Web Service and Cloud-Based Systems 2023: Assignment 1

Yihong Xi(14720035), Xingyou Li(14741628), Zhiheng Yang(14483262)

1 Code Implementation

This section describes and motivates the implementation code of the assignment. The complete implementation code can be found in the <app.py> file submitted together with this report.

1.1 Basic Functions and Database Structure

As suggested in the assignment description, the code to shorten URLs is implemented in Python using Flask, a popular web framework. It contains two routes: one is the root route ("/"), and the other is a dynamic route with the parameter "key" or "unique identifier" ("/<key>"). The root route handles GET, POST, and DELETE requests, while the dynamic route with a parameter handles GET, PUT, and DELETE requests.

Firstly, the code uses the Flask SQLAlchemy library, a popular ORM (Object-Relational Mapper) tool, to encapsulate SQL operations. This saves time spent writing SQL sentences to maintain and update the database.

Secondly, the Url class defines the structure of our database, which contains an ID (primary key and auto-increment), original URL, short URL, and user. The "short URL" column represents the unique identifier of each original URL, while the "user" column is used to control authorization (only users whose names are in the corresponding row and column can update or delete the URLs).

Thirdly, as shown in Table 1, the root route handles three HTTP methods (GET, POST, and DELETE): a GET request returns a JSON object including all short URLs (unique identifiers). A POST request returns a unique identifier if the request parameter is a valid URL; otherwise, it returns HTTP code 400 and an error message. Finally, a DELETE request returns HTTP code 404.

Lastly, the dynamic route handles three HTTP methods (GET, PUT, and DELETE) with the parameter "key." For a GET request, if the parameter is correct, the application redirects to the original URL with HTTP code 301. If the parameter is incorrect, it returns HTTP code 404. For a PUT request, an identifier (HTTP code 200) is returned when the condition is a valid identifier, a valid new URL, and authorization is approved. If the authorization is not approved, it returns HTTP code 403. HTTP code 400 and 404 are returned, respectively, when the identifier is correct with an invalid new URL and the invalid identifier. For a DELETE request, if the parameter is correct and authorization is approved, the application returns HTTP code 204. If the parameter is incorrect, it returns HTTP code 404. If the parameter is correct but authorization is not approved, the application returns HTTP code 403.

Overall, the program provides a URL shortening service with which users can validate, create, update, and delete short URLs.

1.2 Identifier Generation Algorithm

To implement the ID generation algorithm, we first add the original long URL to the database and obtain a unique index from the database. Then, we use the index to generate a short ID using the Hashids [1] library in Python. Hashids allow us to convert integers to hex, which means we can convert a short ID using the unique index in the database. Since the index is unique and corresponds to a long URL, we can ensure that there are no collisions in ID generation.

It is worth noting that there are two main contrains for the input of the Hashids in out algorithm. First, the input of the Hashids must be an integer. Second, the input of the Hashids cannot be a negative number. This is not a problem for our algorithm because we use the unique index in the database as the input of the Hashids. The unique index is always a positive integer.

Path & method	Parameters	Return value(HTTP code, Value)
/:id - GET	valid identifier	301, redirect to original url
	invalid identifier	404, error
/:id - PUT	valid identifier and valid new url (authorization approved)	200
	valid identifier and invalid new url	400, error
	invalid identifier	404, error
	valid identifier and new url (authorization not approved)	403, Authorization Forbidden
/:id - DELETE	valid identifier (authorization approved)	204
	invalid identifier	404, error
	valid identifier(authorization not approved)	403, Authorization Forbidden
/ - GET		200, all keys
/ - POST	valid url	201, short url(identifier)
	invalid url/ no url	400, invalid url or url not exist
/ - DELETE		404, error

Table 1: Response Table

To validate the urls, we used the regex library in Python. The regex library allows us to validate the URL format. We also used the requests library to check if the URL is valid. The requests library allows us to send HTTP requests to the URL and check if the response is valid. If the URL is valid, we add the URL to the database and create a short ID for it.

2 Multi-user Design

The application has a feature where each data (URL) is assigned an owner, and only the owner can perform PUT or DELETE operations on that URL. If someone, not the owner, tries to perform these operations, the application will return a "Authorization Forbidden" response with HTTP code 403. This mechanism ensures that only the authorized user has control over their own data and enhances the application's overall security. With that, multiple users can create or remove URL mappings owned by themselves, avoiding the situation that somebody's mappings are updated or deleted by others accidently.

3 Bonus Parts

3.1 SQLite

SQLite [2] is lightweight database and suitable for small to medium-scale applications, which can handle the increased load up to a certain point. Using SQLite over in-memory storage brings persistence, which could keep the data available when server is stopped or restarted.

So in the context of the URL shortener application, SQLite is used to store the original URLs and their corresponding shortened version of each user, instead of memory.

3.2 Regular Check

The regular expression `^https?:/{2}\w.+$` can be used to match and extract valid URLs from a given text. The expression starts with an optional "https://" or "http://" protocol, followed by two forward slashes, and then any number of word characters (letters, digits, or underscore) until the end of the line. This regular expression ensures that the URL is properly formatted.

3.3 URL Redirect

The redirect of a valid URL is a critical aspect of a URL shortener service. When a user requests a shortened URL, the service must return the original URL. On this basis we have implemented a direct redirect to the corresponding website.

This method ensures that if a valid id is entered into the browser input box, it will automatically redirect to the website. The URL redirector ensures that the shortened URL serves as a useful and functional substitute for the original URL.

3.4 Response Speed vs Storage Cost

A user may upload the same URL multiple times in the URL shortening service. However, we finally decided not to deduplicate the URLs in our system. The main reason is that performing such a check would require additional queries to the database, which may cause a delay in the response time. Since our storage system is based on SQLite, disk space is relatively inexpensive. Therefore, the tradeoff of storing duplicate URLs for faster response times is more reasonable.

3.5 Unit Test Module

The library unittest ensures that the app's various endpoints are functioning as expected. The tests cover a range of scenarios, such as invalid URLs, invalid user IDs, and different authorization levels, and include both positive and negative assertions. Thus, these tests work as a valuable and essential tool for functionalty assurance, but also a bonus for this project.

4 Conclusion

In conclusion, we have successfully developed a URL shortener RESTful service using Flask and SQLite. We have implemented all the basic requests required in the assignment description. Moreover, we implemented an unique identifier generation algorithm, URL validation and authorization for users' requests of PUT and DELETE. As a result, our implementation is believed to be efficient and reliable.

5 Contribution Table

Members	work
Yihong Xi	1. database design 2. building basic project 3. implement all basic requests
Xingyou Li	User Authentication
Zhiheng yang	Test Unit

Table 2: Contribution Table

References

[1] B. Brzóska. Hashids. `https://hashids.org/`.

[2] Sqlite Home Page. `https://www.sqlite.org/index.html`, mar 22 2023.