Game Backend REST

This exercise assumes you have completed most of the tasks given for day1-4 in period 2.
Here we will design the REST backend, required for the "game" related part of the API.
The following will introduce tasks in a prioritized order, so if you don't have time for all, implement what you can in the given order.
In the following, we assume that a player (or a team) is represented by the existing IGameUser in our existing project (we can assign the role team to make this more obvious)

1) Find nearby players (teams)

Observe for this endpoint, that we are not using Basic-authentication, but manually include the userName and Password with each request, actually just like what the browser does, with basic-authentication. This, to simplify matters when used from a mobile app. Just use your existing password support to test the validity of the provided credentials.
Distances below must be given in meters.

POST: /gameapi/nearbyplayers

Request JSON:

```
{"userName":"team1", "password":"secret", "lat":3, "lon": 5,"distance": 3}
```

Response JSON (if authenticated): (empty array if none found)

```
[{"userName": "team2", "lat": 4, "lon" : 10}, {..}, ..]
```

Response JSON (if not authenticated):

```
{message: "wrong username or password", code: 403} (StatusCode = 403)
```

Requirements for this endpoint:

A player sends his credentials, together with his current location and the radius inside which he requests a list of nearby players. On the server the position must be stored in a Mongo Location document for the given user. This document must be given a *limited lifetime* (60 second for example) to avoid old invalid locations. Obviously this means that clients constantly must call the method with updated positions.
For the response, the server must return a JSON object with all teams and their locations found.

Please observe that latitude, longitude is given in the order "natural" for a Client-API
Hints

1. You will receive start help for this part, either via a video or in the "class"
2. Create a gameFacade.ts file, with this method (remember to export it):
   `nearbyPlayers(userName:string,password:string,lon:number,lat:number,distance:number)`
3. The domain model given for position, was just that, a domain model. Use this interface to represent Position in your code and database. Observe the redundant data which removes the need for a cross collection lookup (userName and name is not likely to change often).

```
interface IPosition {
   lastUpdated:Date,
   userName:string,
   name:string,
   location: {
       type: string,//No(easy) way in ts, to restrict this to the only legal value "point"
       coordinates : Array<number>
   }
}
```

Implement `nearbyPlayers` as sketched below

1. Check whether the user exists, and if, whether the password matches. If not, throw an error.
2. Update the `position` for the User. Use `xxxxx.findOneAndUpdate(....)` (provide it with the **upsert** option, so it will create the Document, if it does not exist). Remember to update the date field `lastUpdated` also.
3. Now create a utility method which will find all nearby users, given a point and distance (use this example as a template, but replace `db.` places with the name you have given your position-collection.
4. Use `map` on the found list of users, to reformat data as requested for the endpoint
5. **IMPORTANT:** Somewhere in your facade, remember to set the two required indexes on the collection (`ttll` and `2dspere`)
6. Once the facade method is ready and tested, implement the requested endpoint and relevant tests for the endpoint.

Test data for the API implemented above

In order to test the API, implemented above, we need some nearby users. Implement a few test users and a matching Location Documents for each, with a location nearby the location you will use for "yourself" (remember, eventually this location will come from a phone, so nearby is either near the school or your home).
Since we have added a ttl-index that removes all Location documents older than 60 seconds, or whatever you have decided, you should create these documents with a lastUpdated value out in the *future*, as sketched below:

```
lastUpdated: "2022-09-25T20:40:21.899Z"
```

2) Deploy the project

Since we eventually need to access this endpoint from a mobile device it MUST BE deployed.

Do this, following the guidelines in this playlist:

3) Get Post, and its task, if the post is reached

Initially, as seen below, this endpoint does not require authentication. Assume a team (somehow) has been given information about which post to locate, and directions to get there. In the final game the mobile team will constantly send this request until eventually the post has been reached and the task for the post provided.

`GET: /gameapi/getPostIfReached`

Request JSON:

```
{"postId":"post1", "lat":3, "lon": 5}
```

Response JSON (if found):

```
{"postId":"post1", "task": "2+5", isUrl:false}
```

Response JSON (if not reached):

```
{message: "Post not reached", code: 400} (StatusCode = 400)
```

Requirements for this endpoint:

These are pretty simple. Use the provided inputs as your query criteria, and search for the Post. If found provide a DTO response according to the responses given above.

Hints

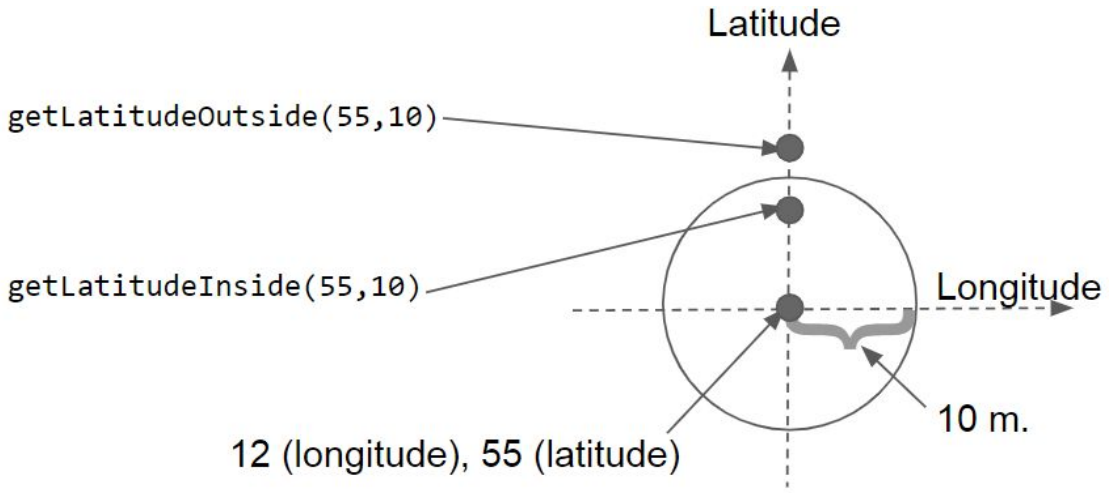
1. Use the existing gameFacade (created in part-1) and add this method:

   ```
   static async getPostIfReached(postId:string,lat:number,lon:number):Promise<any>{}
   ```

2. You might also want to implement an `addPost(...)` method, but you could also decide just to add post's right into the database using the driver-API.
3. You can use this interface to represent a Post (observe that the three domain-classes has been merged into a single entity)

   ```
   interface IPost{
      _id: string,
      task: {text:string, isUrl: boolean}
      location: {
          type: string
          coordinates : Array<number>
      }
   }
   ```

4. Remember to add the `2dsphere` index to the collection
5. When implementing the facade, define a radius inside which any coordinate provided is interpreted as "the Post has been reached". In the start code for this part (utils/geoutils.ts) you will find two small utility methods which can adjust the latitude value for any coordinates (near Lyngby) to get a value inside or outside a post as sketched in this figure.



6. When the facade-method is implemented and tested, implement and test the endpoint

4) Make userName unique

If you think this might take you more than 10-15 minutes, skip this part.
Change the UserFacade to add an index and a unique-index on the userName.
Provide the required unit-tests to verify this new feature.

5) Move the GameApi implemented without a database into this project

This will provide you with one single project, with all your location related backend code.

6) Come up with additional proof-of-concept examples for the Game

If you have time, and ONLY if, come up with additional proof-of-concept example related to the Game Project.