Karta_2.0.js

Armend Azemi

September 2022

Contents

1 Introduction

The purpose of this document is to outline the code in the JavaScript file Karta_2.0.js, which is a file that implements GoogleMaps on the page. This file contains the necessary code to enable simple functionality, such as placing markers on the map, making HTML elements responsive to events that interact with the map, and more. Also, the line numbers on the code preview in this document do not match the line numbers in the actual file, all snippets start on line 1, regardless of their line number in the actual file.

2 Global Variables

At the top of the file we have some global variables, in this section we will cover their meaning and usage.

```
1  //---------------------
2  // GLOBAL VARIABLES
3  //---------------------
4
5  // Google related variabels
6  let focusZoom = 17;
7  let cityZoom = 11;
8  let overviewZoom = 6;
9  var mainMap;
10
11 // DOM variables
12 let resellersParentDiv = document.querySelector('.resellers');
13 let resellerDivs = document.querySelectorAll('.reseller');
14 const searchBar = document.querySelector('#search-city');
15 const autoCompletionList = document.querySelector('.city-list-
       autocomplete');
16
17 // Global storage of all the resellers.
18 let resellersGlobal = [];
19 fetchResellers().then((data) => {resellersGlobal= data})
```

The first three variables in the code-preview above are related to the zoom amount on the map.

- `focusZoom` the zoom when user clicks on a specific reseller, either map marker or HTML element
- `cityZoom` is used for when user searches for a specific city, in which there are resellers available. Gives a city-overview perspective.
- `overviewZoom` this is the broadest overview zoom, and is used on different occasions:
  1. When the map first loads.
  2. If a user is zoomed in a specific marker on the map, and then clicks the marker/DOM element again.
  3. When transitioning between markers and cities.
- `mainMap` Is a global variable holding the Map object. It's initialized in the `initMap()` function.

The DOM element variables are pretty self-explanatory, so we won't dwell any deeper into their meaning. One thing to point out is that the structure of the DOM has to be the same.

- `resellers` (Container of all the reseller divs)

– `reseller` (Individual reseller div, containing reseller information

The final global variable is `resellersGlobal` which is an `object` type. It contains all the data from the resellers. The data is retrieved from the WM3 list on the page, in this case the list is called 'Återförsäljare'. We will talk more about this once we get to the function `fetchResellers()`. This variable should be used with caution.

3 initMap()

This section will cover the functionality of the function `initMap()`. It's important to notice that this function name is has to be indentical to the function name declared in the imported script tag from the Google Maps API, as shown below:

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

Notice the key `callback=` and the value being `initMap`

```
async function initMap() {
    console.log('InitMap Called....');
    // Get the coordinates from the resellers, fetched from  'lists
    .aterforsaljre.lat_lng'.
    let resellerCoordinates = await fetchResellersCoordinates();


    // Set the map options, the center of the map and the zoom on
    load.
    let mapOptions = {
        center: new google.maps.LatLng('57.78594750386966', '
    14.162155747193367'),
        zoom: overviewZoom
    }
    // Create a map and initilize it in the div '#map'
    let map = new google.maps.Map(document.getElementById('map'),
    mapOptions);
    mainMap = map

    // Loop over all the coordinates and create markers for them on
     the map.
    resellerCoordinates.forEach((item)=> {

        // The coordinate are in 'xxxxxxxx, xxxxxx' format. Split
    and get Longitude, and Langtiude by themselves.
        let longAndLang = item.split(',');
        let long = longAndLang[0];
        let lang = longAndLang[1];

        // Create markers
        let markerOptions = {
        position: new google.maps.LatLng(long, lang),
        map: map
        }
        let marker = new google.maps.Marker(markerOptions);


    // Add EventListner to the marker. On click, center the map on
    the marker on zoom in.
    marker.addListener('click', () => {
        map.setCenter(marker.getPosition());
        map.panTo(marker.getPosition());

```

```
37          // If already zoomed in, zoom out.
38          if (map.getZoom() == focusZoom - 1) {
39              smoothZoomOut(map, overviewZoom, map.getZoom());
40          }else {
41              smoothZoomIn(map, focusZoom, map.getZoom());
42          }
43          });
44
45      });
46
47  // Enable all the 'Reseller' divs click functionality.
48      makeResellerDivsClickableOnMap(map)
49  }
```

This function is responsible for initializing the map and populating it with the pin markers. We will skip the first 7 lines of the code preview above, since we will discuss the function `fetchResellerCoorinates()` in a later section.

On line 8 we declare `mapOptions` that has two values, `center` and `zoom` that are used by the Google Map, the centering of the map, and the zoom in which the map should be displayed.

On line 13 we create the `map` object and populate the DOM element with the id 'map', this is a must, and we also call the constructor with the `mapOptions` defined previously.

On line 17 we loop over all the coordinates of the resellers. This data is fetched from the page list with the function `fetchResellersCoordinates()`, which returns an string array. We then extract the Longitude and Latitude coordinates to construct `google.maps.Markers`. The `google.maps.Marker` takes two necessary values, the position of the marker and the map object.

The rest of the code is well documented, and we will discuss the functions in sections below.

4 smoothZoomIn() and smoothZoomOut()

Recusive functions that control the smoothness of the zooming functionality.

```
1  function smoothZoomIn (map, max, current) {
2    // If max zoom is reached, return.
3      if (current >= max) {
4          return;
5      }
6      else {
7        // Increment the zoom until max is reached by recalling self.
        After each zoom, sleep 80ms.
8          let listner = google.maps.event.addListener(map, '
       zoom_changed', (event) =>{
9              google.maps.event.removeListener(listner);
10             smoothZoomIn(map, max, current + 1);
11         });
12         setTimeout(() => {
13           map.setZoom(current);
14
15         }, 80);
16     }
17 }
```

Both functions take 3 arguments:

- `map`: The map object.
- `max/min`: The desired zoom in/out value.
- `current`: The current zoom value.

The smoothness effect is caused by the `setTimeout()` function. We wait 80ms after each incremental/decrement step of zoom. So, if the current zoom in 6 and we want to zoom in to 15 (higher values means more zoom) we increment the `current` variable by 1, recursively call `smoothZoomIn()` with the incremented `current` value, and then apply the zoom after 80ms.

The `smoothZoomOut()` function works identically except that instead of incrementing the `current` value, we decrement it.

5 fetchResellersCoordinates()

```
async function fetchResellersCoordinates (){
  const resellers = await fetchResellers();

  let resellersCoordinates = [];

  for (let index in resellers) {
    resellersCoordinates.push(resellers[index].lat_lng);
  }
  return resellersCoordinates;
}
```

This function is just responsible for extracting the Longitude and Latitude values from the reseller information that is fetched from the page list.

6 fetchResellers()

```
1  async function fetchResellers (){
2  /*
3  * --- NOTES ---
4  *  The current specified list is 'aterforsaljare'
5  */
6
7    // If previously fetched, return the previously saved data, use
       resellersGlobal != null
8    let response = await fetch('/api/v1/lists/aterforsaljare');
9    let json = await response.json();
10   let rows = json.rows;
11
12   let resellers = [];
13
14   for (let key in rows) {
15     resellers.push(rows[key].values);
16   }
17
18   return resellers;
19 }
```

We fetch the data from the 'aterforsaljare' list via the lists API and extract the 'rows' data after converting the response to a json object. The `for` loop on row 14 extract all the values from the `row` object, stores them in the `resellers` array and then returns the `resellers` array.
This array will contain all the information that is in the rows, as key-value pairs. So, each element in the `resellers` array is an object with the data of each row in the list. The example below shows the structure of the `resellers` array.

```
1  [
2  {    namn: HL Design,
3       stad: 'Vaxjo,
4       lat_lng: 59.390768, 17.827633,
5       telefon: 0241-20040,
6       website: www.hldesign.se,
7       address: Kungsgatan 3
8       }
9  ]
```

7 makeResellerDivsClickableOnMap()

This function is only called once in `initMap()`, which we discussed in section 3.

```
1  async function makeResellerDivsClickableOnMap(map){
2    let resellers = await fetchResellers();
3    let previousLocation = null;
4    let currentLocation = null;
5
6    [...resellerDivs].forEach((resellerDiv)=> {
7      resellerDiv.addEventListener('click', (event)=> {
8        // Get reseller information
9        const resellerName = resellerDiv.querySelector('#reseller-
       name').querySelector('b').innerHTML;
10       const resellerAddress = resellerDiv.querySelector('#reseller-
       address').innerHTML;
11       let resellerCoordinates = null;
12
13       // Map the div that was clicked with the 'Resellers' in the
       list and get the correct coordinates.
14       // Map on the Reseller name and address
15       for (let index in resellers) {
16         if(resellers[index].namn.toLowerCase() == resellerName.
       toLowerCase() &&          resellers[index].adress.toLowerCase()
        == resellerAddress.toLowerCase()){
17           resellerCoordinates = resellers[index].lat_lng;
18         }
19       }
20
21      // The coordinate are in 'xxxxxxxx, xxxxxx' format. Split and
        get Longitude, and Langtiude by themselves.
22       let longAndLang = resellerCoordinates.split(',');
23       let long = longAndLang[0];
24       let lang = longAndLang[1];
25
26       currentLocation = new google.maps.LatLng(long, lang);
27       // If already zoomed in, zoom out to the 'overviewZoom' ,
       else, zoom in to 'focusZoom'
28       if (map.getZoom() == focusZoom - 1) {
29           if (previousLocation.equals(currentLocation) ) {
30             map.setCenter(currentLocation);
31             map.panTo(currentLocation);
32             smoothZoomOut(map, overviewZoom, map.getZoom());
33           }else {
34             smoothZoomOut(map, overviewZoom, map.getZoom());
35             // After zooming out, wait, set center, pan over to
       location and zoom in.
36             setTimeout(()=> {
37                 map.setCenter(currentLocation);
38                 map.panTo(currentLocation);
39                 smoothZoomIn(map, focusZoom, map.getZoom());
40             }, 2000);
41
42       }
43
44       }else {
```

```
45              map.setCenter(currentLocation);
46              map.panTo(currentLocation);
47              smoothZoomIn(map, focusZoom, map.getZoom());
48          }
49          previousLocation = currentLocation
50      });
51  });
52  }
```

This function is responsible for making the DOM elements displaying the reseller information clickable and interactive with the map, see the figure below.

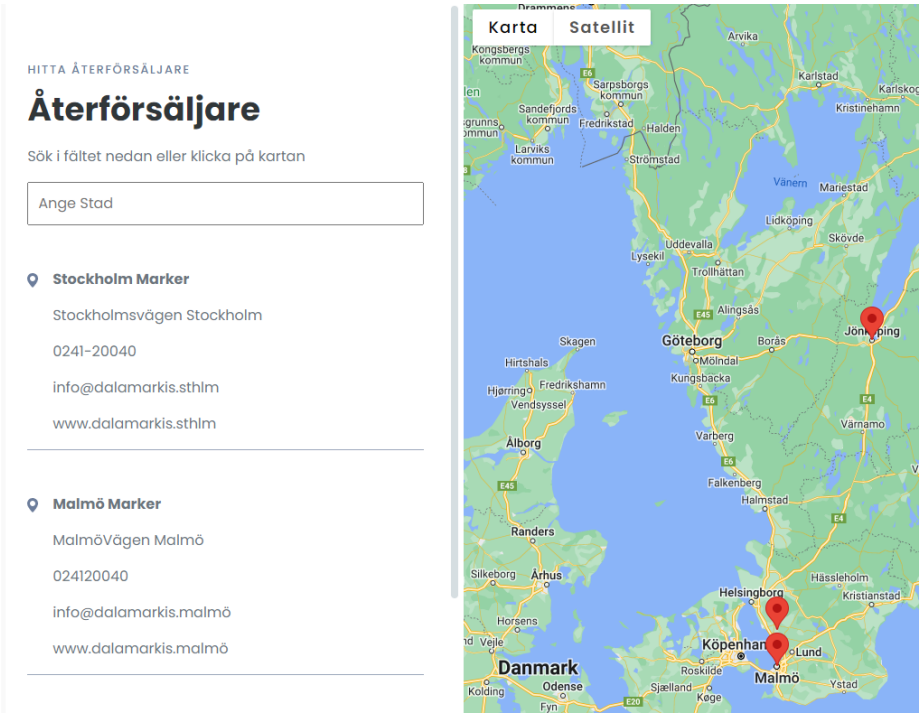

Figure 1: The effected DOM elements are seen on the left-hand side under the input field 'Ange Stad'

To start of, one lines 2-4 we have three variables, `resellers`, `previousLocation`, `currentlocation`. The `resellers` variable just holds the reseller information that is fetched from the list. The last two are just variables to hold the locations, these variables are later used to check if the user interaction on the clickable DOM elements requires the map to change position or not.

We then start adding `eventListeners` to all the reseller DOM elements on line 6. So, how do we map the DOM element to a location on the map? Well, it's pretty straight-forward. Since we don't display the coordinates of the reseller in the DOM element, we can't just extract that information directly from the DOM, we have to get that information from the list.

To do that, we extract the name and the address of the reseller on the clicked DOM element, as can be seen in line 9 and 10. We then use this information to map against the reseller information fetched from the list, we do this on line 15. We loop over all the resellers and check if their name and address match that of the DOM element that was clicked, if that's the case, we extract the coordinates and store them in `resellerCoordinates`

On line 22-24 we just split the coordinates, which is a string with both the longitude and latitude, to get the individual coordinates.

On line 28 we check if the map is already zoomed in on or zoomed out on the map. If we are zoomed in we need to check if the currently selected location is the same as the previous, this means that the user clicked on the same DOM element again. If that is the case, we simply zoom out while keeping the location in the center. But if the user clicks on a different reseller, thus changing the location, then we need to animate the transition. The current transition from a fully zoomed in view on a location to a different location that is also fully zoomed in is the following:

1. Zoom out using `smoothZoomOut()`
2. Wait 2 seconds.
3. Pan over to the next location and center the location on the view.
4. And finally zoom in on the location using `smoothZoomIn()`

And on line 49 we just save the `currentLocation` into `previousLocation`, this will ensure that if the user selects a different reseller using the DOM elements, the comparison will be correct, i.e they will differ.

8 createResellerHTMLElementsWithCity()

This function is primarily used along with the auto completion on the city names. When the user inputs text in the input field searching for a city and selecting one of the suggested cities, this function will remove all resellers from the list that are not in that city.

```
1  function createResellerHTMLElementsWithCity(cityFilter) {
2    [...resellerDivs].forEach((resellerDiv)=> {
3      let resellerCity = resellerDiv.querySelector('#reseller-city');
4
5      // If we have no filter active on the input field, reset the '
       Resellers' list by removing all children
6      // and adding them back in the original order.
7      if (cityFilter == "" || cityFilter == null) {
8          while(resellersParentDiv.firstChild){
9              resellersParentDiv.removeChild(resellersParentDiv.
       firstChild);
10         }
11
12         // Add all the Reseller divs back to the container (
       Resellers)
13         [...resellerDivs].forEach((resellerDiv)=> {
14             resellersParentDiv.appendChild(resellerDiv)
15         });
16     }
17     else{
18         // If a 'Reseller' is not matching the current city filter,
        remove it from the DOM
19       if (!resellerCity.textContent.includes(cityFilter)){
20         resellerDiv.remove();
21       }
22     }
23
24   });
25 }
```

We start of by iterating over all the reseller div elements in the DOM, and extracting the city name of each reseller.

If the `cityFilter` argument is empty then we populate the list with all the resellers, since no filter is currently active. To do this, we firstly remove all the elements, so we don't get duplicates once we insert back all the reseller divs, and then just re-insert all the elements back into the container div `resellersParentDiv`

If the city filter is not empty, then we have some input to filter with. We check if the reseller city is matching that of the `cityFilter`, if it does not match, then we simply remove it from the DOM.

9 autoCompleteCities()

This function is responsible for giving suggestions once the user start searching for a city. The suggestions given are only on cities in which there are resellers in.

```
async function autoCompleteCities() {
  let inputForm = this;
  let cities = [];
  createResellerHTMLElementsWithCity()

  for (let index in resellersGlobal) {
    if (!cities.includes(resellersGlobal[index].stad)){
      cities.push(resellersGlobal[index].stad);
    }
  }

  let sortedCities = cities.sort();
  // Remove previous list items, clear the list.
  removeListElements();

  for (let city in sortedCities) {
    // Check if any city starts with the input search-text
    if (sortedCities[city].toLowerCase().startsWith(inputForm.value
    .toLowerCase()) && inputForm.value !== ""){

      // Create the list elements for the auto completion
      let autoListItem = document.createElement('li');
      autoListItem.classList.add("auto-list-item");
      autoListItem.style.cursor = "pointer";

      // Make the matching letters of the avaiable cities and the
    search term bold.
      let suggestedCity = "<b>" + sortedCities[city].substr(0,
    inputForm.value.length) + "</b>";
      suggestedCity += sortedCities[city].substr(inputForm.value.
    length);

      // Append list elements to the autocompletion list.
      autoListItem.innerHTML = suggestedCity

      autoCompletionList.appendChild(autoListItem);
      let suggestedCityString = sortedCities[city]

      // Clicking on suggested city will remove all suggestions and
      // populate the input form with the city name and call '
    createResellerHTMLElementsWithCity'
      // with the clicked city.
      autoListItem.addEventListener('click', ()=> {
        removeListElements();
        inputForm.value = suggestedCityString
        createResellerHTMLElementsWithCity(suggestedCityString);
        zoomInOnCity(suggestedCityString);
      });
    }
  }
}
```

On line 2 we declare `inputForm = this`. This function is ONLY called from the event listener conncted to the input form, thus, `this` will always reference the input form DOM element.

Line 2 is just an array to store all the cities in which there are resellers located in.

On line 3 we call `createResellerHTMLElementsWithCity()` to ensure that the reseller list is never empty, we start by re-populating the list once the user start inputing text to the input form. On line 6 we simply get all the cities from the resellers, and if the city has already been added to the array, we ignore it, we only want unique cities. All the cities will be displayed in a alphabetical order, this is achieved on line 12.

On line 16 we start iterating over all the cities to check if the input from the user matches any of the cities in which there are resellers available. If there is a match, that is, the input from the user matches one of the available cities, we start creating the `<li>` elements, we add a class name and set the cursor to be a `pointer`.

On line 26 we make the matching letters of the input and the city it's matching bold, as can be seen in the figure below.

Figure 2: The matching letters of input and the city name are displayed in bold text

The `<li>` is then added to the DOM by appending it to the `<ul>` element `autoCompletionList` and we store the city name that's matching the user input in the variable `suggestedCityString` on line 33, this will be used to filter the reseller list by calling `reateResellerHTMLElementsWithCity()` later on.

On line 38 we add an `eventListener` to the `<li>` element. This event listener will listen for a click on itself. If the user clicks the `<li>`, which is a city name, then we remove all `<li>` elements from the `autoCompletionList` by calling

`removeListElements()`. We then set the input form value to be that of the city that was selected in the dropdown auto completion list, and filter the reseller list, only showing the resellers that are in that city, and finally we zoom in on the city to get a city overview view. The figures below demonstrates this behavior.

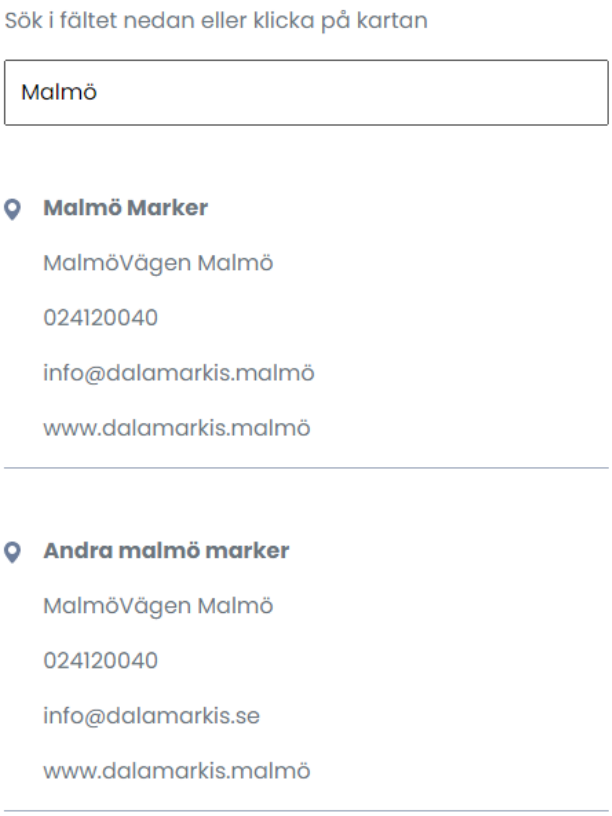

Figure 3: After clicking on the suggestd city, the reseller list is filterd, and the suggested cities list (autoCompleteList) is removed

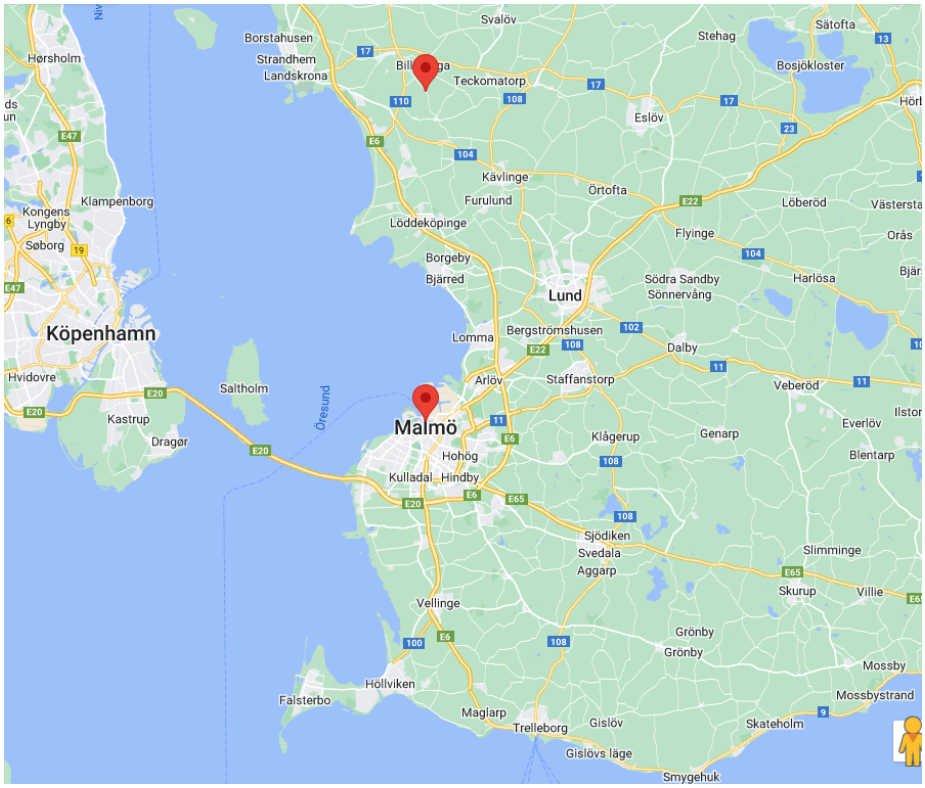

Figure 4: Showing the city overview view after clicking on the suggested city.

10 zoomInOnCity()

Since we don't have the coordinates of the cities that the resellers are located in, we have to get the coordinates from the Google API using the city name.

```
1  async function zoomInOnCity(city) {
2    let cityResponse = await fetch(`https://maps.googleapis.com/maps/
       api/geocode/json?address=${city}+sweden&sensor=true&key=
       YOUR_API_KEY`)
3    .catch(error => console.error('There was an error fetching the
       city coordinates from Google: '  + error));
4
5    let cityResponseJson = await cityResponse.json()
6    .catch(error => console.error('There was an error converting the
       response object from Google for the city:' + city + ' Error: '
       + error));
7    let cityLng = cityResponseJson.results[0].geometry.location.lng
8    let cityLat = cityResponseJson.results[0].geometry.location.lat
9
10   let cityCoordinates = new google.maps.LatLng(cityLat, cityLng);
11
12   // Check the current zoom of the map, if the current zoom is
       greater than the 'cityZoom'
13   // we zoom out and then pan and center on the city and apply the
       cityZoom on the map.
14   if (mainMap.getZoom() > cityZoom){
15     smoothZoomOut(mainMap, overviewZoom, mainMap.getZoom());
16     setTimeout(() => {
17       mainMap.setCenter(cityCoordinates);
18       mainMap.panTo(cityCoordinates);
19       smoothZoomIn(mainMap, cityZoom, mainMap.getZoom())
20     }, 2000);
21   }else {
22     mainMap.setCenter(cityCoordinates);
23     mainMap.panTo(cityCoordinates);
24     smoothZoomIn(mainMap, cityZoom, mainMap.getZoom())
25   }
26 }
```

On line 2 we make a `GET` request to the Google Geocode API. We use the `address` in query string with the city name + the country (in our case it's Sweden) as values On line 5 we convert the response object from the `GET` request and extract the coordinates on line 7 and 8 which are used to create a Google Map LatLng object on line 10.

On line 14 we check if the current zoom of the map is greater then the `cityZoom` value, if that's the case we have to zoom out to the `cityZoom` value, then we wait 2 seconds before centering and panning the map over the the city location. And if the current zoom of the map is less than the city zoom we just center and pan over the location and zoom in.

11 function removeListElements()

Very simple function that doesn't need much detailed explanation. Just remove all the list elements with the class name `.auto-list-item`

```
1 function removeListElements() {
2   let items = document.querySelectorAll('.auto-list-item');
3   items.forEach((item) => {
4     item.remove();
5   })
6 }
```

12 searchBar.addEventListener()

No explanation needed.