ILP CW1 Specifications (Programming Task)

06.10.2024

The first assignment has been designed as a preliminary to tackling larger and more complex situations in the second assignment.

Your main tasks can be summarized as follows:

1. Create a Java-REST-Service
   - Preferred with Spring Boot, though other frameworks can be used as well
   - Port 8080 is consumed for incoming requests
   - Implement the necessary endpoints for the POST and GET requests (see below)
   - Proper parameter handling (check for invalid data) – see below
   - Proper return code handling – see below
   - JSON handling

2. Place the service in a docker image
   - **amd64** as target architecture – **not arm64** (this is relevant for Mac users!)

3. save the docker image in a file called **ilp_submission_image.tar**
   (it is in TAR format anyhow)

4. **place** the file **ilp_submission_image.tar into your root directory of your solution**

   Your directory would look something like this:

   ilp_submission_1 *(your root directory)*
   ilp_submission_image.tar
   src (the Java sources…)
   main
   …
   …

5. Create a ZIP file of your solution directory
   - Image
   - Sources
   - IntelliJ (or whatever IDE you are using) project files

6. upload the ZIP as your submission in Learn

The REST-Service must provide the following endpoints:

1. **`uuid`** (GET)

   Return your student id as a string **without any further** formatting. So, just the plain string with the s in front like in s1234567.

2. **`distanceTo`** (POST)

   return the Euclidian distance between position 1 and position 2 as a double value in the body.

   The body contains (LngLatPairRequest.json):

```
{
  "position1": {
    "lng": -3.192473,
    "lat": 55.946233
  },
  "position2": {
    "lng": -3.192473,
    "lat": 55.942617
  }
}
```

   This method will be called with valid and invalid data (semantically and syntactically). Only for proper syntactically correct records, you are supposed to return a result (and 200); otherwise, you shall return 400 (bad request)

3. **`isCloseTo (POST)`**

   return true if the two positions are close (< `0.00015`), otherwise false.

   The body contains (LngLatPairRequest.json):

```
{
  "position1": {
    "lng": -3.192473,
    "lat": 55.946233
  },
  "position2": {
    "lng": -3.192473,
    "lat": 55.942617
  }
}
```

   This method will be called with valid and invalid data (semantically and syntactically). Only for proper syntactically correct records, you are supposed to return a result (and 200); otherwise, you shall return 400 (bad request)

4. **nextPosition (POST)**

return the next position as LngLat (LngLat.json) for a start position and an angle.

The body contains (NextPositionRequest.json):

```
{
  "start": {
    "lng": -3.192473,
    "lat": 55.946233
  },
  "angle": 45
}
```

This method will be called with valid and invalid data (semantically and syntactically). Only for proper syntactically correct records, you are supposed to return a result (and 200); otherwise, you shall return 400 (bad request).

Every angle in the valid range 0..360 should be considered; the limit of the drone movement is only relevant for generating flight paths and not here.

Based on the information provided in the ILP general document, you can work out the next point based on the start and the angle.

5. **isInRegion (POST)**

return true if the point is inside the named region (including the border), otherwise false. A region is usually rectangular yet can be any polygone.

The body contains (https://github.com/mglienecke/IlpDataObjects/blob/main/src/main/json/IsInRegionRequest.json):

```
{
  "position": {
    "lng": 1.234,
    "lat": 1.222
  },
  "region": {
    "name": "central",
    "vertices": [
      {
        "lng": -3.192473,
        "lat": 55.946233
      },
      {
        "lng": -3.192473,
        "lat": 55.942617
      },
      {
        "lng": -3.184319,
        "lat": 55.942617
```

```
        },
        {
          "lng": -3.184319,
          "lat": 55.946233
        },
        {
          "lng": -3.192473,
          "lat": 55.946233
        }
      ]
    }
}
```


This method will be called with valid and invalid data (semantically and syntactically). Only for proper syntactically correct records, you are supposed to return a result (and 200); otherwise, you shall return 400 (bad request).

If the region is not closed by the data points, you should assume invalid data.
The last point in the above list closes the region (going back to the origin) - if omitted, it would be an open region and invalid.

All data necessary for POST-endpoints will be passed in the body of the request as a JSON data structure and any results will be returned there too.

To help you with your assignment, there is a GitHub repository, where all JSON data is shown exemplary as well as implementations of the data classes.
Have a look: https://github.com/mglienecke/IlpDataObjects

The following should be considered when implementing the REST-service:

- Do proper checking for URLs, data, etc. Don't handle anything not accurate (you will receive error data and requests!)
- You are using http, not https
- Your endpoint names must match the specification (so **no global prefix** like /api, /ilp, etc.). Your API must be reachable at i.e. **http://server:8080/uuid**
- Test your endpoints using a tool like Postman or curl. Plain Chrome / Firefox, etc. will do equally for the GET operations
- The filename for the docker image file must be exactly as defined as well as the location of it in the ZIP-file. Should you be in doubt, use copy & paste to get the name right

Should you need help:

- See the literature links in the "Library" section in Learn. You should find most information there

- If you cannot find an answer to your question, please post it on Piazza, though try finding it yourself first, please (as we have only limited capacity)

Disclaimer: We will not be able to answer any question on piazza less than 3 days before the assignment deadline

So, please make sure you start the assignment in good time.

Marking:

This programming task has a maximum mark of 30 / 100 points in relation to the entire ILP course.

The marks will be purely allocated on auto-tests with the following values:

- (4) proper runnable docker image
- (26) Proper behavior (functionality)

Endpoint	Comment	Points
uuid (1)		1
distanceTo (5)	Correct call	2
	Semantic error call	1
	Syntactical error call	1
	Empty body call	1
isCloseTo (5)	Correct call	2
	Semantic error call	1
	Syntactical error call	1
	Empty body call	1
nextPosition (7)	Correct call	4
	Semantic error call	1
	Syntactical error call	1
	Empty body call	1
isInRegion (8)	Correct call	4
	Semantic error call	1
	Syntactical error call	1
	Empty body call	1
	Open vertices data	1

Should you fail to provide a runnable docker image according to the specification or provide no source code in the submission, no marking will be possible, and you will receive 0 points.