Assignment 2 - Group 18

Anusha Ali, Priyanka Roy, Jeewon Heo

1. Implementation

We have created three micro services — JWT, user management, and URL shortener. To keep track of the users and URLs, we have two tables (users and urls) in our sqlite3 database.

1.1 JWT Microservice (port 5003)

We implemented JWT with the help of PyJWT library. We created a microservice named JWT authentication which is responsible for generating and validating JWT token. This service runs on port 5003. For generating JWT token, we have created an API named **/generateToken**. This API uses username and password of the user and the HS256 algorithm to create and return the unique hash. This token is valid for 30 minutes.

For validation of token, we have created **/verifyToken API** which first checks for the presence of an access token in the request headers. If the token is present, it is decoded using the app's secret key and the HS256 algorithm. If the decoding is successful, the function extracts the username from the decoded token and uses it to retrieve the corresponding user from the database, which is returned along with status code 200. If the token is missing or invalid, it returns 403 error code.

1.2 User Management Microservice (port 5000)

Path	Method	Parameters	Return	Description
/users	POST	Username and password	201 409 “duplicate”	Create new user if username is not duplicate
/users	PUT	Username, password, and new password	200 403 “forbidden”	Update user password if the old password is correct.
/users	GET	Username and password	200	Welcome page for a user
/users/login	POST	Username and password	200 JWT 403 “forbidden”	Generates a JWT token if the detail is correct

User sign up

A post request to endpoint /users with JSON body with username and password will create a new user account. If JSON body is empty or username and/or password is missing, it will return 400 error. If the given username exists in the database, it will return 409 duplicate status. Otherwise, it will successfully store in the users table, and return 201 code. Before being stored in the database, the password is hashed using werkzeug’s generate_password_hash function.

User login

A post request to endpoint /users/login with JSON body with username and password will have the user login to the service. If JSON body is empty or username and/or password is missing, it will return 400 error. The given login details exists in the database and the passwords matched according to werkzeug’s check_password_hash function, the user can successfully login. Then, we generate a JWT token along with 200 code. Otherwise, login fails and we return 403 forbidden code.

User password update

A put request to endpoint /users with JSON body with username, password, and new password will update the user’s password. If the JSON body is empty or not all required fields are present, it will return 400 error. If JWT token in the request header is missing or not validated by the JWT microservice, then it returns 403 forbidden code. In case the login details (username and old password) does not match with ones in the database or not matched by check_password_hash function, we also return 403 code. Otherwise, the user’s password is updated, and we return code 200.

User welcome page

A get request to endpoint /users will return a username, hashed password, and check result by check_password_hash. This is to have a quick check over the status of users during development.

1.3 URL Shortener Microservice (port 5001)

The functionalities of the URL shortener service remain the same. For all API requests to this service, there should be a valid JWT token, otherwise we return 403 forbidden code. Also, a user can only have access to the URLs created by them. They are not allowed to view, delete, or change information created by different users. This was easily implemented by extending urls table with username, and only handling the rows with matching username as the current user.

Question 1

One of the most common approach to provide single entry point to the users is API gateway. It is a frontend for the users to access multiple microservices and sits between the users and services. API gateway is responsible for request routing, policy enforcement, and composition. Also API gateway reduces the client side and microservices implementations. Based on these benefits, we would try API gateway to support single entry point for the users. In Kubernetes system, we can introduce API gateway as a load balancer, ingress controller, or inside the service mesh.

Question 2

Service overloading can happen easily when there are many users that wants to use the services at the same time. One way to avoid this is to employ a load balancer that distributes the traffic across multiple instances. Also, in order to scale the microservices, we can try horizontal scaling. With Kubernetes, this can be easily implemented by deploying multiple instances of a microservices. Also, many cloud services such as GCP provide automatic scaling functions which scales the number of microservices according to the traffic size.

Question 3

One way to check the health of the microservices can be done by monitoring response time and error rates. If a service shows a significant increase in either of those metric could signify a bad health. Locations of microservices is necessary to ensure workload is well distributed across the servers. If there is any geographical imbalance, then we can consider adjusting the server location of the services. Also, it is also beneficial to track the resource utilization of the microservices. If there is any service that uses significant amount of resources, we can address it allocate more resources for it. Monitoring tools such as Grafana and Prometheus can help with this.

Work Division

	Anusha	Priyanka	Jeewon
Code	X	X	
Demo/Testing	X	X	X
Report	X		X
Article summary	X		