Getting started with GeoJson

This exercise is meant to give you a basic understanding of how to use GeoJSON. The example does not require a database but will instead use [this simple API](), to perform the required math. To make the following exercises a bit more fun/realistic, we will add a simple problem domain to justify what we do:

Assume that our Game-Project, introduced for this semester, was to include additional features, as described by the following simple rules/requirements, for which we need to provide a proof of concept server, that eventually can be used by a mobile client:

- *Users may only be located in a certain area, given as a GeoJson polygon*
- *It must have a way to check whether a user is inside this area or not, given the users geo-location*
- *It must have a way to find all nearby Users given a geolocation*
- *It must have a way to find the distance (in meters - straight line) between two users given their geo-location*

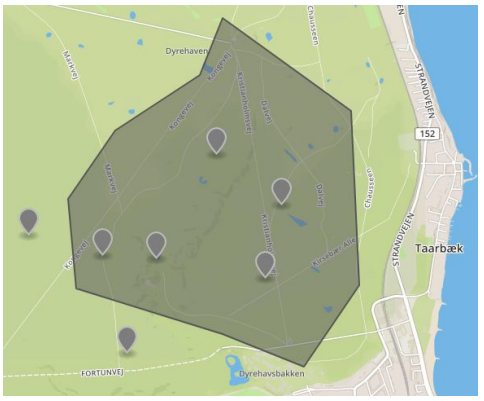
The following will provide a simple (database-less) proof of concept API for these requirements.

The Server for the exercise

1. Create a new node project (npm init), install express,
2. Install this library to provide GeoJson help: **npm install geojson-utils --save** (note: this library is probably the best/most all round: http://leafletjs.com, but the suggested API is extremely simple to use). You should also install express, which we will use as the server.
3. Add a single file *app.js* and add this content to the file:

```
const express = require('express')
const app = express()
app.get('/', (req, res) => res.send('Geo Demo!'))
app.listen(3000, () => console.log('Example app listening on port 3000!'))
```

Getting test data for the exercise.



We need some test data, for the game area, and also, some test users. Follow [this link](), and draw a “game area” (polygon) and a number of “users”, both inside and outside the area as sketched in this figure. Create a new javaScript file, `gameData.js`, in your project, and copy the “feature” object of the polygon into a property which you should export as sketched below:

```
const gameArea =  {
      {
        "type": "Feature",
        "properties": {},
        "geometry": {
          "type": "Polygon",
        ...
module.exports = {
gameArea
}
```

In the same file, add a new const; players, and initialize it with the features of the "users" you created above, as sketched below. Add a unique name to the properties object for each:

```
const players= [
  {
    "type": "Feature",
    "properties": {"name":"Peter"},
    "geometry": {
      "type": "Point",
      "coordinates": [
        12.576169967651367,
        55.784488990708795
      ]
    }
  },
 …
]
```

Add this to list to you export declaration: `module.exports = { gameArea,players }`

(Longitude, Latitude) or (Latitude, Longitude)

Unfortunately, there is no fixed answer to that question. The GeoJSON data we just created uses longitude,latitude while Google Maps, and other client-side technologies (including Airbnb's React Native Components) uses latitude,longitude.
This is important to know since otherwise, you are going to spend a lot of time debugging when your backend data is originally sent from a client.
Go to http://google.com/maps and navigate/zoom-in to the same area as used above. Observe, that the order is the opposite of our Geo-JSON data. REMEMBER THIS.

Rest Endpoints:

Implement and test the three endpoints described in the following.

Hints:

1. Use the examples to see how to use the GeoJSON-utilities, and import the utilities into app.js as: `const gju = require('geojson-utils');`
2. Import the test data into app.js as gameArea and `players`.
3. For all the three endpoints, convert the incoming location into a Point geometry: `const point = {"type":"Point","coordinates":[req.params.lon,req.params.lat]}`
4. For **endpoint-1**, use the method: `pointInPolygon(..)` (Make sure your `gameArea` polygon matches the polygon used in the examples)
5. For **endpoint-2** use the method: `geometryWithinRadius(..)` (Make sure only to use only the geometry part of you players, when you compare.
6. For **endpoint-3,** first find the other player in the list of players, and then use the method: `pointDistance(..)`

Check whether the caller is located in the Game Area

GET: `/geoapi/isuserinarea/:lon/:lat`

Arguments: longitude and latitude

Return Responses:

User found	User was not found
<pre>{ "status": true, "msg": "Point was inside the tested polygon" }</pre>	<pre>{ "status": false, "msg": "Point was NOT inside tested polygon" }</pre>

Example call (see response above):
`http://localhost:3000/geoapi/distanceToUser/12.568359375/55.78540598108821`

Find Players near the caller

GET: `/geoapi/findNearbyPlayers/:lon/:lat/:rad`

Arguments: longitude, latitude and the radius (in meters) to search for other players

Return response

Nearby Players Found	NO nearby Players Found
<pre>[{ "type": "Feature", "properties": {"name": "Peter"}, "geometry": { "type": "Point", "coordinates": [12.576169967651367, 55.784488990708795] } }, ...]</pre>	<pre>[]</pre>

Example call (see response above):

`http://localhost:3000/geoapi/findNearbyPlayers/12.568359375/55.78540598108821/1000`

Find Distance between caller, and another player

GET: `/geoapi/distanceToUser/:lon/:lat/:username'`

Arguments: longitude, latitude and userName to find distance to

Return response

User found	User was not found
<pre>{ "distance": 498.88825733403075, "to": "Peter" }</pre>	<pre>{ "msg": "User not found" } Statuscode = 404</pre>

Example call (see response above):

http://localhost:3000/geoapi/distanceToUser/12.568359375/55.78540598108821/Peter

Use this URL to verify your result: https://gps-coordinates.org/distance-between-coordinates.php