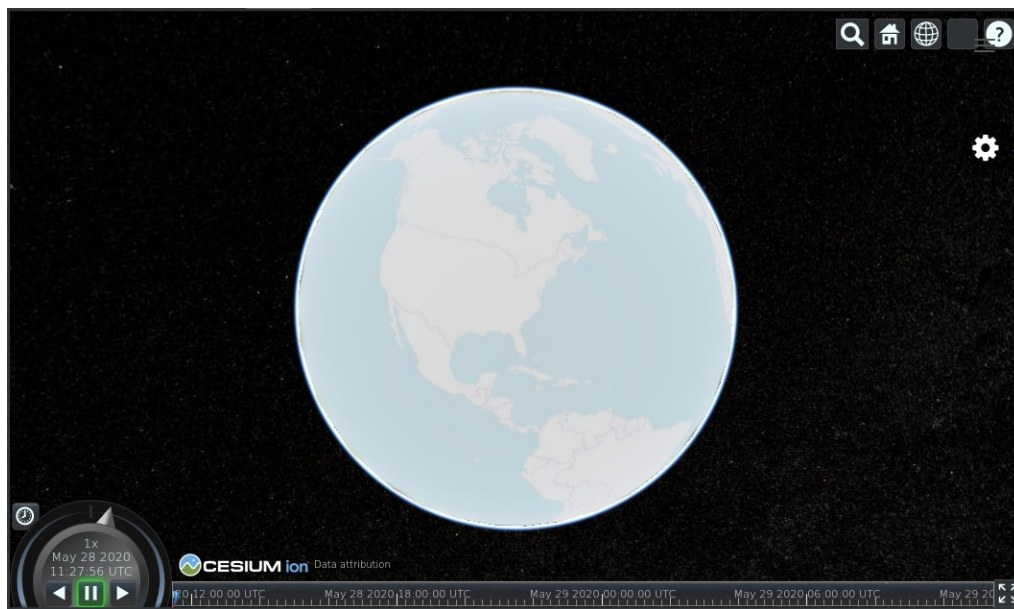

As you may have noticed, the controls at the top right of the screen (encircled in red in the illustration above) can not be clicked. This is because our Navbar overlaps with the buttons. To fix this we need to edit a line of code in our map.component.html file. So let's open the file and edit the div element with the id:"map" according to the illustrations below:
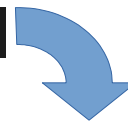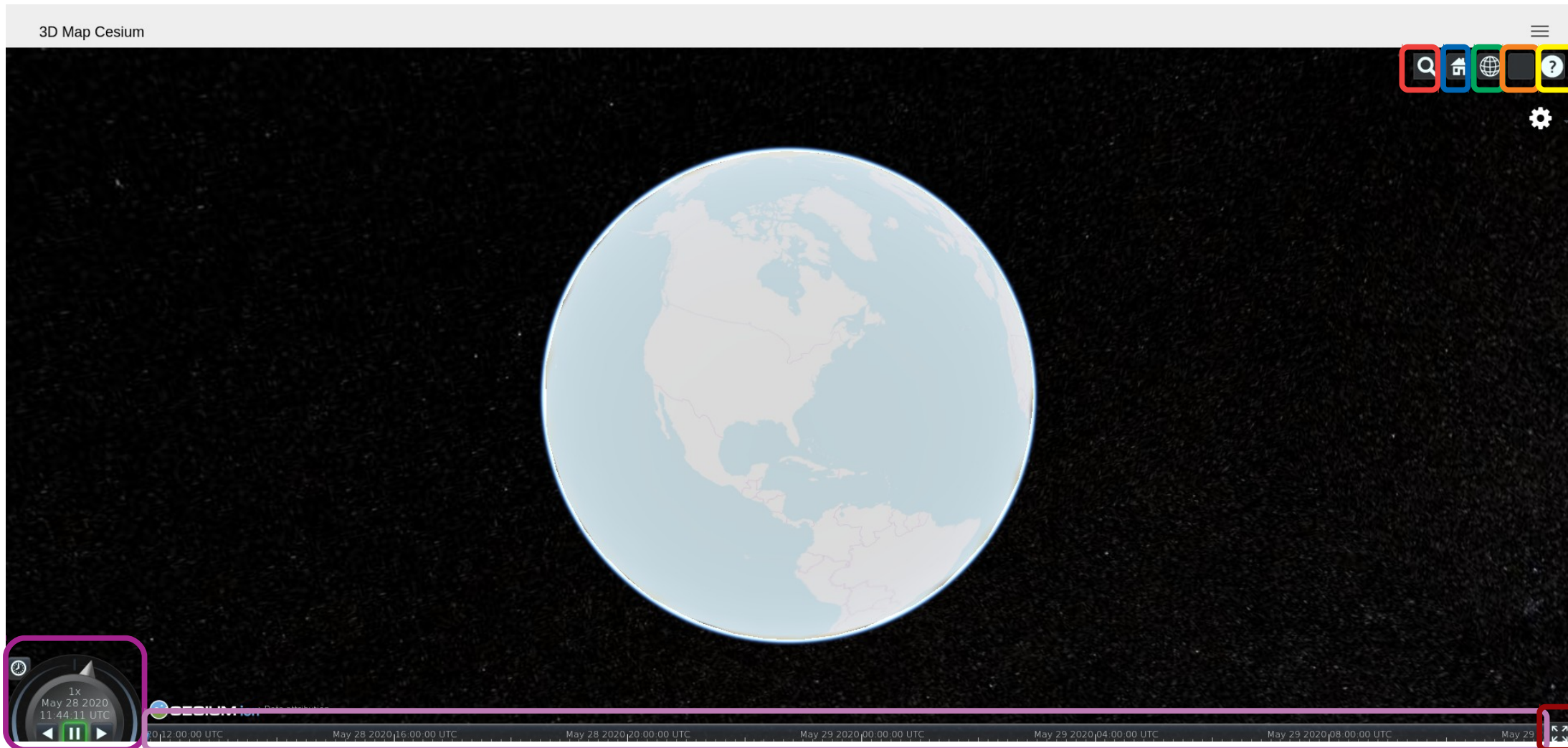

Now if your refresh the page you will be greeted with the following screen in which the buttons are accessible:
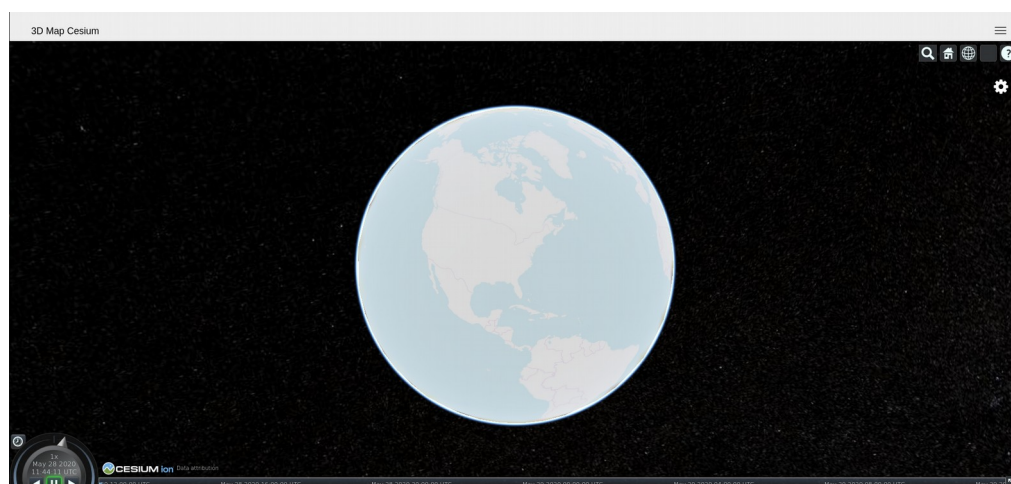
In the next section an explanation is given regarding the default buttons in the Cesium Map Viewer.

4.1.2 Explaining the Cesium Viewer

As you may have noticed the Cesium Viewer contains a lot of default options. In this section a description is given regarding the default Cesium Viewer options using the illustration below. The illustration below contains colors which each represent a different functionality which will be explained during this section.

ddLayerGrou

setLayerGroup