REST Tutorial

This tutorial covers REST-programming for server and clients.

All code mentioned is available in github and can be cloned using:

`git clone https://github.com/mglienecke/IlpTutorialRestService.git`
Run the code as a local server in docker with a port binding to 8080.

I would suggest the following order for the tutorial:

- Explain (once more) the basics of exchanging messages over TCP/IP (after all REST is only text documents)

- Explain URL structure and parts
- Explain JSON (a bit)
- Show the difference between
    - GET / POST (most commonly used)
    - PUT and DELETE

- Show the examples below using curl (or the browser for the GET calls)

- show the class how the server is working (but not the TestGet and TestPostClient!) - the main logic is in controller/CoreRestController.java

    - explain the methods

- **Explain once more what it takes to run the service in docker – from within intelliJ and command line**

- with the server running locally at http://localhost:8080 run some GET and POST from below and show in the debugger what is happening

- let the students write a simple client for GET and discuss
    - show beforehand what is needed, how, etc.
    - use Maven projects -> much easier

- let the students write a simple client for POST and discuss
    - show beforehand what is needed, how, etc.
    - use Maven projects -> much easier

- if that is not ok, show the sources and explain what goes on

- Let them then clone the server locally
- and add a new GET method and a POST method
- test with CURL (or any tool)

GET-calls (works with browser or curl)

GET HTML

```
curl -X GET localhost:8080/test
```

GET isAlive

```
curl -X GET localhost:8080/isAlive
```

GET with URL parameters

```
curl -X GET localhost:8080/testPath/AAA
```

GET JSON

```
curl -X GET localhost:8080/restaurants
```

GET file as static resource via URL

```
curl -X GET localhost:8080/demoFile1.json
```

POST with JSON data (item 3 will be ignored on the server)

```
curl -X POST -H "Content-Type: application/json" -d '{
 "item1": "JSON1",
 "item2": "JSON2",
 "item3": "AAA"
}' localhost:8080/testPostBody
```

This is transmitted over the wire (so it is just plain text with the content-body as JSON):

```
POST /testPostBody HTTP/1.1
Content-Type: application/json; charset=utf-8
Host: localhost:8080
Connection: close
User-Agent: RapidAPI/4.2.0 (Macintosh; OS X/13.5.2) GCDHTTPRequest
Content-Length: 47

{"item1":"JSON1","item2":"JSON2","item3":"AAA"}
```

This works with the hosted instance as well:

```
curl -X POST -H "Content-Type: application/json" -d '{
 "item1": "JSON1",
 "item2": "JSON2"
```

```
}' https://ilptutorial.azurewebsites.net/testPostBody
```

POST with request parameters (is quite unusual as POST mostly uses the body)

```
curl -X POST localhost:8080/testPostPath -d item1=data1 -d item2=data2
```

This is transmitted over the wire:

```
POST /testPostPath?item1=AAAA&item2=BBBB HTTP/1.1
Host: localhost:8080
Connection: close
User-Agent: RapidAPI/4.2.0 (Macintosh; OS X/13.5.2) GCDHTTPRequest
Content-Length: 0
```

Tools for REST:

- RapidAPI: https://paw.cloud/
- Postman (market leader)
- curl (command line - simple, easy)
- ...