Assignment 2 - Group 28

UvA - Web Services and Cloud-Based Systems

Tianhao Xu
14129027

Summer Xia
14094584

Yiming Xu
13284657

1 DESIGN

We constructed and implemented a RESTful API for creating and authenticating users in this version of the project based on assignment 1. Its main functions include user creation and permission management, as well as the generation of authentication tokens via JWT which identifies the login status of users. The URL-shortener service is likewise linked to the user service. Before utilizing the shortener, it will verify the token to ensure that the user has signed in.

User Creation and Permission Management: A new user can set up an account by entering his/her username and password. We also separate administrator permissions, that administrators may utilize to control all users, such as viewing all user profiles, designating normal users as administrators, removing records from URL-shorteners, and deleting users, etc.

User Valid Tokens Generation: A Json Web Token (JWT) will be generated and returned when a user signs in to the user service. This JWT is an authentication credential that is exchanged between microservices for critical communication, such as indicating whether or not a user is logged in. The users need to bring this token to use the shortener.

1.1 Design UML

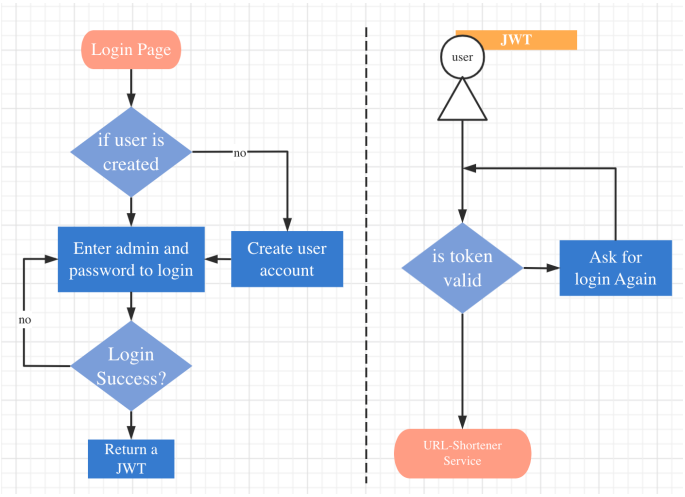

Figure 1: User Login and Token Authentication

2 IMPLEMENTATION

2.1 User Service

User Model The user data is stored in an SQLite table called "User" with attributes: id, user_id, name, password, admin.

Regular User This "create user" function is based on the *POST* method, which accepts json-formatted information, including the user's *name* and *password*. After that, it generates user_id with *uuid* and encrypts password to hashed-pwd with *werkzeug.security*. At the same time, we set the *admin* parameter to indicate if the user is an administrator, which is a bool type and defaulted to *False*.

The user must log in by *name* and *password* before accessing the URL-shortener service. In the shortener service, the short URL links created by each user can only be viewed, accessed or edited by the user himself, that is, the records of each user are not shared. We achieve this function by adding the user_id attributes to each URL link record. (The updated Link model is of 6 attributes: *id, originial_url, short_url, visits, date_created, user_id*)

Administrator Permissions The user has administrator capabilities if the admin parameter is set to *True*. The administrator has the ability to use the *GET* method to obtain information about all users in the database; by using the *PUT* method, the administrator can set the admin of a common user to *True* to promote he/she to administrator; and the administrator can use the *DELETE* method to remove user information from the database.

We also set a function for the administrator user to delete the original and short URL information of a user or all users. It takes a user's id and short URL, searches the database for associated information, and then deletes it.

2.2 User Valid Tokens Generation

To generate tokens for users, we utilize JWT (JSON Web Token), with x-access-token as the header. Our JWT is generated by user_id, timestamp, and SECRET KEY, using HS256 as the signature algorithm, and then encodes the corresponding returns into the user token.

Before the user calls the services in URL-shortener, we need to get the user's token, decode the token to get the user_id, and look up the information of user in the database. This token contributes to the shortener service in determining whether or not the user has logged in and whether he is permitted to utilize the service he has requested.

3 ANSWER TO QUESTION 3

Both of our user service and shortener service are running on one port. To accomplish this function, we use the Blueprint class in flask which is a collection of routes and other app-related functions allowing to define application functions without requiring an application object ahead of time. We have written all the route functions in one file. When running the flask app, it will automatically register this blueprint. Therefore, both user services and shortener services can be implemented on one port.

4 ANSWER TO QUESTION 4

We considered two approaches to resolving the issue of service overload during peak traffic times.

The first option is, when resource expansion allows such as when performing container instantiation, to expand available CPU resources to accelerate the processing capacity of the service. While the traffic of external requests rises, dynamically scale the microservice's configuration, like enlarging the capacity of cache and the number of CPU cores.

The second method is to add a routing node before the service. All of the requests are received by the routing node, and then distributed to associated microservices. When traffic pressure is low, a single service can address the requests; when traffic pressure is great, many services can be created to process requests at the same time and return the results to the router. Through this architecture, it can achieve the effect of decentralizing the amount of calculation, leading to a advantage that different parts perform their own duties and cooperate with each other to handle the overload situation.

5 ANSWER TO QUESTION 5

(1) We can track the location or health status of web services distributed across multiple back-end servers through a service mesh. The current status is recorded in a registry [2]. Their respective location or health status is obtained by writing a

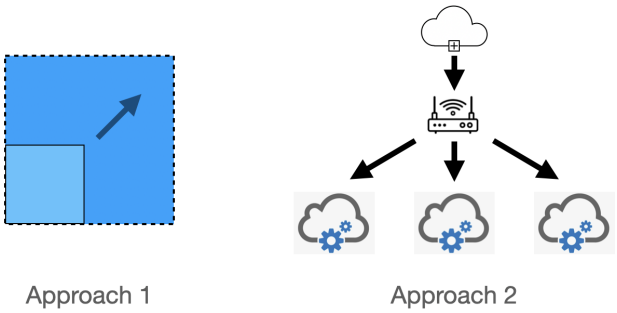

Figure 2: Approaches to Q4

separate service. The service will obtain service and service instance data from a third-party service registry and convert it into resources to be written to the server via the K8s API server interface. A controller is used to listen for changes to the K8s API Server resource object and convert it to an internal service model to enable the tracking of information.

(2) Distributed link tracing methods and techniques can be used to track the location and health of web services. This is because the help of logs and interactable parameters can give us traceable information. Based on OpenTracing's distributed tracing standard [1], we can build on the open source work. Connecting different spans to form a trace. In this traceable record, the health of a moment is monitored and its location is recorded in a data-interaction format. Each span will mark the start and end time nodes, each consisting of a link that maintains a unique identifier until it is returned to the requesting party.

6 APPENDIX

Run methods are listed in detail in *README.md* file.

We drew inspiration for this project from a YouTube instructions[3].

REFERENCES

[1] Jonas Höglund. An analysis of a distributed tracing systems effect on performance jaeger and opentracing api, 2020.

[2] Wubin Li, Yves Lemieux, Jing Gao, Zhuofeng Zhao, and Yanbo Han. Service mesh: Challenges, state of the art, and future research opportunities. In *2019 IEEE International Conference on Service-Oriented System Engineering (SOSE)*, pages 122–1225. IEEE, 2019.

[3] Pretty Printed. "Building a URL Shortener in Flask". [Online], 2019. Available: https://youtu.be/rGQKHpjMn_M. [Accessed JUN.12, 2019].