UNIVERSITY OF AMSTERDAM

Web Services and Cloud-Based Systems Assignment-2 *RESTful Micro service Architecture*

Web Services and Cloud-Based Systems 2026

Group: 05

Member names: (Ali Akbar, Jaden Aguiar, Muhammad Akbar Rahman)

Emails: (ali.akbar@student.uva.nl, jaden.aguiar@student.uva.nl, muhammad.akbar.muhammad.akbar.rahman@student.uva.nl)

UvA-IDs: (16082095, 16517059, 165115072)

Master program and Track: Computer Science - BDE

February 20, 2026

Contents

1 Introduction

In this assignment, we have implemented a simple authentication service that manages user credentials and facilitates user login. We are using JSON Web Tokens (JWT) for the secure exchange of authentication information between different services. Only authenticated users are able to use the protected resources in the system.

1.1 Overview of Approach

The overall approach is an extension of the URL-shortening service (implemented in assignment 1) by adding an authentication service which is responsible for managing users. The authentication service is a basic validation service that contains users and their credentials. When a user logs in, a JSON Web token is generated which contains the username, expiration time, signed with a shared secret and the HS256 algorithm. The token is then returned to the client as a Bear token and is later used by other services to verify the user.

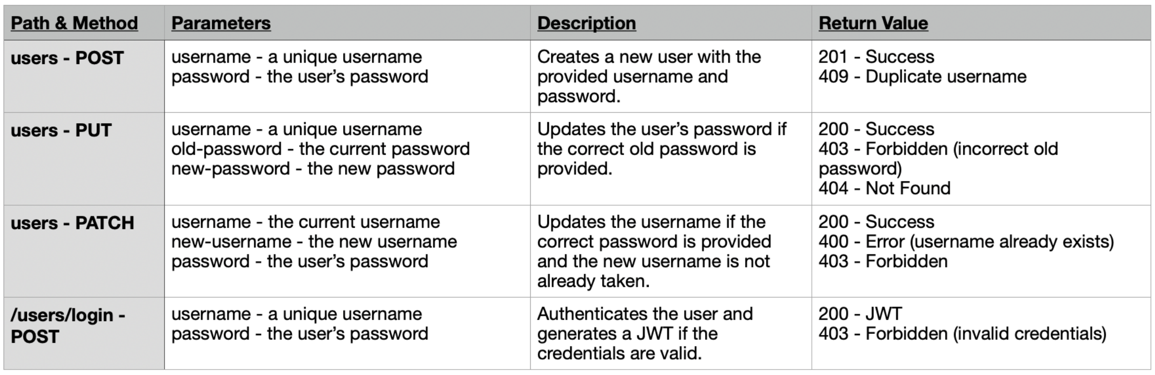
2 Implementation

The authentication service provides a basic function of user management and authentication. All users can directly work with the authentication service to create an account, update credentials, and log in to the authentication service. After the user successfully logs in, the authentication service generates a JWT for that user, which will be used to authenticate and authorize requests made to the URL shortener service. The JWT will be passed in the Authorization header on requests sent to the URL shortener service to ensure secure communication between users and the two services.

2.1 Authentication Service Specification:

The **User Authentication Services** provides an authentication API that allows for managing users via a RESTful architecture. The API allows for **creating users (POST), changing passwords (PUT), and changing usernames (PATCH).** The User Authentication Services also allows for authenticating users and **generating a JSON Web Token (JWT)** for making secure service-to-service calls between services through the '**/users/login**' endpoint.

The table below describes the endpoints, parameters, and behavior of the authentication service.



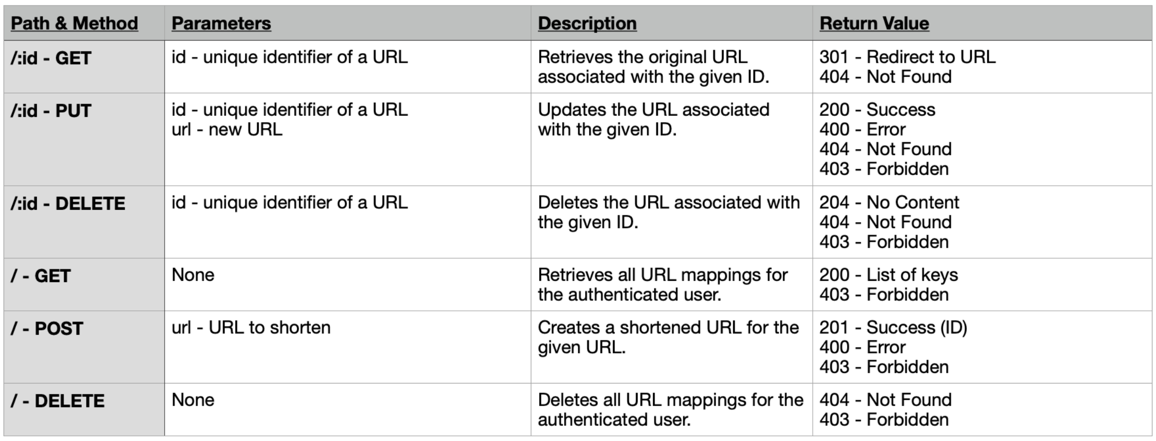
Path & Method	Parameters	Description	Return Value
users - POST	username - a unique username password - the user's password	Creates a new user with the provided username and password.	201 - Success 409 - Duplicate username
users - PUT	username - a unique username old-password - the current password new-password - the new password	Updates the user's password if the correct old password is provided.	200 - Success 403 - Forbidden (incorrect old password) 404 - Not Found
users - PATCH	username - the current username new-username - the new username password - the user's password	Updates the username if the correct password is provided and the new username is not already taken.	200 - Success 400 - Error (username already exists) 403 - Forbidden
/users/login - POST	username - a unique username password - the user's password	Authenticates the user and generates a JWT if the credentials are valid.	200 - JWT 403 - Forbidden (invalid credentials)

Figure 1: Table Of Authentication Service API Specification

2.2 URL Shortener Service Specification:

We have added a **require_jwt** decorator to ensure all endpoints are accessible only to authenticated users. The Shortened URLs are now restricted to the **authenticated user (request.user)** for proper data separation of multi-users and the service returns 403 Forbidden responses when users attempt to access or modify resources which they are not authorized.

The table below describes the endpoints, parameters, and behavior of the URL shortener service.



Path & Method	Parameters	Description	Return Value
/:id - GET	id - unique identifier of a URL	Retrieves the original URL associated with the given ID.	301 - Redirect to URL 404 - Not Found
/:id - PUT	id - unique identifier of a URL url - new URL	Updates the URL associated with the given ID.	200 - Success 400 - Error 404 - Not Found 403 - Forbidden
/:id - DELETE	id - unique identifier of a URL	Deletes the URL associated with the given ID.	204 - No Content 404 - Not Found 403 - Forbidden
/ - GET	None	Retrieves all URL mappings for the authenticated user.	200 - List of keys 403 - Forbidden
/ - POST	url - URL to shorten	Creates a shortened URL for the given URL.	201 - Success (ID) 400 - Error 403 - Forbidden
/ - DELETE	None	Deletes all URL mappings for the authenticated user.	404 - Not Found 403 - Forbidden

Figure 2: Table Of URL Shortener Service API Specification

2.3 Communication Between Services:

The authentication service demonstrates how users are registered and logged in. Once a user logs in successfully, the authentication service creates a JWT for the user. The user sends a JWT in the Authorization header as part of their request to the URL shortening service. The URL shortening service validates the JWT to verify the user's identity. This validation involves decoding the JWT using the shared secret and verifying its signature and claims. Once validated, the sub (subject) claim from the JWT is extracted and used as the user ID to identify the URLs associated with the authenticated user. Only after successful validation, the service process the request and return the appropriate response. Hence, authentication and URL linking remain two separate processes but securely linked via JWT based communications between an authenticated user and the URL shortener service.

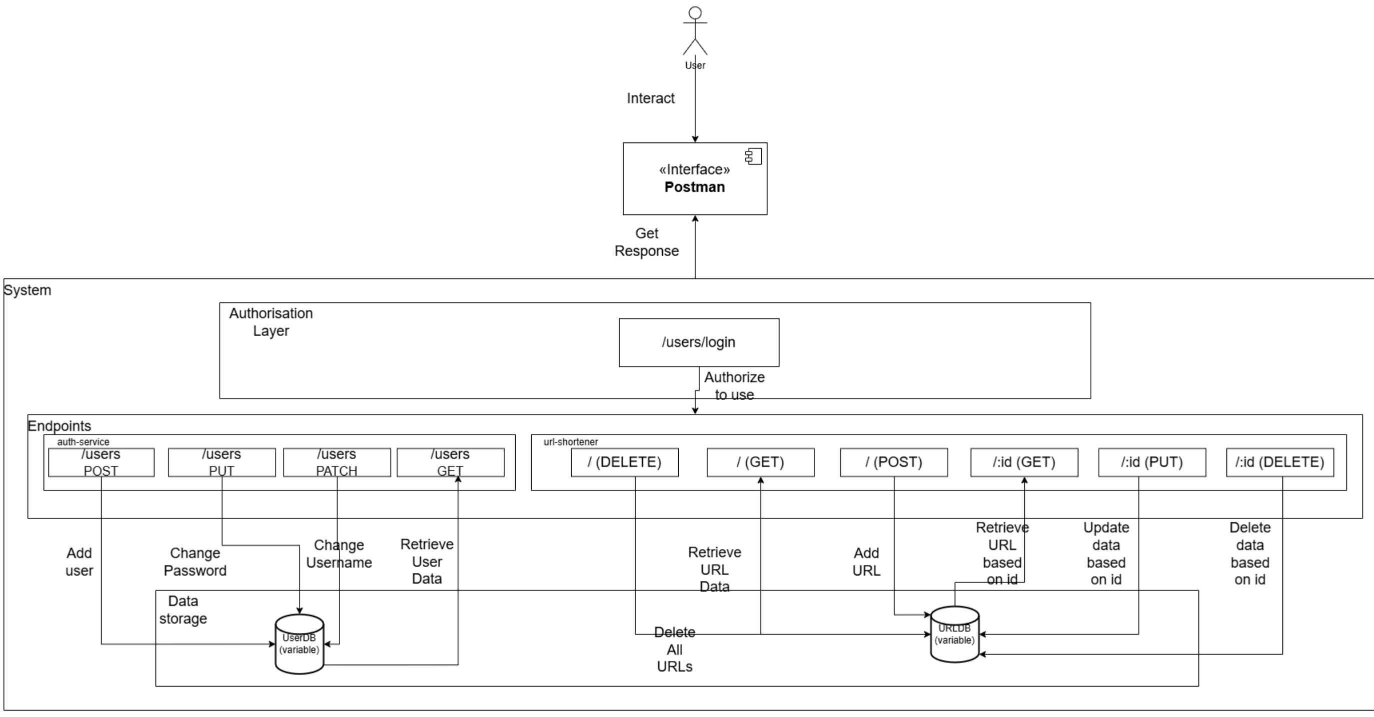
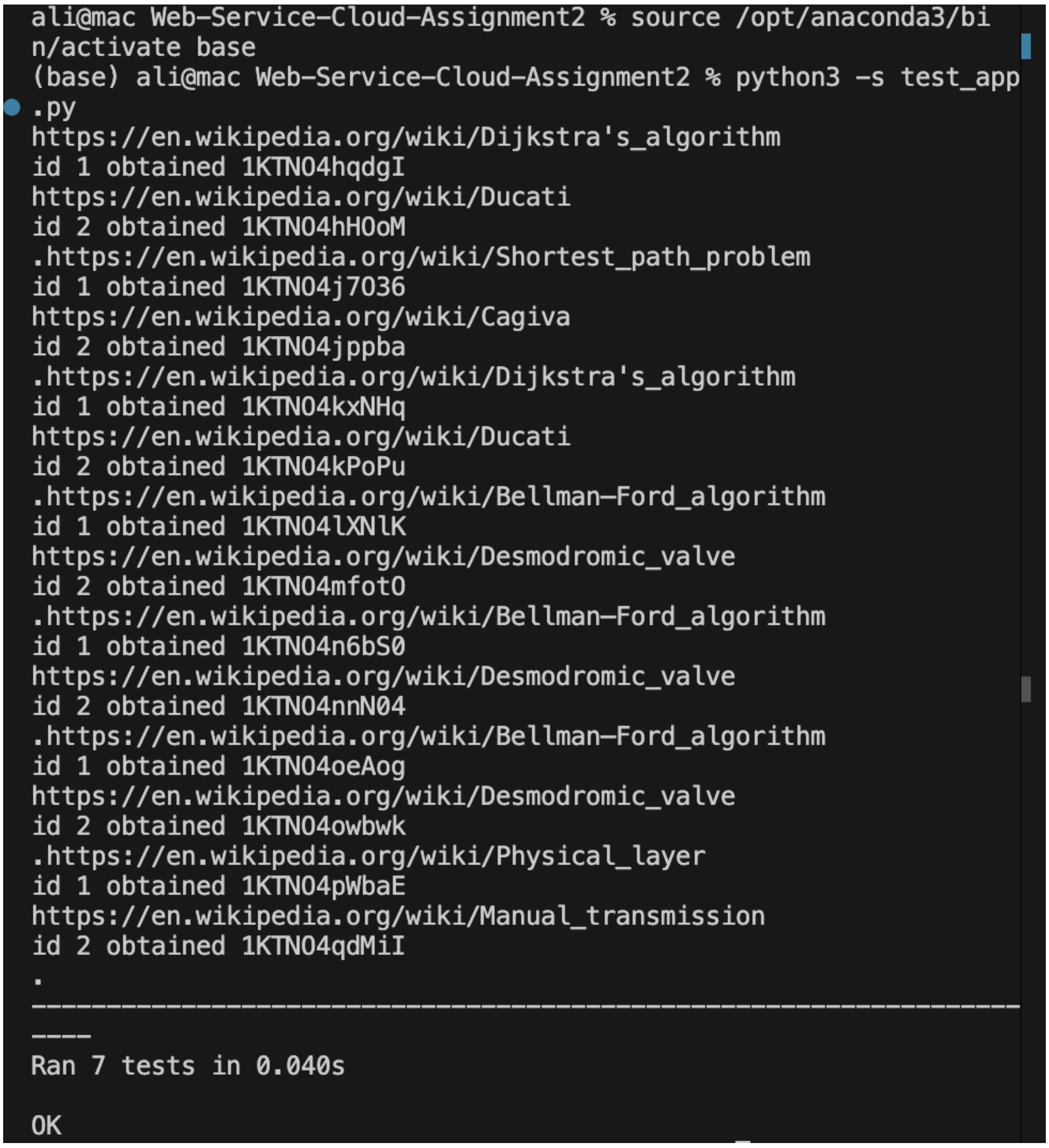


Figure 3: Service Interaction Diagram

2.4 Test Cases:

The tests conducted on the URL Shortener and Authentication services confirmed that all endpoints are functioning as expected. The actions tested were creating, retrieving, updating, and deleting shortened URLs for authenticated individual user which was achieved by generating JWT token based authentication as explained above. The end-to-end testing verified that requests were processed according to their validity, complying with the defined HTTP status codes and behavior.



```
ali@mac Web-Service-Cloud-Assignment2 % source /opt/anaconda3/bi
n/activate base
(base) ali@mac Web-Service-Cloud-Assignment2 % python3 -s test_app
.py
https://en.wikipedia.org/wiki/Dijkstra's_algorithm
id 1 obtained 1KTN04hqdgI
https://en.wikipedia.org/wiki/Ducati
id 2 obtained 1KTN04hHOoM
.https://en.wikipedia.org/wiki/Shortest_path_problem
id 1 obtained 1KTN04j7036
https://en.wikipedia.org/wiki/Cagiva
id 2 obtained 1KTN04jppba
.https://en.wikipedia.org/wiki/Dijkstra's_algorithm
id 1 obtained 1KTN04kxNHq
https://en.wikipedia.org/wiki/Ducati
id 2 obtained 1KTN04kPoPu
.https://en.wikipedia.org/wiki/Bellman–Ford_algorithm
id 1 obtained 1KTN04lXNlK
https://en.wikipedia.org/wiki/Desmodromic_valve
id 2 obtained 1KTN04mfotO
.https://en.wikipedia.org/wiki/Bellman–Ford_algorithm
id 1 obtained 1KTN04n6bS0
https://en.wikipedia.org/wiki/Desmodromic_valve
id 2 obtained 1KTN04nnN04
.https://en.wikipedia.org/wiki/Bellman–Ford_algorithm
id 1 obtained 1KTN04oeAog
https://en.wikipedia.org/wiki/Desmodromic_valve
id 2 obtained 1KTN04owbwk
.https://en.wikipedia.org/wiki/Physical_layer
id 1 obtained 1KTN04pWbaE
https://en.wikipedia.org/wiki/Manual_transmission
id 2 obtained 1KTN04qdMiI
.
------------------------------------------------------------------
----
Ran 7 tests in 0.040s

OK
```

Figure 4: Snapshot Of Test Cases

3 Future Scope

Currently, user data is stored in an in-memory dictionary, which is lost whenever the server is shutdown. Therefore, to make it persistent and reliable, we can store user's credentials in a database (PostgreSQL, MySQL, or MongoDB). The database would also provide additional features such as password recovery, account locking after multiple failed login attempts, and access controls based on the user's role (Admin vs Normal user).

Other improvements and enhancements can be:

- **Creating Single Entry Point for All Microservices:** An API Gateway can be used which will act as a reverse proxy and route incoming client requests to the appropriate microservice based on request path or other criteria. As a result, clients will only interact with a single port, while the API Gateway is routing their request to the appropriate microservice.
- **Scaling Services Independently:** We can use tools such as **Docker** and **Kubernetes** to achieve independent scaling of our services by using containerization and orchestration. A microservice can be built and run in an isolated Docker container with all of its dependencies, allowing it to be scaled independently. In addition to providing the means for deploying multiple copies of a service using the horizontal scaling provided by Kubernetes, it also includes an integrated load balancing mechanism to distribute traffic among the multiple copies, and it utilizes the **Horizontal Pod Autoscaler** feature to automatically adjust the number of service instances (pods) running based on metrics such as CPU utilization or request load.
- **Managing a Distributed Microservice Architecture:** It requires focus on monitoring, centralized logging, and automated health checks. Some standard metrics to track will be: **service availability, latency, error rate, cpu/memory usage, throughput, and network traffic**. Tools such as **Prometheus** and **Grafana** can be used to collect and visualize these metrics, while **Elasticsearch**, **Logstash**, and **Kibana** would be suitable for log management. Also, health check endpoints and **Kubernetes probes** provide the capability for automatic detection and recovery of unhealthy/malfunctioning services.

4 Anything done for bonus?

As a bonus, we have implemented the PATCH method to allow users to update their usernames. Also, we enhanced security by encrypting passwords using SHA-256 hashing, preventing exposure of plaintext password in the application.

5 Contribution

Group Member Name	Contribution
Ali Akbar	He has implemented the code for JWT authentication and has written the report.
Muhammad Akbar Rahman	He has implemented the URL endpoints, the bonus part, and refactored code based on the test cases.
Jaden Aguiar	He has drawn service interaction diagram and has written the README file.