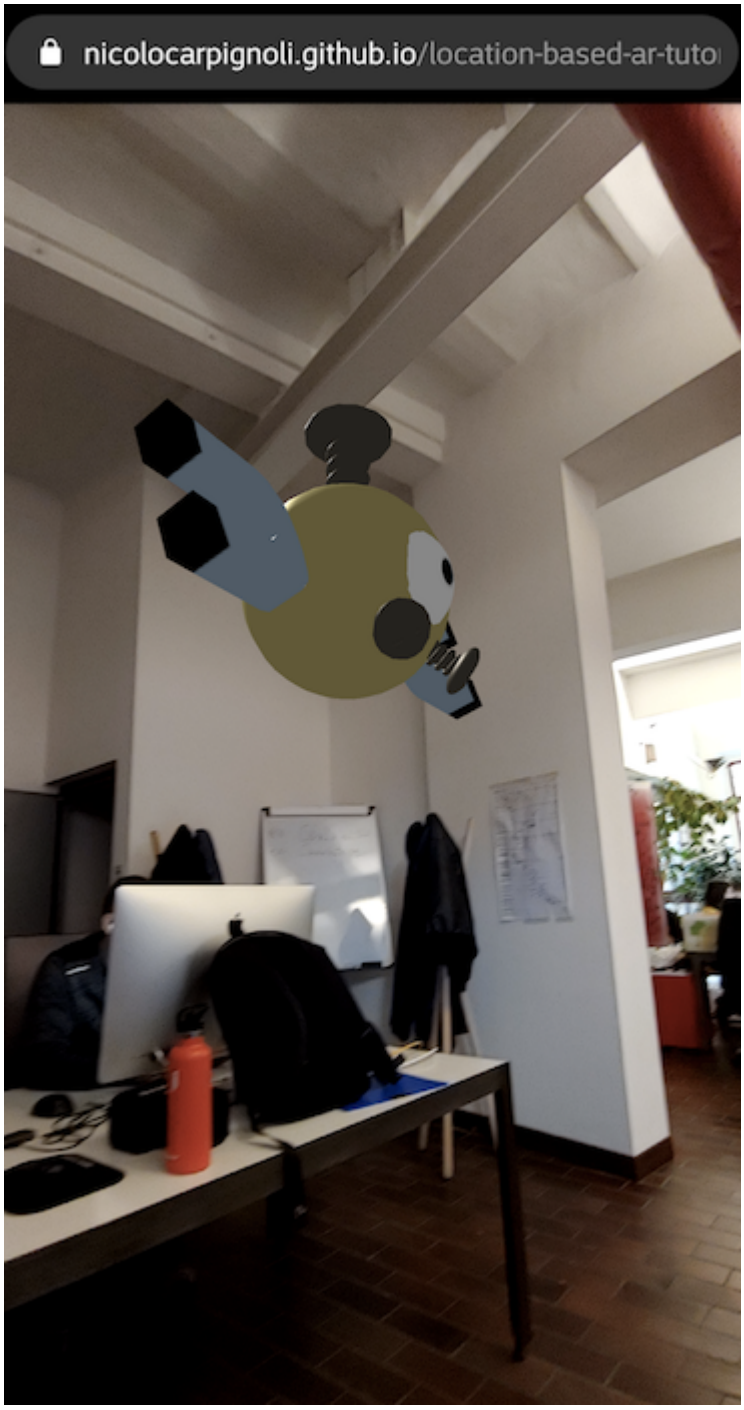
# Build your Location-Based Augmented Reality Web App

A tutorial for building Location-Based Augmented Reality apps that run on every browser, without any installation on your phone. For free.

Nicolò Carpignoli Follow

Nov 15, 2019 · 12 min read




A wild Magnemite appears on our studio.

> *Disclaimer: AR.js v3 is out, with a new official Documentation. If you found problem with this tutorial, you can look for more updated tutorials at: https://ar-js-org.github.io/AR.js-Docs/location-based/.*

## Location-Based AR on the Web

AR.js v2 introduced Location-Based AR on the Web for the first time. This enables new experiences in AR and a great opportunity for developers interested in Augmented Reality.

> Basically, it means that is now possible to deliver markerless AR experiences. The developers can specify places of interest, represented by real-world coordinates, on which the AR content will appear.

Furthermore, this new feature makes possible to combine both Marker-Based AR, the 'classic' AR.js way to augment reality, and the new Location-Based AR, based on GPS data.

For example, a classic combination of both features would be to show outdoor augmented information to the users, who are moving around holding their phone, and then, when a place of interest is spotted, they can move physically near it and enjoy a marker-based in-place experience.

Anyway, it makes a lot of sense to use the new Location-Based AR alone, to show situated informations about places near the user.

Some use cases? Think of learning experiences for history, biology, or treasure hunts for students. Or to use a Location-Based web app to spot open restaurants, to learn which historical building is that one 100 meters from us. Or again, to find out cinemas, bar, theaters with reviews, prices and so on, just moving the camera around, and interacting with a tap to learn more.

Yea, I know what you're thinking: "imagination is your only limitation".


A random Spongebob GIF joking about the “Imagination” concept.

On this tutorial, I will show you how to start with AR.js v2.0.x and Location-Based AR using the following ways:

- Adding places statically **only using HTML**
- Adding places statically using Javascript, also with a **minimal UX/UI**
- Adding places **dynamically** using Javascript and remote APIs.

You will also see how to deploy your first Web AR app on a remote server.

In this tutorial I’m assuming you have a basic knowledge of HTML and Javascript, and I suggest you to read the introductory article about AR.js and about its new Location-Based feature, first.

## First way: adding places only using HTML

### Template file

Let’s start creating a new folder and a new file, let’s call it index.html. Then copy and paste the following lines:

```
<!DOCTYPE html>
<html>
<head>
    <meta charset='utf-8'>
    <meta http-equiv='X-UA-Compatible' content='IE=edge'>
    <title>GeoAR.js demo</title>
    <script src='https://aframe.io/releases/0.9.2/aframe.min.js'></script>
    <script src="https://raw.githack.com/jeromeetienne/AR.js/master/aframe/build/aframe-ar.min.j
    <script src="https://raw.githack.com/donmccurdy/aframe-extras/master/dist/aframe-extras.load
    <script>
        THREEx.ArToolkitContext.baseURL = 'https://raw.githack.com/jeromeetienne/ar.js/master/th
    </script>
```

```
</head>

<body style='margin: 0; overflow: hidden;'>
    <a-scene
        vr-mode-ui="enabled: false"
                embedded
        arjs='sourceType: webcam; sourceWidth:1280; sourceHeight:960; displayWidth: 1280; displa
        <a-camera gps-camera rotation-reader></a-camera>
        </a-scene>
</body>
```

index.html hosted with ❤ by GitHub view raw

Basic HTML file for a Location-Based AR.js web app.

This is your entry point, the template application for a Location-Based AR.js app. Inside the `head` tag you can see imports for aframe, AR.js (containing the Location-Based code from v2), and donmccurdy's library to handle animation of 3D models. We will need the latter every time we'd like to animate a 3D model.

Refer to the official AR.js Location-Based documentation for doubts or extra details.

## Add your first place

Now it's time to define a place of interest. There is an important tip to remember:

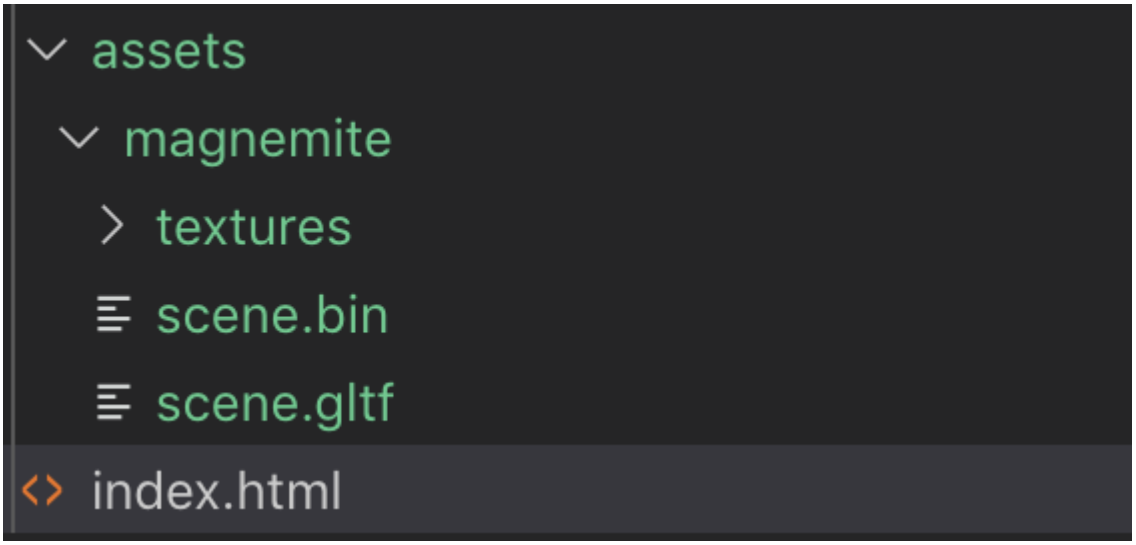
> *This kind of Augmented Reality makes more sense outdoor, where GPS signal is good and the user can move without restrictions.*

If you want to make some indoor experiments, try opening an application like Google Maps, activate GPS data on your phone, and see if the app can retrieve your position. If so, you can use it inside buildings, otherwise you have to move outdoor in order to test your code. Anyway, GPS signal is reflected by walls and even if seems to work indoor, results are not to be considered reliable.

Let's go back to the code. On the previous snippet you can see the `a-scene` element — the container of our AR application — and `gps-camera`, the element that calculates rotation, position of user and the distance between the user and places.

When adding a place, we contextually decide which content to show when we move our camera towards its direction. On this tutorial, we will show a 3D model: we will use a free model representing the Magnemite Pokémon, you can find it at this address. Download it, unzip it and create a folder called 'assets' on the same folder of your index.html file.

You should end up with a files structure like this:




Files structure for Location-Based web app using only HTML.

Inside your `a-scene` tag, add the following:

```
<a-entity gltf-model="./assets/magnemite/scene.gltf" rotation="0 180 0" scale="0.15 0.15 0.15" gps-entity-place="longitude: 12.489820; latitude: 41.892590;" animation-mixer/>
```

The `scale` attribute is used because that model is pretty big and the custom rotation will make the model 'look' towards the user. The `animation-mixer` attribute tells the model to use its built-in animation.

Using `gps-entity-place` attribute, we specify geospatial coordinates. I suggest you to insert valid data instead of those above (that are pointing to the Colosseum in Rome), referring to your current position, so you can see immediately the result. You can retrieve those data using this tool, inserting your current address.

Once you have done that, you are now ready to deploy your first web AR app!

### Run it on the Web

We will use Github and Github Pages feature. It's free and straightforward. First, you have to register on Github.com, the most used website that manages Open Source repositories. Then, you have to create a new repository, we can call it 'location-based-ar-tutorial'. After that, you have to set up git (if not already done) on your machine in order to push to the remote Github repository.

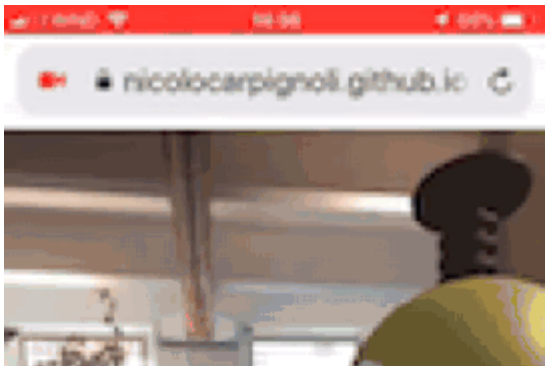
I cannot explain here about git theory and practice, we would go out of scope. I'll leave you some links in order to get done with this part, for those who are not familiar with it:

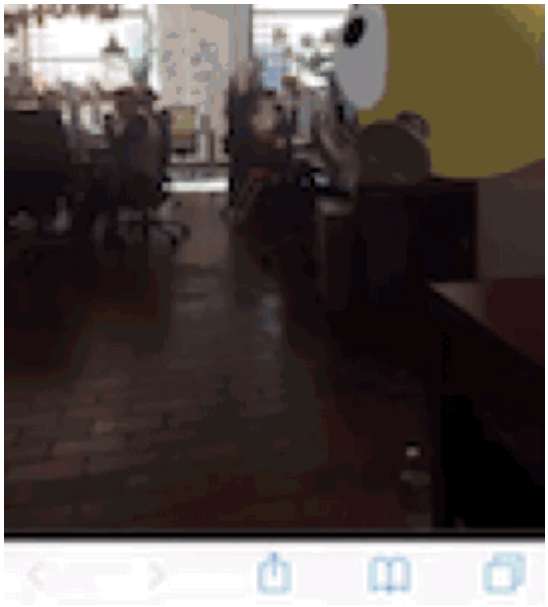
- Create a repository on Github
- Setup Git on your machine
- Fork your repository.

After this part, you should come up with a local and remote git repositories. Now, we have to deploy our code on a public, remote address.

Do you know the funny thing? That's already done! Try it yourself, navigate to: `https://<your-username>.github.io/location-based-ar-tutorial/` replacing <your-username> with your Github username and eventually change the last part of the URL — the repository name. Be sure to navigate using https protocol to avoid security problems with browsers.

You should be advised that the web page is requesting camera permissions and geolocation permissions: give both, and you should see a wild Magnemite moving near your position!





A Magnemite is moving in our studio. Only with a few lines of HTML.

📖 This should have taught you:

- How to create a Location-Based Web AR app using only HTML
- How to upload your files
- How to deploy your App on the web.

💡 Tip to remember: we have shown a 3D model (GLTF file) but we can show a lot of different content, thanks to `aframe` power! From geometry primitives to images and videos: you can check all kind of available content on the official aframe documentation.

## Second way: adding places statically using Javascript

Using Javascript, you will get an imperative approach, keep your HTML cleaner, and be able to do a lot more.

For example, you could load a lot of places iterating through an array, and add them programmatically, but most important, you can give interaction and a bit of UX/UI to your application. Let's see how.

### Add places from Javascript

As a first step, we will clean our HTML file and add places through Javascript. We will end up with the same behaviour as above with the following files (index.html and script.js follow):

```html
<!DOCTYPE html>
<html>
<head>
    <meta charset='utf-8'>
    <meta http-equiv='X-UA-Compatible' content='IE=edge'>
    <title>GeoAR.js demo</title>
    <script src='https://aframe.io/releases/0.9.2/aframe.min.js'></script>
    <script src="https://raw.githack.com/jeromeetienne/AR.js/master/aframe/build/aframe-ar.min.j
    <script src="https://raw.githack.com/donmccurdy/aframe-extras/master/dist/aframe-extras.load
    <script>
        THREEx.ArToolkitContext.baseURL = 'https://raw.githack.com/jeromeetienne/ar.js/master/th
    </script>
</head>

<script src="./script.js"></script>
<body style='margin: 0; overflow: hidden;'>
    <a-scene
       vr-mode-ui="enabled: false"
                embedded
        arjs='sourceType: webcam; sourceWidth:1280; sourceHeight:960; displayWidth: 1280; displa
        <a-camera gps-camera rotation-reader>
                </a-camera>
        </a-scene>
</body>
```

index.html hosted with ❤ by GitHub view raw

index.html cleaned, places are loaded from Javascript.

```javascript
window.onload = () => {
     let places = staticLoadPlaces();
     renderPlaces(places);
};

function staticLoadPlaces() {
    return [
        {
            name: 'Magnemite',
            location: {
```

```
            location: {
                lat: 44.496470,
                lng: 11.320180,
            }
        },
    ];
}

function renderPlaces(places) {
    let scene = document.querySelector('a-scene');

    places.forEach((place) => {
        let latitude = place.location.lat;
        let longitude = place.location.lng;

        let model = document.createElement('a-entity');
        model.setAttribute('gps-entity-place', `latitude: ${latitude}; longitude: ${longitude};`
        model.setAttribute('gltf-model', './assets/magnemite/scene.gltf');
        model.setAttribute('rotation', '0 180 0');
        model.setAttribute('animation-mixer', '');
        model.setAttribute('scale', '0.5 0.5 0.5');

        model.addEventListener('loaded', () => {
            window.dispatchEvent(new CustomEvent('gps-entity-place-loaded'))
        });

        scene.appendChild(model);
    });
}
```

script.js hosted with ❤ by GitHub view raw

Places added via Javascript.

Looking at the `staticLoadPlaces` function, you can see that places are statically loaded using an array. You can try to add more places, and you will see them while moving the camera if they are not too far from your position. If they are far but not too much, you will see them small, if they are closer to you, they will look bigger.

For the purpose of this example, though, we are going to use just one place.

## User Interaction

Noe, let's do something interesting. Using Javascript and `aframe` we can add basic user interaction.

First, we have to replace the entire index.html with the following code:

```
<!DOCTYPE html>
<html>

<head>
    <meta charset='utf-8'>
    <meta http-equiv='X-UA-Compatible' content='IE=edge'>
    <title>GeoAR.js demo</title>
    <script src='https://aframe.io/releases/0.9.2/aframe.min.js'></script>
    <script src="https://raw.githack.com/jeromeetienne/AR.js/master/aframe/build/aframe-ar.min.j
    <script src="https://raw.githack.com/donmccurdy/aframe-extras/master/dist/aframe-extras.load
    <script>
        THREEx.ArToolkitContext.baseURL = 'https://raw.githack.com/jeromeetienne/ar.js/master/th
    </script>
    <script src="./script.js"></script>
    <link rel="stylesheet" type="text/css" href="./style.css"/>
</head>

<body style='margin: 0; overflow: hidden;'>
    <div class="centered instructions"></div>
    <a-scene
        vr-mode-ui='enabled: false'
        embedded
        arjs='sourceType: webcam; sourceWidth:1280; sourceHeight:960; displayWidth: 1280; displa
    <a-camera gps-camera rotation-reader></a-camera>
</a-scene>
<div class="centered">
    <button data-action="change"></button>
</div>
</body>
```

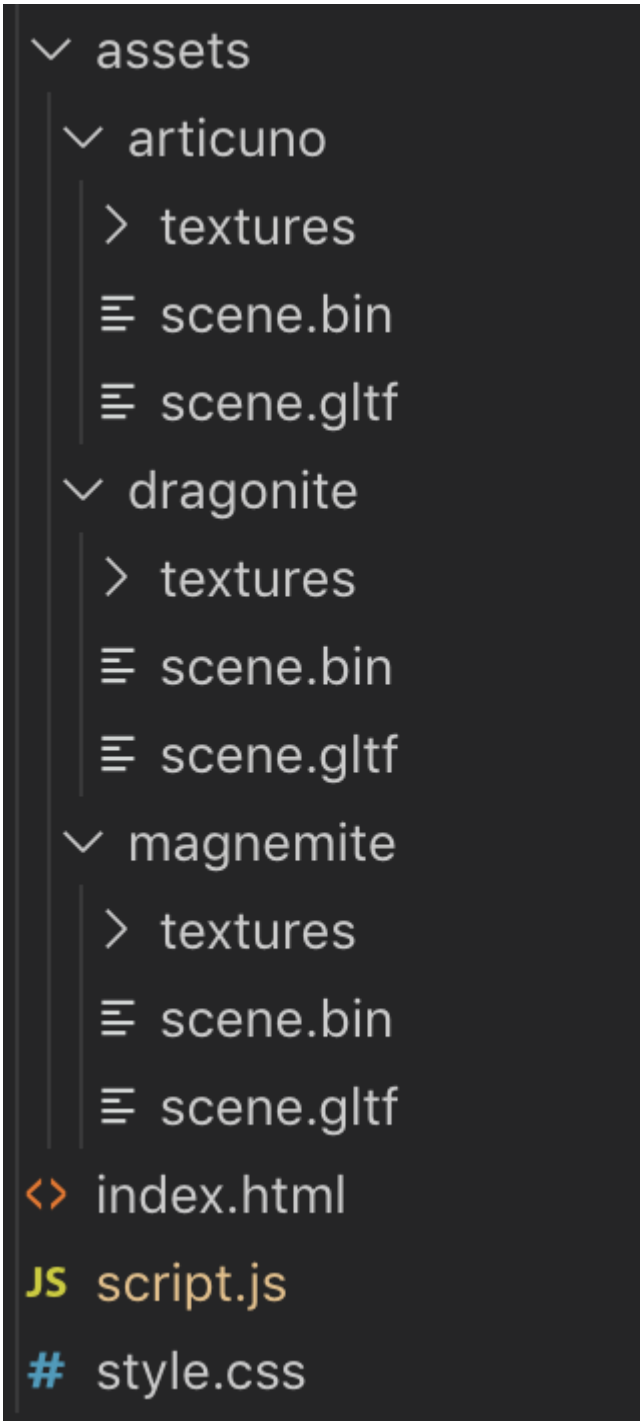
index.html hosted with ❤ by GitHub view raw

index.html for statically loaded places using Javascript example.

We have imported a script, a stylesheet, added a button and an empty div. That's all for the HTML side. You have also to create the new stylesheet called 'style.css' on the same folder of index.html.

The idea is to add a button with a click event handler, that will change our Pokèmon every time we interact with it! I found three free 3D models, one is the Magnemite we already used and the other two can be found at the following link: https://github.com/nicolocarpignoli/location-based-ar-tutorial/tree/master/static-places/assets.

So, first of all, download the assets and put them under the 'assets' folder. You will end up with the following files structure:



Files structure for this example.

Now, fill the stylesheet with the following CSS rules:

```css
.centered {
    height: 20%;
    justify-content: center;
    position: fixed;
    bottom: 0%;
    display: flex;
    flex-direction: row;
    width: 100%;
    margin: 0px auto;
    left: 0;
    right: 0;
}

button {
    display: flex;
    align-items: center;
    justify-content: center;
    border: 2px solid white;
    background-color: transparent;
    width: 2em;
    height: 2em;
    border-radius: 100%;
    font-size: 2em;
    background-color: rgba(0, 0, 0, 0.4);
    color: white;
    outline: none;
}

.instructions {
    position: fixed;
    top: 5%;
    font-size: 1.25em;
    color: white;
    z-index: 999999;
}
```



style.css

And now we add behaviour with Javascript. Things to note in the next script:

- we will add the place of interest after loading the main window
- we will add a click listener on the button that calls the ‘setModel’ function
- on ‘setModel’ function we will iterate through an array of models: each of them specifies a textual content for the upper DIV, the 3D model URL and custom properties — this because not all the models that you can find online will have the same rotation, orientation and size. We give them some kind of ‘configuration’ in order to have a consistent view for each of them (we can’t make Magnemite bigger than Articuno, right?)

> *You can change a lot of attributes: try to tune them according to your personal taste. For each property, the triple of values represents the property expressed in the X,Y and Z coordinates. Rotation is expressed in degrees, position in meters and scale is relative to initial scale (default to 1).*

```
window.onload = () => {
    const button = document.querySelector('button[data-action="change"]');
    button.innerText = '?';

    let places = staticLoadPlaces();
    renderPlaces(places);
};

function staticLoadPlaces() {
    return [
        {
            name: 'Pokèmon',
            location: {
                // lat: <your-latitude>,
                // lng: <your-longitude>,
            },
        },
    ];
}

var models = [
    {
        url: './assets/magnemite/scene.gltf',
```

```
        scale: '0.5 0.5 0.5',
        info: 'Magnemite, Lv. 5, HP 10/10',
        rotation: '0 180 0',
    },
    {
        url: './assets/articuno/scene.gltf',
        scale: '0.2 0.2 0.2',
        rotation: '0 180 0',
        info: 'Articuno, Lv. 80, HP 100/100',
    },
    {
        url: './assets/dragonite/scene.gltf',
        scale: '0.08 0.08 0.08',
        rotation: '0 180 0',
        info: 'Dragonite, Lv. 99, HP 150/150',
    },
];

var modelIndex = 0;
var setModel = function (model, entity) {
    if (model.scale) {
        entity.setAttribute('scale', model.scale);
    }

    if (model.rotation) {
        entity.setAttribute('rotation', model.rotation);
    }

    if (model.position) {
        entity.setAttribute('position', model.position);
    }

    entity.setAttribute('gltf-model', model.url);

    const div = document.querySelector('.instructions');
    div.innerText = model.info;
};

function renderPlaces(places) {
    let scene = document.querySelector('a-scene');

    places.forEach((place) => {
        let latitude = place.location.lat;
        let longitude = place.location.lng;

```

```
69          let model = document.createElement('a-entity');
70          model.setAttribute('gps-entity-place', `latitude: ${latitude}; longitude: ${longitude};`
71
72          setModel(models[modelIndex], model);
73
74          model.setAttribute('animation-mixer', '');
75
76          document.querySelector('button[data-action="change"]').addEventListener('click', functio
77              var entity = document.querySelector('[gps-entity-place]');
78              modelIndex++;
79              var newIndex = modelIndex % models.length;
80              setModel(models[newIndex], entity);
81          });
82
83          scene.appendChild(model);
84      });
85  }
```

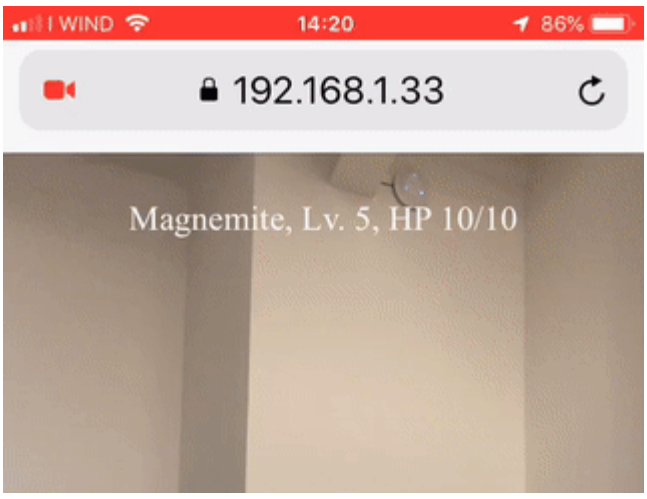
script.js hosted with ❤ by GitHub view raw

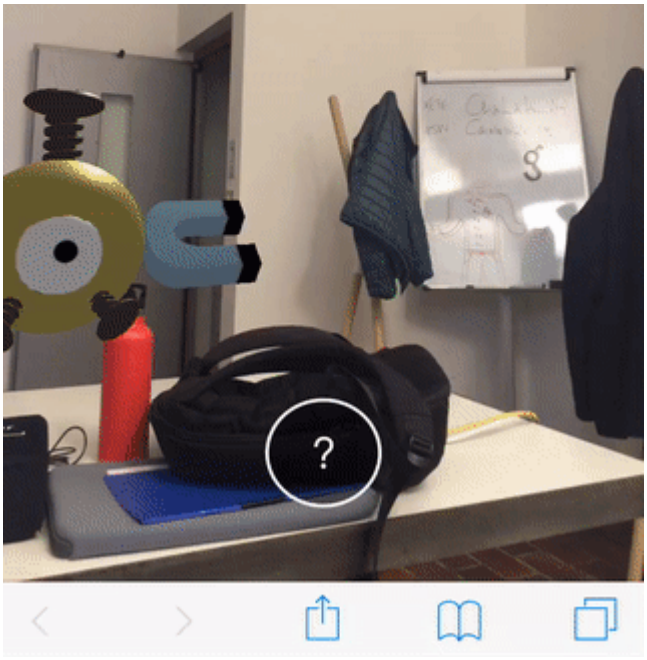
script.js

Important: before you go on, at line 18 of script.js you have to set valid place coordinates, as done in the examples before.

## Gotta Switch them all!

The result should be a very basic Web AR app with a minimal UX/UI with only one button, that switches the Pokèmon when clicked. The Pokèmon positions should be approximately the same, after every switch. Some of them are moving, other not, accordingly to their built-in animation.





Switch 3D Pokémon model just clicking on the HTML button.

📖 This should have taught you:

- How to load places from Javascript, so you can add more of them with few lines of code
- How to style your Web App and add UX/UI basics using click listeners on HTML elements
- How to change size, position, rotation and assets of the AR content.

💡 Tip to remember: You can also interact directly with 3D models or AR content in general. I don't advise it because of bad reliability when click is not on the centre of the screen. Most of the times it's possible to achieve the same results (with better performances) using DOM elements like on the example above. If you're still interested and want to experiment with click events on AR content, you should look at this article: it's about marker-based feature but the click concept and related code remains the same for 'gps-entity-place' elements.

## Third way: adding places dynamically using Javascript

Here comes the most interesting way.

When you statically add places of interest on your Web app, **you have to know their position at development time**. What happens if you deploy your app and give the URL to a friend, or share it on socials? If users are not near the position you arbitrarily set, they won't see anything. Only the camera video stream.

You cannot programmatically know the place where your users will be when they will open your web app. And you should not.

The idea is to first retrieve users position, and then dynamically load places of interest near them. In order to do that, we will need external APIs. I found a good service from Foursquare Places API. For this example, we will use them, but you can choose to use also something else: in that case, only the data-retrieve function will change.

### Register to Foursquare APIs and get credentials

Navigate to https://developer.foursquare.com and click on 'Create Account'. It's free, of course. After the process, you will receive a confirmation email. Open the email, click on the link and login again. Now you're ready to create your first 'app'.

Click on 'Create new app'. On the 'App name' you can type something like 'location-based-ar' and on 'Company URL' I set 'https://localhost:3000', my local environment URL.

On the next page, keep note of CLIENT ID and CLIENT SECRET values. You will need them later. We will use, on our app, the 'Places API', you can look here for the documentation.

### Let's fetch some data

For this example, we can re-use our cleaned index.html from previous example, without custom UI, and change the way we add places, on the script.

There's one configuration that I suggest you to add: on `gps-camera` element, it's better to add an attribute called `minDistance`. It says to AR.js to hide elements that are too close to the user. What means 'too close' is specified with the attribute value, like this: `gps-camera="minDistance: 40;"` : this means 'do not show elements that are 40 meters or less close to the user'.

> *When we are moving on locations with a lot of places of interest, I suggest to not have too much content on the screen: if we are very close to a place of interest, it's strange to see the AR content so big and so close to us. It's better, in such cases, to deliver a different kind of experience (show an alert, a message for the user, and so on).*

You can also remove the 3D assets folder and its content, we will not need them. Finally, the index.html will look like the following:

```
<!DOCTYPE html>
<html>

<head>
    <meta charset='utf-8'>
    <meta http-equiv='X-UA-Compatible' content='IE=edge'>
    <title>GeoAR.js demo</title>
    <script src='https://aframe.io/releases/0.9.2/aframe.min.js'></script>
    <script src="https://raw.githack.com/jeromeetienne/AR.js/master/aframe/build/aframe-ar.min.j
    <script>
        THREEx.ArToolkitContext.baseURL = 'https://raw.githack.com/jeromeetienne/ar.js/master/th
    </script>
    <script src="./script.js"></script>
</head>

<body style='margin: 0; overflow: hidden;'>
    <a-scene
        vr-mode-ui='enabled: false'
        embedded
        arjs='sourceType: webcam; sourceWidth:1280; sourceHeight:960; displayWidth: 1280; displa
    <a-camera gps-camera="minDistance: 40;" rotation-reader></a-camera>
</a-scene>
</body>
```

index.html hosted with ❤ by GitHub — view raw

index.html for dynamically added places example.

The idea is to show an icon and name for each place of interest near the user. Places are added dynamically.

Let's now retrieve user position and fetch some locations. We will show, for each place, an 'a-link'. It's a customizable entity that shows a text and if clicked, it may redirect the

user to an external webpage, if specified with the 'href' attribute. You can customize its visual aspect, default is a red circle. Check out the documentation to learn how.

```
// getting places from APIs
function loadPlaces(position) {
    const params = {
        radius: 300,    // search places not farther than this value (in meters)
        clientId: '<YOUR-CLIENT-ID>',
        clientSecret: 'YOUR-CLIENT-SECRET',
        version: '20300101',    // foursquare versioning, required but unuseful for this demo
    };

    // CORS Proxy to avoid CORS problems
    const corsProxy = 'https://cors-anywhere.herokuapp.com/';

    // Foursquare API (limit param: number of maximum places to fetch)
    const endpoint = `${corsProxy}https://api.foursquare.com/v2/venues/search?intent=checkin
        &ll=${position.latitude},${position.longitude}
        &radius=${params.radius}
        &client_id=${params.clientId}
        &client_secret=${params.clientSecret}
        &limit=30
        &v=${params.version}`;
    return fetch(endpoint)
        .then((res) => {
            return res.json()
                .then((resp) => {
                    return resp.response.venues;
                })
        })
        .catch((err) => {
            console.error('Error with places API', err);
        })
};


window.onload = () => {
    const scene = document.querySelector('a-scene');

    // first get current user location
    return navigator.geolocation.getCurrentPosition(function (position) {

        // than use it to load from remote APIs some places nearby
        loadPlaces(position.coords)
```

```
41          loadPlaces(position.coords)
42              .then((places) => {
43                  places.forEach((place) => {
44                      const latitude = place.location.lat;
45                      const longitude = place.location.lng;
46
47                      // add place name
48                      const placeText = document.createElement('a-link');
49                      placeText.setAttribute('gps-entity-place', `latitude: ${latitude}; longitude
50                      placeText.setAttribute('title', place.name);
51                      placeText.setAttribute('scale', '15 15 15');
52
53                      placeText.addEventListener('loaded', () => {
54                          window.dispatchEvent(new CustomEvent('gps-entity-place-loaded'))
55                      });
56
57                      scene.appendChild(placeText);
58                  });
59              })
60      },
61          (err) => console.error('Error in retrieving position', err),
62          {
63              enableHighAccuracy: true,
64              maximumAge: 0,
65              timeout: 27000,
66          }
67      );
68  };
```

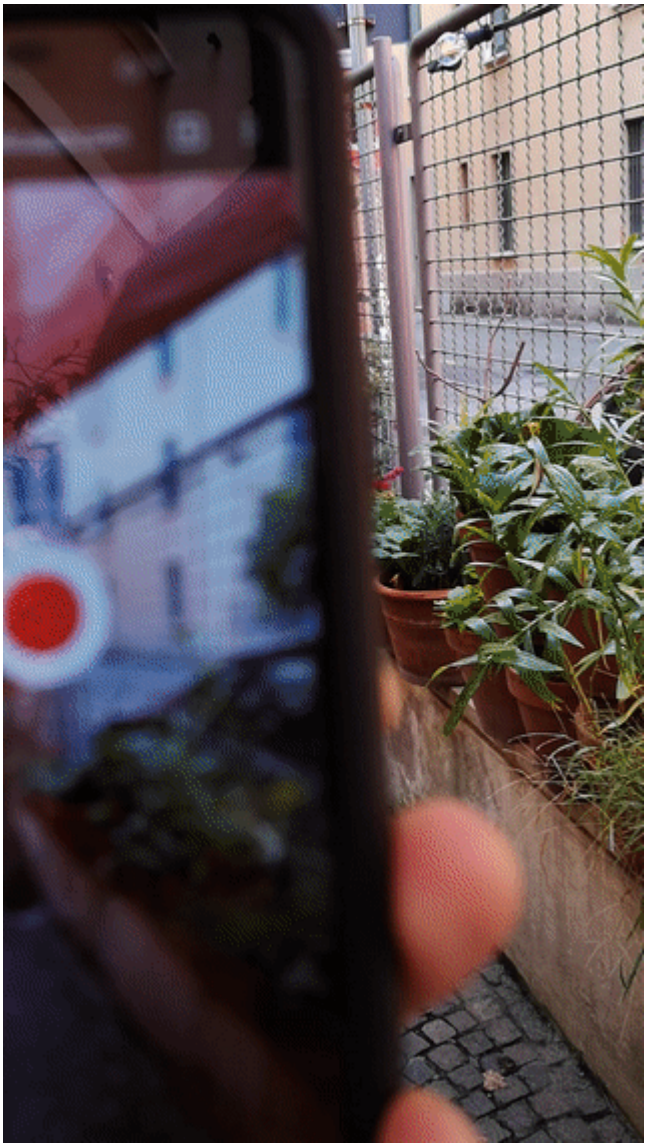
script.js hosted with ❤ by GitHub view raw

script used to load places dynamically using Foursquare APIs.

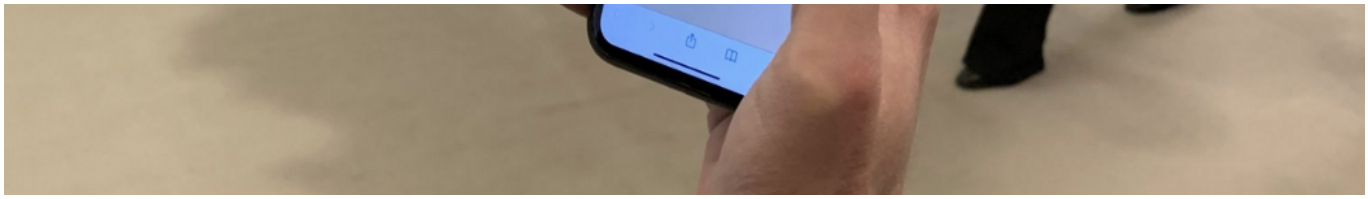
This won't work yet, you will need to add your CLIENT ID and CLIENT SECRET data on lines 5 and 6.

Now, try to commit, push and visit again the github.io URL for your application. Keep in mind that Github.io pages URLs follow the structure of your project: if you have added this new example under a different folder, you will have to visit `https://<your-username>/<your-repository>/<your-folder>/`

You will end up with something that I already show you on previous article on Medium, about this project:



Move around to see places of interest where they are in the real world.



Show a-link with places of interest names. Click on each of them will redirect to an URL if defined.

📖 This should have taught you:

- How to dynamically fetch places of interest starting from user position
- How to customize data fetching (limit, radius, etc.), looking at API documentation
- How to customize AR.js location-based system (see doc).

💡 Tip to remember: You already know that is possible to add different kind of content, using `aframe`. And you know that interaction is always possible using DOM elements and events. As written above, to interact directly with AR content is a bit cumbersome, but it can be achieved. Here's a little snippet for you to start with:

```
1   // this click listener has to be added simply to a click event on an a-entity element
2   const clickListener = function (ev) {
3       ev.stopPropagation();
4       ev.preventDefault();
5
6       const name = ev.target.getAttribute('name');
7       const el = ev.detail.intersection && ev.detail.intersection.object.el;
8
9       if (el && el === ev.target) {
10          // after click, we are adding a label with the name of the place
11          const label = document.createElement('span');
12          const container = document.createElement('div');
13          container.setAttribute('id', 'place-label');
14          label.innerText = name;
15          container.appendChild(label);
16          document.body.appendChild(container);
17
18          setTimeout(() => {
19              // that will disappear after less than 2 seconds
```

```
            container.parentElement.removeChild(container);
        }, 1500);
    }
};
```

snippet.js hosted with ❤ by GitHub view raw

click on AR content events handling.

Want to learn more about about this? Full code here.

That's all for now! These are the three, different ways to use AR.js location-based feature. The first one is the simpler but still efficient, and it's recommended for people who's not comfortable with scripting, or want to keep the app clean with just one file.

The second and the third way are the most interesting and let the developers create full AR Web Apps: you can 'emulate' those AR mobile apps we're comfortable with, starting with a loading page, a tutorial, a cool UI and eventually user interaction.

> This is possible thanks to standard Web technologies, and it's much, much faster to develop and deploy than using mobile app technologies. Furthermore, users will not have to install an app to reach your AR experiences, they will just visit an URL or scan a QR Code.

Please let me know about your AR experiences developed with new AR.js location-based! Can't wait to hear from you.

*My thanks go to utopiah and sinanatra for their help in reviewing this article.*

> **Chialab** is a design studio. We create relations between things and people, handling the strategy, planning, software and contents.
>
> https://www.chialab.it.

Thanks to Giacomo Nanni.

Web Development Augmented Reality Design Programming JavaScript