ILP CW1 Specifications (Programming Task)

29.09.2024

This first microservice of yours will be extended in CW2 and perhaps your own implementation and provides many useful functionalities which you need later for CW2.
The main intention is to equip you with the knowledge and skills to tackle larger and more complex situations as well as providing quick feedback for you how your coding, design and implementation – including testing – is getting along. Most tasks in this assignment should be rather good to implement, yet due to error checking and handling special scenarios, can become a bit more complex.

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

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 root directory (see point 4 above) which contains
   - Docker Image (as tar-file)
   - Sources
   - IntelliJ (or whatever IDE you are using) project files

6. upload the ZIP as your submission in Learn

The REST-Service which you are developing must expose the following endpoints (which always must start with /api/v1/):

1. **actuator/health** (GET)

   This endpoint is the only one that has no /api/v1 in front of it. It must return the standard Spring response (https://docs.spring.io/spring-boot/api/rest/actuator/health.html) with at least one member in the JSON-data

   ```
   {
         "status": "UP"
   }

   More data is ok...
   ```

2. **uid** (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.

3. **distanceTo** (POST)

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

   The body contains:

   ```
   {
     "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. **isCloseTo (POST)**

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

   The body contains:

   ```
   {
     "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)

5. **nextPosition (POST)**

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

The body contains:

```
{
  "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).

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

6. **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 polygon.

The body contains:

```
{
  "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.

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

- Your endpoint names must match the specification (so **use the global prefix** /api/v1 and nothing else – aside the endpoint name) and your API must be reachable via i.e. **http://server:8080/api/v1/uid**
- Extra data members in JSON as pass-in-parameters to your service, **must** be ignored, your response must be accurate and according to the specification.

- 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
- 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 25 / 100 points in relation to the entire ILP course.

This distribution will be applied:

- 10 points for style, code quality, naming and structure
    - Code quality & style (0..2) with regards to readability, commenting, clearness of code
        - 0 = low quality
        - 1 = some quality, many challenges
        - 2 = very good quality
    - Good naming (0 / 1) of variables and methods / classes
        - 0 = no good naming
        - 1 = naming makes sense and is meaningful
    - Structure (0..4) of the service with regards to separation of concern (Dto, service, controller, repository, etc), usage of classes, interfaces and code separation.
        - 0 = low structure
        - 1 = some structure
        - 2 = good structure
        - 3 = very good structure
    - Testing / Mocking (0..3) of the controller and underlying services / repositories
        - 0 = no testing / mocking
        - 1 = some
        - 2 = ok
        - 3 = extensive and good application

- 15 points for code (auto-marker)

The marks will be allocated on auto-tests with the following values (*always /api/v1 in front of the endpoint*):

Endpoint	**Comment**	**Points**
/actuator/health *(this is the only endpoint which has no /api/v1 in front)*		1
uid (1)		1
distanceTo (5)	Correct call	2
	error call(s)	1
isCloseTo (5)	Correct call	2
	error call(s)	1
nextPosition (7)	Correct call	2
	error call	1
isInRegion (8)	Correct call	2
	error 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.