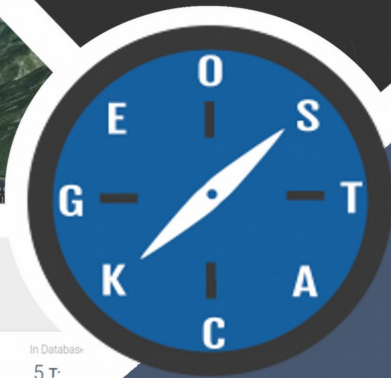
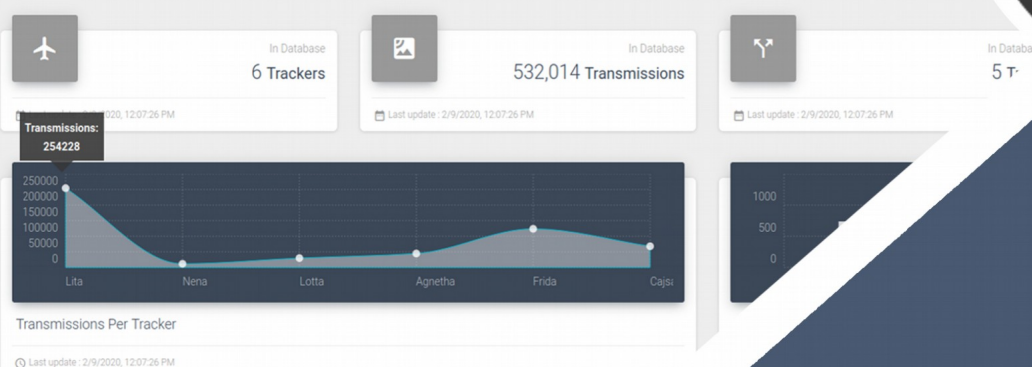


Programming Manual

CREATING AN 3 DIMENSIONAL MAP VIEWER



GPS Dashboard



Categories: **TRACKERS** TRAILS

MongoID	Local Identifier	Crane name	Study name
5e3ec20e146786f4f6917b85	9407	Agnetha	GPS
5e3ec2c5146786f4f6940d1a	9472	Cajsa	
5e3ec23c146786f4f692297c	9381	Frida	

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

Purpose of this document.....2

1.Introduction.....4

 1.1 Getting ready.....4

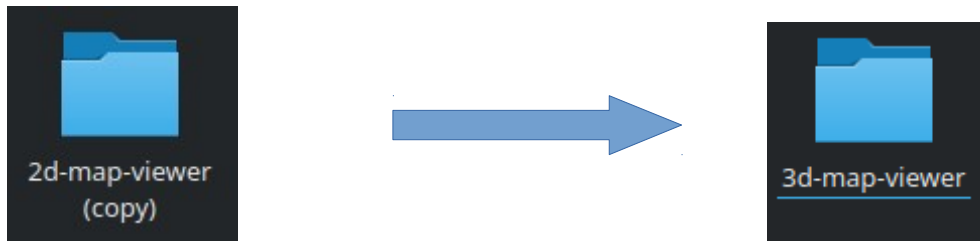
 1.2 Cleaning up.....5

 1.2 Adding the geospatial framework Cesium.....5

1.Introduction

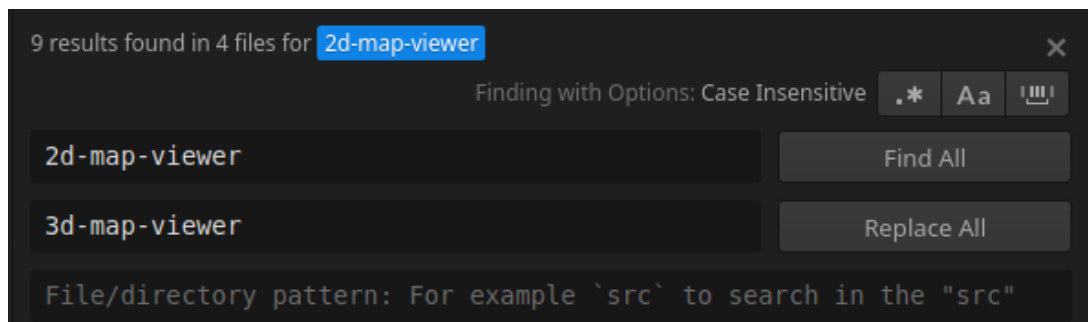
1.1 Getting ready

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.

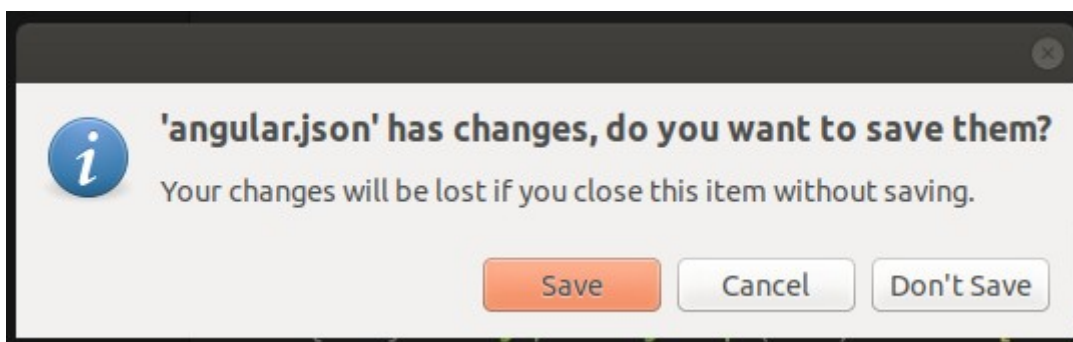


We also need to edit the project name: "2d-map-viewer" to "3d-map-viewer". If you are using the code-editor Atom, this is done by performing the following steps:

- 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.

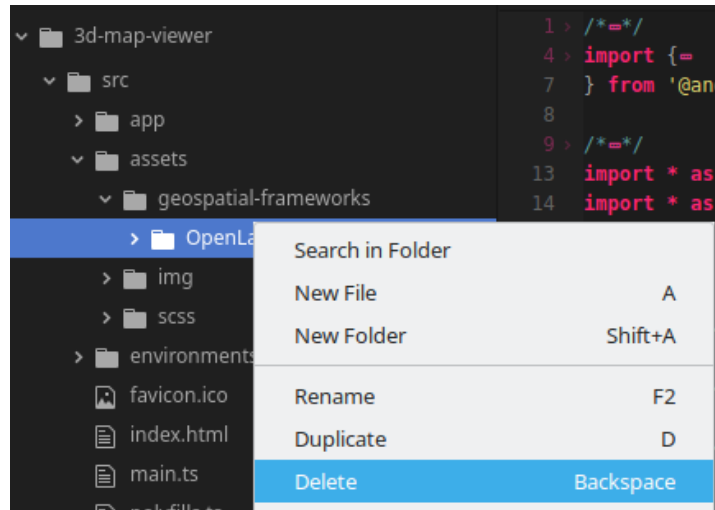


- 4) In the file: index.html located in the folder 3d-map-viewer/src, replace the title from 2D Map Viewer to 3D Map Viewer.
- 5) In the file: sidebar.component.html located in the folder src/app/components/sidebar/, change the text: "3D Map Viewer" to "3D Map Viewer".

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 be 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/>

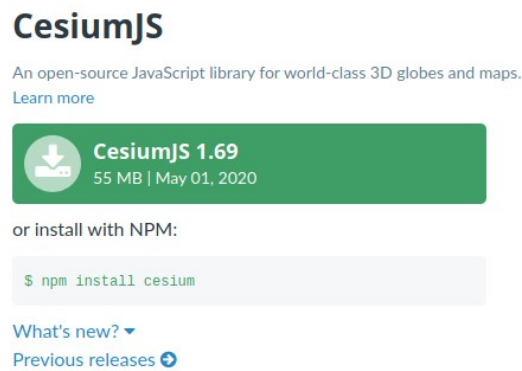
2) Downloading the Cesium source code:

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.

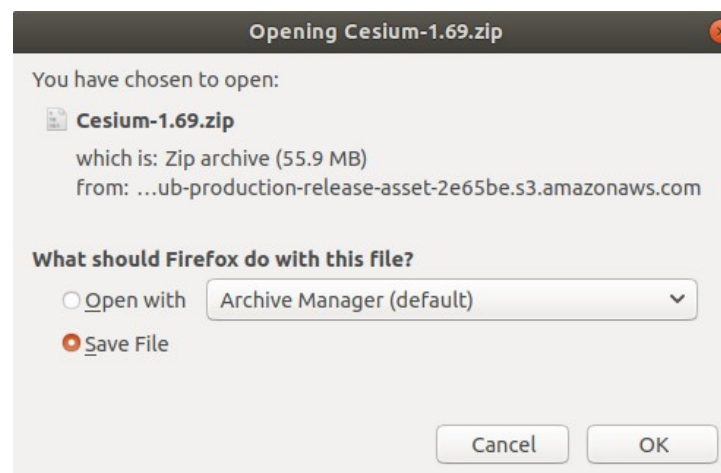
First we want to download the Cesium source code from the Cesium website which is located on the following URL:

<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:

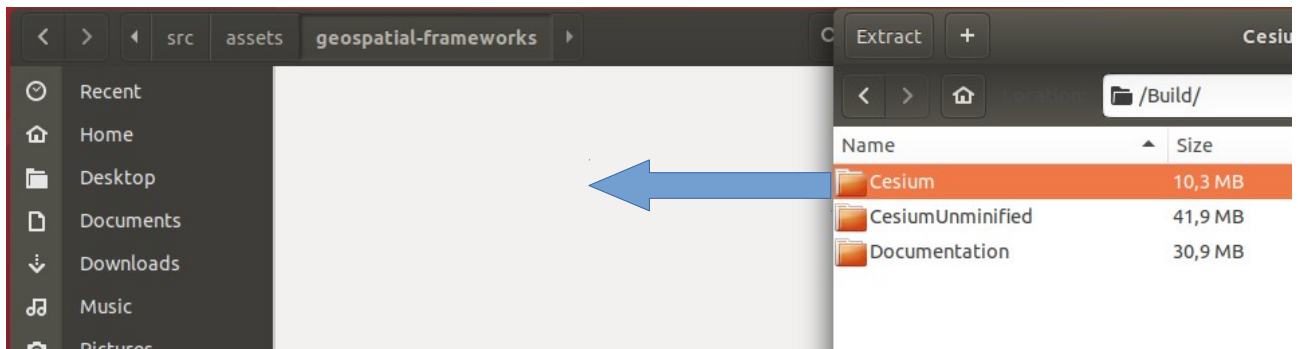


Open the ZIP file. This ZIP contains all the files related to the JavaScript framework Cesium. Since we only need the Cesium source code we should click on the build folder as shown in the illustration below:



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 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.

When we run the 3D Map Viewer application we will encounter a lot of errors as shown in the illustration below:

1.2 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 {=};
43
44 > /*=*/
48 > @Component({=})
49 > export class MapComponent implements OnInit {=};
50
```

1.2.1 Cleaning the unused imports

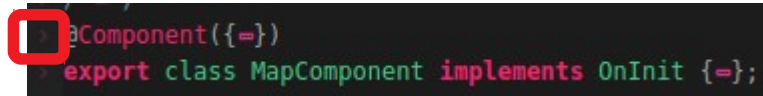
We want to start of by removing the module imports which we don't need anymore. The following module imports have to be removed:

- ➔ 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 metadata. So open the Component metadata by clicking on the arrow (encircled in Red) next to the line @Component({}) as shown in the illustration below:



```
@Component({=})  
export class MapComponent implements OnInit {=};
```

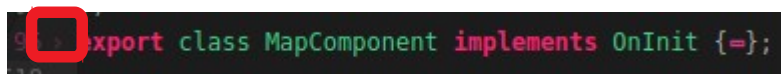
Now remove the PortService from the providers list as shown in the illustration below:

```
@Component({  
  selector: 'app-map',  
  templateUrl: './map.component.html',  
  providers: [MapService, CraneService, PortService]  
})
```



```
@Component({  
  selector: 'app-map',  
  templateUrl: './map.component.html',  
  providers: [MapService, CraneService]  
})
```

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:



```
export class MapComponent implements OnInit {=};
```

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

```
/*= */  
constructor(private _MapService: MapService,  
  private _CraneService: CraneService,  
  private _PortService: PortService) {}
```



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

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.

1.2.2 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;
```

1.2.3 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 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 layer styling 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 {=};

/*= */
getInitialItemData(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.

1.2.4 Editing 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:



The diagram shows two code snippets separated by a blue arrow pointing from left to right. The left snippet is a TypeScript declaration: `declare const ol: any;`. The right snippet is the updated declaration: `declare const Cesium: any;`.

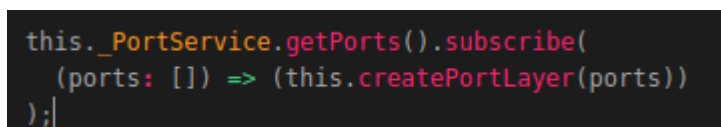
Now we can use the syntax: "Cesium.{a build in function}" to use the functionalities of the JavaScript framework Cesium.

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:



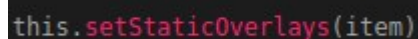
The diagram shows two code snippets for the ngOnInit() function. The left snippet contains: `ngOnInit() {
 this.createOpenLayersMap();
 this.getItems();
};`. The right snippet, after editing, contains: `ngOnInit() {
 this.getItems();
};`. A blue arrow points from the left to the right.

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:



The image shows a code snippet with a line of code that is being removed, indicated by a vertical line through it. The code is: `this._PortService.getPorts().subscribe(
 (ports: []) => (this.createPortLayer(ports))
);`

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 image shows a code snippet with a line of code that is being removed, indicated by a vertical line through it. The code is: `this.setStaticOverlays(item)`

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.