Topic	AR NAVIGATION	
Class Description	**Students will learn about location based augmented reality. Students will also learn to use location coordinates to create augmented reality navigation paths from the source to the destination using Mapbox GL JS library.**	
Class	**C180**	
Class time	**45 mins**	
Goal	• Learn about location based augmented reality. • Learn how to render augmented navigation paths using Mapbox GL JS library.	

Resources Required	• Teacher Resources:
	  ○ Visual Studio Code Editor
	  ○ laptop with internet connectivity
	  ○ smartphone
	  ○ earphones with mic
	  ○ notebook and pen
	• Student Resources:
	  ○ Visual Studio Code Editor
	  ○ laptop with internet connectivity
	  ○ smartphone
	  ○ earphones with mic
	  ○ notebook and pen	

Class structure	**Warm-Up** **Teacher-led Activity** **Student-led Activity** **Wrap-Up**	**5 mins** **15 mins** **20 mins** **5 mins**
Attributions	**Title:** Google Maps Screenshots Images. **Author:** Lars Rasmussen, Jens Eilstrup	

	Rasmussen. **Source:** https://www.google.com/maps	
WARM-UP SESSION - 5 mins		
CONTEXT • **Understand location based augmented reality.**		
Teacher Starts Slideshow  **Slide 1 to 4** Refer to speaker notes and follow the instructions on each slide.		
Hey <student's name>. How are you? It's great to see you! Are you excited to learn something new today? **Following are the WARM-UP session deliverables:** • Greet the student. • Revision of previous class activities.		**ESR**: Hi, thanks! Yes I am excited about it! Click on the slide show tab and present the slides
WARM-UP QUIZ Click on In-Class Quiz		
Continue WARM-UP Session  **Slide 5 to 10**		
Following are the session deliverables: • Appreciate the student. • Narrate the story by using hand gestures and voice modulation methods to bring in more interest in students.		
Class Steps	**Teacher Action**	**Student Action**
Step 1: Warm-Up (5 mins)	Hi, how are you? Great!	**ESR:** I am good!

	Let me quickly summarize the maps concepts that we have covered till now: ***Note**: Encourage the student to discuss what they remember and help them to be more involved.* • We created a Mapbox account and learned to render maps using Mapbox JS GL library. • We used the Map object to render maps using the user's Mapbox account access token. • We learned to add controls (using Map object methods) to the maps like MapboxDirections and GeolocateControl. • Finally we created two web pages and used query parameters to get the longitude and latitude coordinates of the source and destination from the URL of the main web page on the next page. Well, we did learn lots of new concepts in the few previous classes. In today's class we will learn a new concept: location based augmented reality, then we will merge all concepts	***Note**: The student discusses his/her views with the teacher.*

	together to create augmented reality navigation. Remember what we discussed about markerless augmented reality? Yes. Superb! We will continue using A-Frame to create location based augmented reality. Are you excited? Let's get started then.	**ESR**: Yes. We have used different types of markers to render augmented objects in the scene and we can use our location instead of markers to create markerless augmented scenes. **ESR**: Yes.
Teacher Ends Slideshow 		
TEACHER-LED ACTIVITY - 15 mins		
Teacher Initiates Screen Share		
CHALLENGE • **Render AR Navigation Path.**		

Step 2: Teacher-led Activity (15 mins)	*<The teacher can refer to the code from before the class begins>.* To begin, we will need A-Frame and AR.js libraries. Do you remember how we had initialized the arjs in the scene? Yes. Great!	**ESR**: We can attach the **arjs** component in the <scene>.

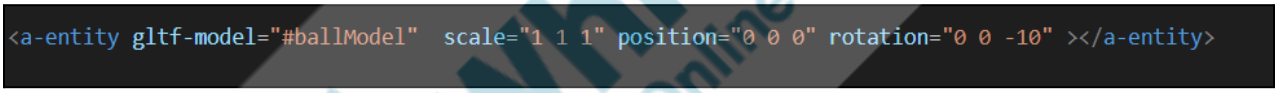
```
<script src="https://aframe.io/releases/1.0.4/aframe.min.js"></script>

<script src="https://raw.githack.com/AR-js-org/AR.js/master/aframe/build/aframe-ar-nft.js"></script>
```

```
<a-scene vr-mode-ui="enabled: false" embedded arjs="sourceType: webcam; debugUIEnabled: false;">


</a-scene>
```

	Let's first understand how we can use the A-Frame AR.js components to create location based AR content. Remember we can render 3D models, images, videos, text or any other assets in AR? Let's add the asset that we can render. We have a 3D model of a ball that we can use.	**ESR:** Yes.

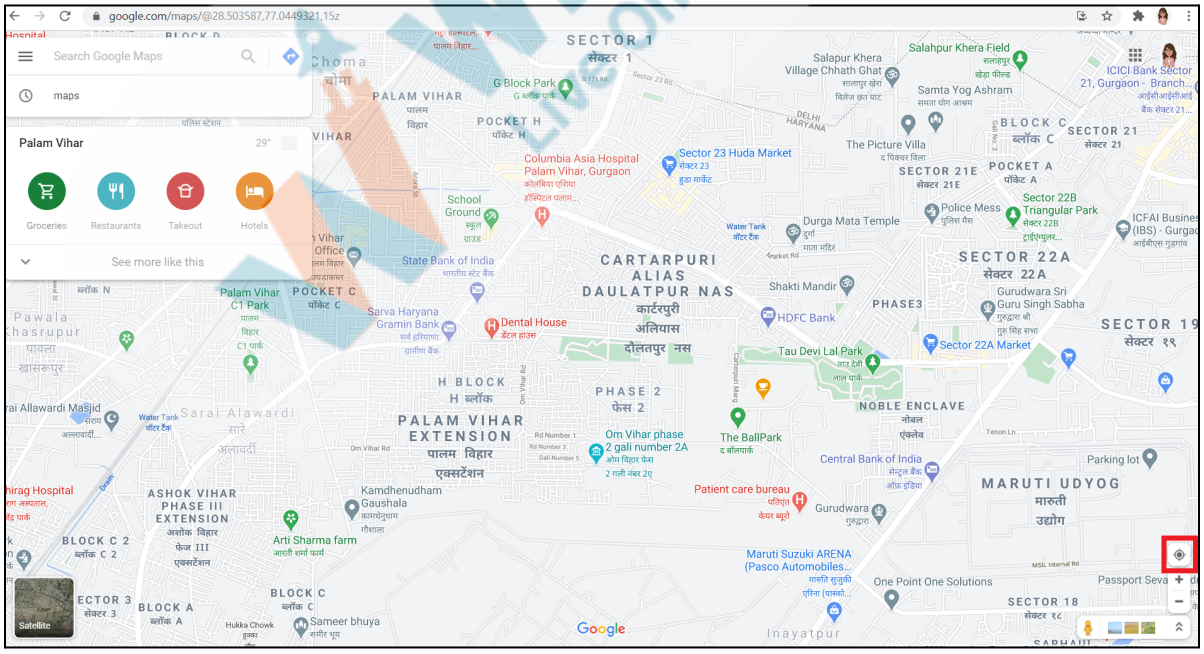
	Can you tell me how shall we include this asset in A-Frame? Yes. Amazing! And then we can use the <a-entity> tag to add the 3D model in the scene. We can set the rotation, position and scale attributes to set the orientation and size of the model.	**ESR:** We can use the <a-assets> tag, then to add 3D models we will use <a-asset-item>.
	 	
	Now we are going to use two components to place the 3D model at a particular location. In our application we have been using Mapbox to get the latitude and longitude values of any particular place. But for now, let's take the values of latitude and longitude from Google Maps. For this:	

```
<a-assets>
  <a-asset-item id="ballModel" src="./ball/scene.gltf">
  </a-asset-item>
</a-assets>
```

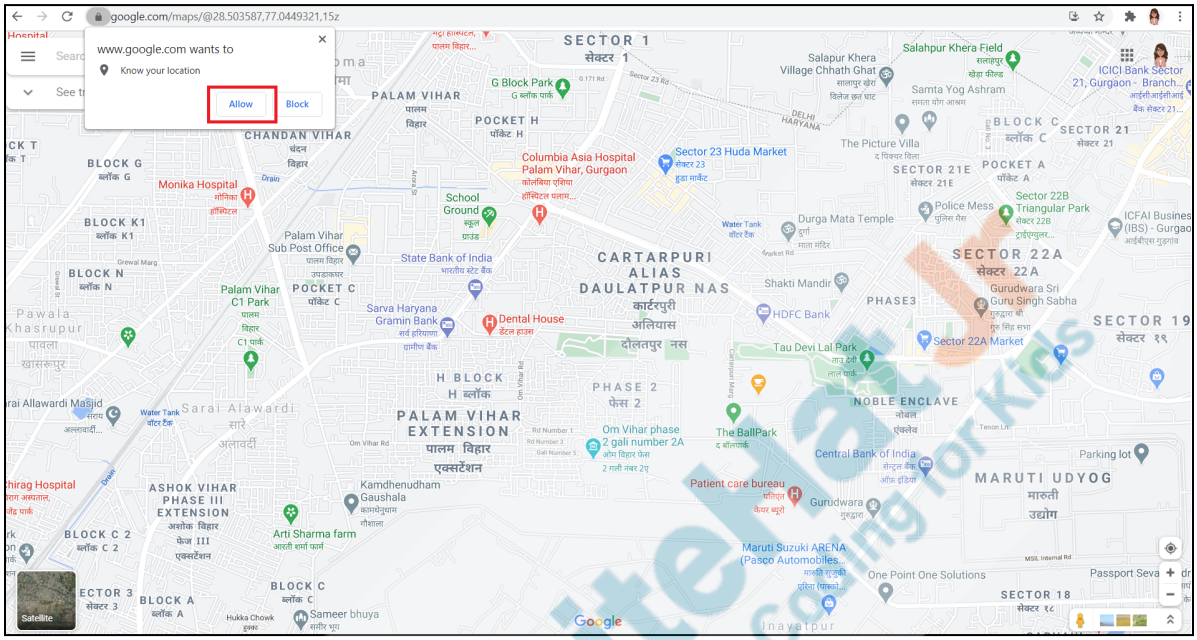
```
<a-entity gltf-model="#ballModel"  scale="1 1 1" position="0 0 0" rotation="0 0 -10" ></a-entity>
```

	• Open Google Maps. • Click on the “Show Your Location” button. • Give permission to “Know your location”. • Click on the “Show Your Location” button again. • Find the “blue” dot pointing to your location. • Right-click on the “blue” dot and click on the first row (having latitude and longitude). It will be copied automatically. • Paste it somewhere for later use. Note: Use values till 7 decimal places at max.	

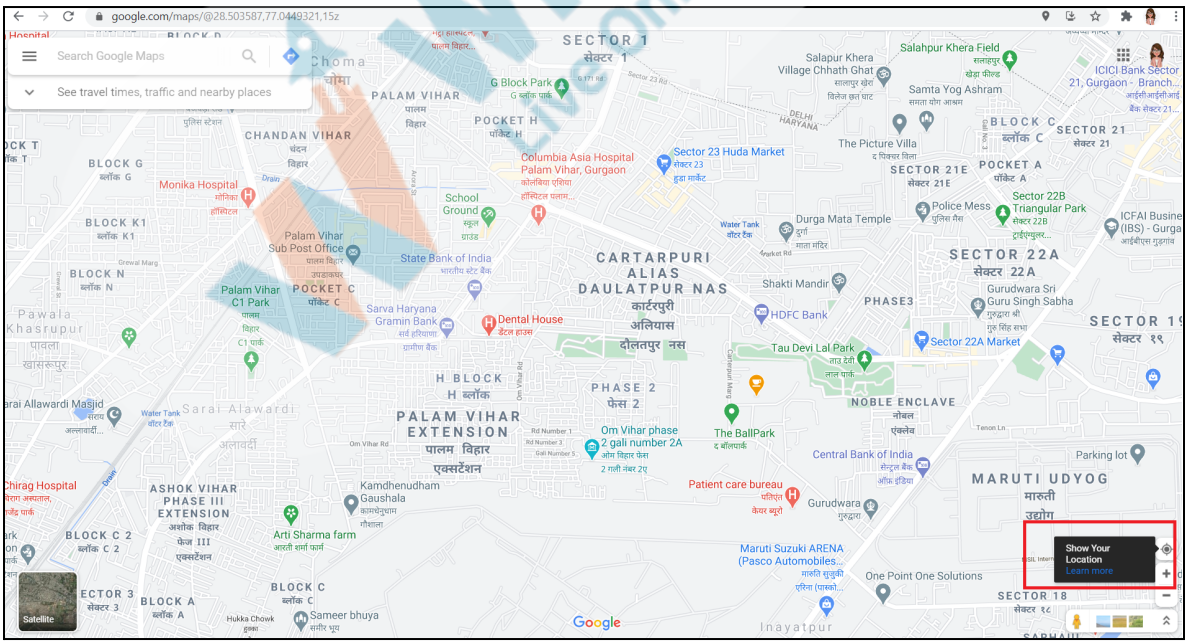
Open Google Maps in your browser.



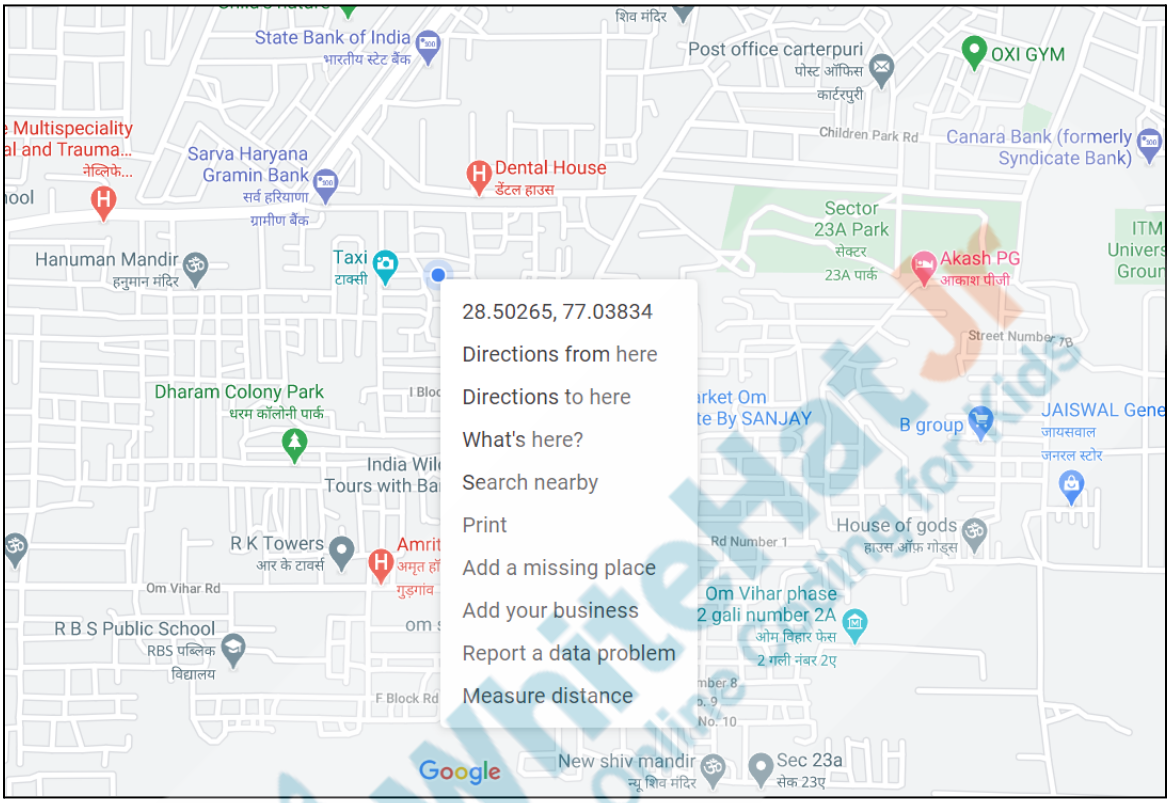
Give permission to “Know your location”.



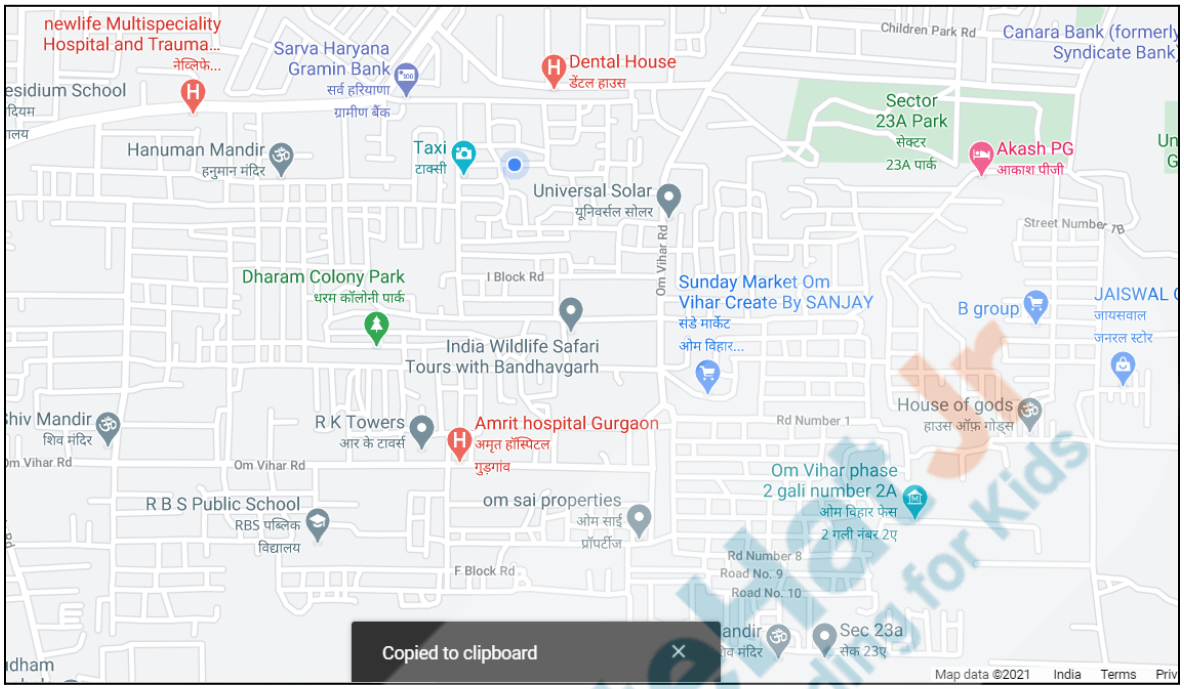
Click on the “Show your location” button.



Find the “blue” dot located at your location and right click on that.



Click on the coordinates and it will be copied automatically.



***Important Note**: The order of the location coordinates in Google Maps is **[latitude, longitude]**. It is different from standard map applications, which have reverse order as [longitude,latitude].*

Now once we have the location coordinates (latitude and longitude), we can place any entity in A-Frame using the **gps-entity-place** component.

We can provide the latitude and longitude as values to this component in the string format:

gps-entity-place="latitude: <your-latitude>; longitude: <your-longitude>"

This component makes the GPS entity trackable.

	Can you tell me what is GPS? GPS is a **Global Positioning System**. It tells us where we are on Earth. It's a satellite navigation system, that means location or navigation data are collected using satellites. 24 satellites are orbiting around Earth that send data to GPS devices. GPS is a powerful technology and it can be used for: • **Location**—Determining a position. • **Navigation**—Getting from one location to another. • **Tracking**—Monitoring objects/personal movement. • **Mapping**—Creating maps of the world. • **Timing**—Making it possible to take precise time measurements. So, when the **gps-entity-place** component is attached in <a-entity> tag, it will assign a position (latitude and longitude) on the Earth to that entity. And when we point the device towards that position we can see the entity at that position. If we are far from that	**ESR**: Varied.

	position (latitude and longitude), then the entity will look small. ***Note**: If we are very far from the position (latitude and longitude), we won't be able to see it.* Let's attach this component to the model entity we created and use the latitude and longitude we copied before.	

```
<a-entity gltf-model="#ballModel"  scale="1 1 1" position="0 0 0" rotation="0 0 -10"
gps-entity-place="latitude: 22.7868542 ; longitude: 88.3643296;"></a-entity>
```

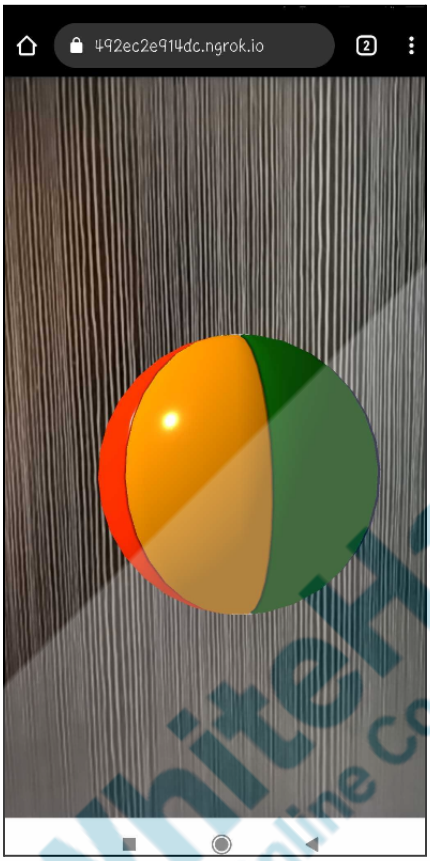
	Now to enable location AR, we need to set <a-camera> as a GPS camera. We can use the **gps-camera** component for this. We can set the attribute **minDistance** and **positionMinAccuracy** of the component: • **minDistance**: When this value is set, then if the distance of the user (having the GPS device) is lower than this value, then the entity at that position/location will not be shown. • **positionMinAccuracy**: Sets the tracking accuracy at that position.	

```
<a-camera gps-camera rotation-reader minDistance="1" positionMinAccuracy="0"> </a-camera>
```

	And to make sure the entity always points to the camera, we will use an A-Frame **look-at** component to the entity. We can give the value of the **look-at** component as a gps-camera so that it always faces the camera. This will make sure the entity is facing the camera. For this we have to use the A-Frame library link: https://unpkg.com/aframe-look-at-component@0.8.0/dist/aframe-look-at-component.min.js	

```
<a-entity gltf-model="#ballModel" look-at="[gps-camera]" scale="1 1 1" position="0 0 0" rotation="0 0 -10"
gps-entity-place="latitude: 22.7868542 ; longitude: 88.3643296;"></a-entity>
```

	Now we can test the output using ngrok.	

<table>
<tr><td></td><td> 
</td><td></td></tr>
<tr><td></td><td>That's really interesting!

We can now render AR content without markers.

Now you will render the arrow images for the navigation from source to destination in augmented reality.

Are you excited?</td><td>ESR: Yes!</td></tr>
<tr><td colspan="3">Teacher Stops Screen Share</td></tr>
<tr><td></td><td>Now it's your turn. Please share your screen with me.</td><td></td></tr>
</table>

Teacher Starts Slideshow  **Slide 11 to 15** Refer to speaker notes and follow the instructions on each slide.		
We have one more class challenge for you. Can you solve it? Let's try. I will guide you through it.		
Teacher Ends Slideshow 		
STUDENT-LED ACTIVITY - 20 mins		
• **Ask the student to press the ESC key to come back to the panel.** • **Guide the student to start screen share.** • **Teacher gets into fullscreen.**		
ACTIVITY • **Render maps using Mapbox APIs.** • **Add direction controls to the map.**		
Step 3: Student-led Activity (20 mins)	*The teacher guides the student to clone the code from Student Activity 1.* ***[Student Activity 1]*** ***Note**: The student will continue to add new functionality after teacher activity.*	
	From the previous class we have our location coordinates that we got from the URL on the next page.	

	Now we can use the mapbox direction API again to get the complete navigation path data. The direction of the navigation path will be represented using arrow images on the path to show the direction to take turns. The possible arrows directions are given in below table: ***Note**: The student can check out images in downloaded code.*	

Direction	Image
Start	
Left	
Right	
Slight Left	
Slight Right	
Straight	

We are going to write a function **render_elements()**, to show the arrow images on the path to show the direction to take turns and call the function in the dollar function.

```
$(document).ready(function () {
    get_coordinates();
    render_elements();
})

function render_elements() {

}
```

Now, we will jQuery the ajax() method to call the Mapbox Directions API using the access token.

This API gives turn-by-turn instructions.

In the ajax() method call, the **url** will be like this:
https://api.mapbox.com/directions/v5/mapbox/driving/${coordinates.source_lon}%2C${coordinates.source_lat}%3B${coordinates.destination_lon}%2C${coordinates.destination_lat}?alternatives=true&geometries=polyline&steps=true&access_token=pk.eyJ1IjoiYXBvb3J2ZWxvdXMiLCJhIjoiY2ttZnlyMDgzMzlwNTJ4a240cmEzcG0xNyJ9.-nSyL0Gy2nifDibXJg4fTA

This URL has **source_lon,source_lat;destination_lon,destination_lat** variables from the URL of the main template we created in the previous class.

***Note 1**: The latitude and longitude are separated by the comma in the URL.*

***Note 2**: The source and destination are separated by a semicolon in the URL.*

***Note 3**: The %2C in the URL represents the comma and %3B represents the semi-colon. This special encoding is done for the URLs to transfer data.*

At the end of the URL, there is an **access token** of the user.

We can set the **type** of the API call as **'get'** to get the directions in the response.

```
$.ajax({
    url: `https://api.mapbox.com/directions/v5/mapbox/driving/${coordinates.source_lon}%2C${coordinates.source_lat}%3B${coordinates.destination_lo
    type: "get",
    success: function(){

    }
})
```

The success callback of an ***AJAX*** request takes an argument ***response*** which contains the response from the API. Let's add it and see what it gives us.

```
function render_elements() {
    $.ajax({
        url: `https://api.mapbox.com/directions/v5/mapbox/driving/${coordinates.source_lon}%2C${
        type: "get",
        success: function (response) {
            console.log(response)
        }
    })
}
```

On running it in the browser and checking the Google Inspect result, we see:

	The JSON response of the API has following structure:
	routes:[{
	...
	legs:[{
	...
	steps:[{
	
	..
	}]
	}]
	}]
	Check out the sample response of the Mapbox Directions API in the document below:	

	[Teacher Activity 3]	


```
{
  "routes": [
    {
      "geometry": "mnn_Ick}pAfBiF`CzA",
      "legs": [
        {
          "summary": "Köpenicker Straße, Engeldamm",
          "weight": 44.4,
          "duration": 26.2,
          "steps": [
            {
              "intersections": [
                {
                  "out": 0,
                  "entry": [ true ],
                  "bearings": [ 125 ],
                  "location": [ 13.426579, 52.508068 ]
                },
                {
                  "out": 1,
                  "in": 2,
                  "entry": [ true, true, false ],
                  "bearings": [ 30, 120, 300 ],
                  "location": [ 13.426688, 52.508022 ]
                }
              ],
              "driving_side": "right",
              "geometry": "mnn_Ick}pAHUlAqDNa@",
              "mode": "driving",
              "maneuver": {
                "bearing_after": 125,
                "bearing_before": 0,
                "location": [ 13.426579, 52.508068 ],
                "modifier": "right",
                "type": "depart",
                "instruction": "Head southeast on Köpenicker Straße (L 1066)"
              },
              "ref": "L 1066",
              "weight": 35.9,
              "duration": 17.7,
              "name": "Köpenicker Straße (L 1066)",
              "distance": 98.1,
```


	***Note**:There is a lot of information provided as part of API call response, but we are going to use a few of the information from the response.* Now in the success function, we are going to get the **steps** and loop	

through all the steps from source to destination.

Guide the student to get data from the response in the success function.

```
success: function (response) {

    let steps = response.routes[0].legs[0].steps

    for (let i = 0; i < steps.length; i++) {
        let image;
        let distance = steps[i].distance
        let instruction = steps[i].maneuver.instruction

        if (instruction.includes("Turn right")) {
            image = "turn_right"
        } else if (instruction.includes("Turn left")) {
            image = "turn_left"
        }


    }
}
```

Now we can take a JSON variable, **images**, with direction keywords and their respective images.

Guide the student to define image variables in JSON format.

```
success: function (response) {
    let images = {
        "turn_right": "ar_right.png",
        "turn_left": "ar_left.png",
        "slight_right": "ar_slight_right.png",
        "slight_left": "ar_slight_left.png",
        "straight": "ar_straight.png"
    }
    let steps = response.routes[0].legs[0].steps
```

	Now we can use the jQuery **append()** method to add the arrow images in the AR scene. We can use the id to select the <a-scene> using the $ function. In this method we can use <a-entity> to set the **gps-entity-place** component like we did before. And <a-image> to set the image. *Guide the student to write the append() method:* • *Select the <a-scene> element using jQuery selector.* • *Use the append() method to set the <a-entity> and <a-image>.*	

```
if (i > 0) {
    $("#scene_container").append(
        `
            <a-entity gps-entity-place="latitude: ${steps[i].maneuver.location[1]}; longitude: ${steps[i].maneuver.location[0]};">
                <a-image
                    name="${instruction}"
                    src="./assets/${images[image]}"
                    look-at="#step_${i - 1}"
                    scale="5 5 5"
                    id="step_${i}"
                    position="0 0 0"
                >
                </a-image>
                <a-entity>
                    <a-text height="50" value="${instruction} (${distance}m)"></a-text>
                </a-entity>
            </a-entity>
        `
    )
} else {
    $("#scene_container").append(
        `
            <a-entity gps-entity-place="latitude: ${steps[i].maneuver.location[1]}; longitude: ${steps[i].maneuver.location[0]};">
                <a-image
                    name="${instruction}"
                    src="./assets/ar_start.png"
                    look-at="#step_${i + 1}"
                    scale="5 5 5"
                    id="step_${i}"
                    position="0 0 0"
                >
                </a-image>
                <a-entity>
                    <a-text height="50" value="${instruction} (${distance}m)"></a-text>
                </a-entity>
            </a-entity>
        `
    )
}
```

*Test the output using **ngrok** as we did in the previous AR applications during the class.*

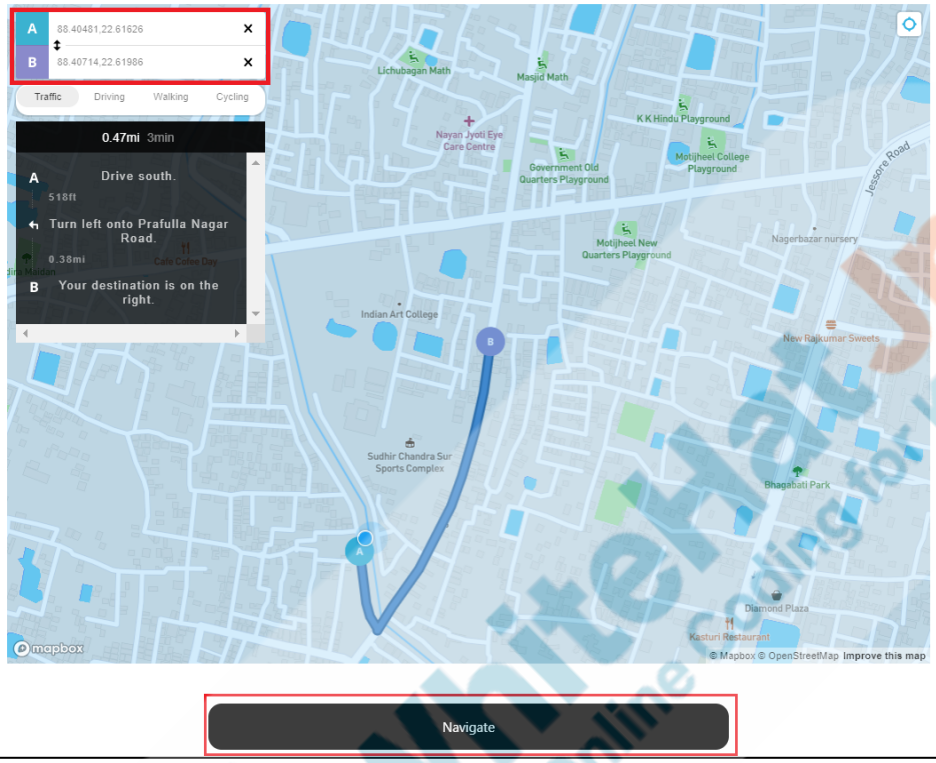
***Note**: To test the complete navigation from source to destination with significant distance between the source and destination, host the application on GitHub. This is because ngrok is limited in range and you would need to move away from the system where the ngrok server is running, for navigation. To avoid this we can host this application on GitHub.*

	Guide the student to host the application on GitHub to test the output later on after the class: • *<Your **github.io** url path>**/main.html*** • *Enter **source** or tap/click on the map to get exact latitude and longitude.* • *Enter **destination** or tap/click on the map to get exact latitude and longitude.* • *Click on the "**Navigate"** button.* *Output will be the AR Navigation path from the source to destination. The path* *Sample Output 1(With GitHub hosted Link)* *Sample Output 2 (With ngrok Link)* ***Note**:We need to get the exact latitude and longitude values of source and destination to run this application.*	

- *<Your **github.io** url path>**/main.html***

- *Enter **source** or **tap/click on the map** to get exact latitude and longitude.*
- *Enter **destination** or **tap/click on the map** to get exact latitude and longitude.*
- *Click on the **"Navigate"** button.*

AR Navigation



Navigate

***Output** will be the AR Navigation path from the source to destination.*

Teacher Guides Student to Stop Screen Share

WRAP UP SESSION - 5 mins

Teacher Starts Slideshow
Slide 16 to 20

Activity details
Following are the WRAP-UP session deliverables:

- Appreciate the student.
- Revise the current class activities.
- Discuss the quizzes.

WRAP-UP QUIZ
Click on In-Class Quiz

Continue WRAP-UP Session
Slide 21 to 26

Activity Details

Following are the session deliverables:

- Explain the facts and trivia
- Next class challenge
- Project for the day
- Additional Activity (Optional)

FEEDBACK

- **Appreciate and compliment the student for trying to learn a difficult concept.**
- **Get to know how they are feeling after the session.**
- **Review and check their understanding.**

Teacher Action	Student Action
You get Hats off for your excellent work!	*Make sure you have given at least 2 Hats Off during the class for:* 

PROJECT OVERVIEW DISCUSSION
Refer the document below in Activity Links Sections

Teacher Clicks ✖ End Class

Additional Activities	*Encourage the student to write reflection notes in their reflection journal using markdown.*
	
	Use these as guiding questions:
	• What happened today?
	  ○ Describe what happened.
	  ○ The code I wrote.
	• How did I feel after the class?
	• What have I learned about programming and developing games?
	• What aspects of the class helped me? What did I find difficult?
		The student uses the markdown editor to write their reflections in a reflection journal.

Activity	Activity Name	Links
Teacher Activity 1	Teacher Activity Reference Code	https://github.com/whitehatjr/PRO-C180-TA-Ref
Teacher Activity 2	Final Reference Code	https://github.com/whitehatjr/PRO-C180-Code-Ref
Teacher Activity 3	Sample Mapbox Directions API Response Reference Doc	https://obj.whitehatjr.com/45e69d29-2135-49df-89b7-c7d49f43a022.pdf
Teacher Activity 4	Output Sample 1 (With GitHub Hosted Link)	https://drive.google.com/file/d/1fMCFEpnMc8zTaEEY5MIIJXNkvb-6Fn8l/view?usp=sharing

Teacher Activity 5	Output Sample 2 (With ngrok Link)	https://drive.google.com/file/d/1q3uHRKSONp0XyBo269dg-wbX2UDed2BV/view?usp=sharing
Student Activity 1	Boilerplate Code	https://github.com/whitehatjr/PRO-C180-Student-Boilerplate
Teacher Reference 1	GitHub Pages Documentation	https://pages.github.com/
Teacher Reference 2	Project Document	https://s3-whjr-curriculum-uploads.whjr.online/617ce421-92f1-474e-8cea-b1accbce2430.pdf
Teacher Reference 3	Project Solution	https://github.com/whitehatjr/PRO-C180-Project-Solution
Teacher Reference 4	Visual-Aid	https://s3-whjr-curriculum-uploads.whjr.online/de4c18c5-f29f-4ffb-8c9a-e52761ac8c4f.html
Teacher Reference 5	In-Class Quiz	https://s3-whjr-curriculum-uploads.whjr.online/4d3555f7-a87e-4831-8dde-86a006caab16.pdf