Assignment 2: RESTful microservices architectures

Background:

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 functionalities. 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.

You will also be using JSON Web Tokens [2], or JWTs, in this assignment. Essentially, a JWT is a piece of information together with a signature, where the signature was produced using a private key known to only a particular entity (e.g., service) within a system. When the JWT is later presented to that same entity, it can use the signature to verify it has seen this information before and has approved of it in the past. This makes them perfect for authentication purposes, where a user can login once keep the token, which they can then use repeatedly to prove their identity. This way, the state of being logged-in is kept at the client-side, which matches well with RESTful principles.

Assignment:

In this assignment, you will extend your URL Shortener from assignment 1 with multi-user support. To do so, you will create a new microservice, the *authentication service*, that has a database of users and that can be used to login. Specifically, the service should send out JWTs when a user has successfully been authenticated.

Then, extend your URL shortener service from assignment 1 to require users to be authenticated, and associate mappings with specific users so only they can manage their mappings. Users should present the JWT to your shortener service, which in turn can use the authentication service itself to validate the token and see whether the user is actually logged in.

Just like in assignment 1, your authentication service must adhere to the following specification:

Path & method	Parameters	Description	Return value

			(HTTP code, Value)
/users - POST	:username-a unique username :password-a user's password	Create a new user with username and password and store it in a table	201 409, "duplicate"
/users - PUT	:username-a unique username :old-password:the current password of the user :new-password:the new password of the user	Update the user's password if the user presents the correct old password, or else return 403.	200 403, "forbidden"
/users/login - POST	:username-a unique username :password-a user's password	Check if username and password exist in the table and generate a JWT or else return 403	200, JWT 403, "forbidden"

For your URL shortener service itself, the specification has been updated to:

Path & method	Parameters	Return value (HTTP code, Value)
/:id - GET	:id-unique identifier of a URL	301, value 404
/:id - PUT	:id-unique identifier of a URL	200 400,"error" 404, 403,"forbidden"
/:id - DELETE	:id-unique identifier of a URL	204 404 403,"forbidden"
/ - GET		200, keys 403,"forbidden"
/ - POST	:url-URL to shorten	201, id 400,"error" 403,"forbidden"
/ - DELETE		404 403,"forbidden"

The only update to the specification is the inclusion of the paths that return 403 Forbidden. It is up to you to decide how to implement that, like before, and when to request for the JWT. Similarly, it is allowed to add additional return codes for existing paths as long as you keep it CRUD.

Although there is no explicit "full points" part for this assignment (i.e., implementing the specifications will get you a 10), do pay attention to the following points:

1. Make sure your JWTs do not contain unnecessary information; more information means more to hash and more to transmit, which costs performance.
2. Keep the secret in your authentication service secret; do not share it with the URL Shortener service!

You can still earn bonus points for anything else you implement but is not covered here. The same limitations apply as in the first assignment (i.e., you must implement the specification correctly to earn the points and you must mention/describe your bonus in your demo/report).

Implementation
Because this assignment builds on the previous one, we recommend you use the same tools as you did before. You can refer to the assignment document of the first assignment for more details.

Note: Do not use libraries that implement JWTs for you. The purpose of this assignment is to learn how JWTs work, so you are supposed to construct the final token yourself by appropriately generating the required JSON, encoding it in base 64 and signing it. You are, however, allowed to use libraries for the individual parts (with references, obviously, if required). **Using a library that does everything will cost you a few points.** Ask your TA if you are in doubt!

Grading
There are two parts to this assignment:

1. *Code* – Your implementation will account for 50% of your grade. While you have to submit the code itself to Canvas, as usual (see 'Submission'), most of the grading will be done during a **demo** that you have to show to your TA during the lab sessions. Don't worry, you don't have to create any slides or anything; instead, it is meant as a moment for you to show the TAs how you interpreted the specification, how you implemented it and that your implementation works. Specifically, you should show the following:
    a. Show to us that your authentication service works by showing all of the functions
    b. Then, show to us that your URL shortener service properly handles logging in by using JSON web tokens
        i. You should show all paths again, but you can go through them quickly
    c. Highlight what you changed to make your URL shortener multi-user (e.g., what can a user do, which functions are user-specific, etc), and also when you require authentication.
    d. Highlight what you put in the JWTs, how you create them and how you validate them.

e. **[Bonus points]** Highlight any other aspect of your design that you think is relevant / you are proud of, most notably any bonus things you did.

2. *Report* – The other 50% of your grade is decided by a short "report" that you have to write. It doesn't have to be a full scientific report, but pay attention that your language is easily understandable and suitable for a scientific environment. Specifically, include the following in your report:
    a. Describe your implementation (max 1 page). Focus on aspects relevant to the assignment: how did you implement JWTs? What do you put in them? How do you split the work between the services? You should also briefly mention how you changed your URL shortener to be multi-user, but you can continue where your previous report left off (i.e., you don't have to describe your full implementation again).
    b. Give answer to the following questions (max ½ page per question):
        i. *Your services will, by default, run on two different ports. Yet, most microservice architectures are reachable by a single port. How can you create one entry point for all your microservices? Describe your approach.*
        ii. *During peak traffic times, any one of your services can easily be overloaded with traffic. How would you scale up services independently of each other? Describe your approach.*
        iii. *A microservice architecture means that many web services can be distributed over several backend servers. How would you manage a system like this? Think of metrics you might need (e.g., health, location, …) and how you can collect them.*
        iv. If you have any technology in mind when answering any of the question i - iii, please include a (high-level) description of how your technology answers that question; just answering "I would use technology X" is not sufficient.
    c. Anything you did for the bonus part. While there is no page limit to this part, try to keep it succinct to shorten grading times.
    d. Provide a small table or description detailing which of your project members did what. What you write here won't influence your grade, but we will look at this if you encounter problems with cooperation in your group.

Note that your presentation style and language used in your report will be taken into account, although lightly (both ~9%, so ~18% in total).

1. For your presentation style, this is mostly based on whether you have made efforts to make the presentation go smoothly and whether you stay within time (10 minutes). Also note that we may deduct points if members of your groups are missing on presentation day (unless you have a good reason and let us know beforehand).
2. For the language in your report, this is mostly based on whether what you wrote is easily understandable and relevant. We might also deduct a few

points if your language is too colloquial (it is a scientific environment, after all).

Submission
Your work should be submitted in Canvas, under Assignment 2. Specifically, bundle your code and your report (as PDF) in a tarball or zipfile called *<group number>_web_service_2.tar.gz* or *<group number>_web_service_2.zip*, respectively. Concretely, your archive must contain the following:

1. Your working**(!!!)** source files
2. A *README.md* that explains how to get your code up-and-running
3. Your report, as a PDF

Note that, before you submit, you should have already joined a group in the People tab, or else we won't grade your submission. If you did, however, then only one member in your group has to submit, and Canvas will count it as a submission for everyone.

Note: Please do not just change groups after you submitted assignment 1! Canvas is actually really annoying in this, so please prevent chaos and let us know if you need to do so for whatever reason before you actually do so.

Tips

a. Neither your authentication service, nor your URL shortener service is required to connect to an external database at this point; both are still fine to keep their content in memory only (like in assignment 1).
b. You are still free to implement additional functionality and/or make additional requirements on parameters given in the body of a request, for both services – just make sure you support *at least* the specification.
c. Good idea for bonus points: implement your answers to the questions in the report :)

Important Notes:

- TAs, and the rest of the team, are happy to help you! However, they will only do so **during working hours**.
- TAs have extensive experience in Python, but they can only provide **best-effort support** if you decide to use another language than Python, or another library than listed here.
- Using code written by someone else is not a sin; however, **you must provide a source in your README.md / comments!!!** You are also expected to provide a detailled explanation of how the copied code works, to show that you understand it. **Failure to do so will be considered plagiarism!**
    - Don't hesitate to ask a TA for advice if you are unsure about this or a particular piece of code.

- Code written by ChatGPT (or similar AI assistants) does not count as your own work. And even if you provide a source, you will not get a passing grade using these tools.

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