Assignment 2: RESTful microservices architectures

Objectives:

In assignment 1 you learned how to implement a single RESTful service. As you might expect, typical real world setups include many services. Many online websites organise their backends using a microservice architecture. The basis of the architecture is to split functionality into multiple simple services each with a defined role instead of having one monolith service with all the functionality. As with any other method this approach has its pros and cons. There are many online articles about microservices. Get familiar with this approach. A starting point would be [1]. In this assignment you will learn how to organise and architect multiple services using the microservice architecture.

Background:

In this assignment you will create a user/login service to protect your URL-shortener by requiring users to login first. In a microservice architecture this requires having 2 services. A user service and a URL-shortener service. The user service is responsible for maintaining the user database with typical CRUD (**C**readting, **R**eading, **U**pdating, **D**eleting) operations. When a user logs in to the user service, the user service returns a **J**son **W**eb **T**oken (JWT) [2]. This JWT is a signed piece of information which is used between microservices for crucial communication such as to indicate that a user is logged in. To protect the URL-shortener service, the token is sent with every request (GET, PUT, DELETE, etc) from assignment 1. The updated URL-shortener will first check the token to verify that the user has logged in.

Assignment:

1. Create a new RESTful user service to create users and authenticate users. You can keep the list of users in memory i.e. no need for databases.

Path and method	Parameters	Description	Return value (HTTP code, value)
`/users - POST`	`Username, password`	`Create a new user with username and password and store it in a table.`	`200`
`/users/login - POST`	`Username, password`	`Check if username and password exists in the table and`	`200, JWT` `403, “forbidden”`

		generate a JWT or else return 403	

2. Update the URL-shortener from assignment 1 so that the user must have a valid token to access the methods.
    a. Include the JWT retrieved from the user service as an HTTP header e.g. x-access-token.
    b. Update the methods to first verify the JWT before allowing it to proceed. Only the GET method remains unprotected since it is public. The updated calls should be:

Path and method	Parameters	Return value (HTTP code, value)
/:id - GET	:id – identificator of a URL	301, value 404
/:id - PUT	:id – identificator of a URL	200 400, “error” 404, 403, “forbidden”
/:id - DELETE	:id – identificator of a URL	204 404, 403, “forbidden”
/ - GET		200, keys 403, “forbidden”
/ - POST	:url – URL to shorten	201, id 400, “error” 403, “forbidden”
/ - DELETE		204 403, “forbidden”

3. The two microservices are running separately on different ports but we know that website backends, typically, have one entry point port. How can you create one entry point for all your microservices? Describe (not implement) your approach.
4. During peak traffic times any one of the services can be easily overloaded with traffic. How would you scale up services independently of each other? Describe (not implement) your approach.
5. A microservice architecture means that many web services can be distributed over several backend servers. How would you keep track of all the services? Location, health? Describe (not implement) your approach.
6. Bonus: implement any of step 3,4,5

Notes:

- Use tools like Postman (https://www.postman.com) to easily test api calls, set headers etc.
- NodeJS express and Python flask are well equipped for microservices with many online tutorials.

Grading:

- Show a working prototype to the Lab assistant
- **Submit via Canvas a** tar file containing
  - All the needed files to deploy your service
  - Readme file to explain how to deploy and run the client
  - A short report describing your design and implementation of the URL-shortener and answers to questions 3,4,5

References:

[1] What are microservices?: https://microservices.io/
[2] Json Web Tokens (JWT): https://jwt.io/introduction/
[3] A Practical Guide for JWT Authentication using Nodejs and Express: https://medium.com/swlh/a-practical-guide-for-jwt-authentication-using-nodejs-and-express-d48369e7e6d4
[4] Introduction to microservices: https://www.nginx.com/blog/introduction-to-microservices/
[5] An introduction to OAuth 2: https://www.digitalocean.com/community/tutorials/an-introduction-to-oauth-2