Project Report(URL Shortener)

1. Understanding the Task:

Before starting the project, I carefully read the problem statement to understand what was expected. The main goal was to create a basic URL Shortener web application. The application should take a long URL from the user, generate a short version of it, and allow the user to use this short link to open the original website. Another important requirement was to store the URLs so that users can view them later on a history page. Since this was an internship task, I decided to keep the application simple and easy to understand, focusing more on functionality than design.

2. Planning the Solution:

After understanding the task, I planned the overall flow of the application. I decided that the application should have two main pages. The first page would allow users to enter a URL and generate a short URL. The second page would show all previously shortened URLs. I also planned that the backend should handle URL validation, URL shortening logic, database storage, and redirection. Planning the steps in advance helped me avoid confusion while coding.

3. Selection of Technologies:

I selected technologies that are beginner-friendly and suitable for small projects. HTML was used for creating the user interface because it is simple and sufficient for forms and tables. Flask was chosen as the backend framework because it is lightweight and easy to learn. SQLite was used as the database since it does not require any complex setup. SQLAlchemy ORM was used to interact with the database easily using Python code. These tools helped me build the application without unnecessary complexity.

4. Database Design Approach:

To store the URLs, I created a simple database table. The table contains an id to uniquely identify each record, the original URL entered by the user, and the generated short URL. I used SQLAlchemy ORM to define this table as a Python class. This approach made database operations such as saving and retrieving data straightforward. The database is automatically created when the application runs for the first time, which made the setup easier.

5. Implementing URL Shortening Logic:

For shortening URLs, I created a function that generates a random combination of letters and numbers. This random string acts as the short URL code. I kept the length of the short code small so that the final URL remains short and easy to share. This method was chosen because it is simple and effective for a basic URL shortener application.

6. Handling User Input and Requests:

The Flask routes handle all user interactions. When the user submits a URL on the home page, the backend first checks whether the URL is valid. If the URL is valid, a short URL is generated and saved in the database. The shortened URL is then displayed to the user. When a user opens a shortened URL, the application searches the database for the matching short code and redirects the user to the original URL. If the short code is not found, an error message is displayed.

7. URL Validation:

To make sure that incorrect URLs are not processed, I used a Python validation library. This library checks whether the entered text follows a valid URL format. If the URL is invalid, the application shows an error message and does not save the URL. This step improves the reliability of the application.

8. Frontend Development:

The frontend of the application was built using plain HTML. The home page contains a text input field for entering the URL, a button to shorten the URL, and a copy button to copy the shortened link. The history page displays all stored URLs in a table format. The frontend was kept simple to make the application easy to use and understand.

9. Testing the Application:

After completing the implementation, I tested the application by entering different URLs. I checked whether valid URLs were shortened correctly and invalid URLs were rejected. I also tested the history page to ensure that all stored URLs were displayed properly. Testing helped confirm that the application worked as expected.

10. Challenges Faced:

One challenge was making sure that the URLs were properly stored and displayed in the history page. This was resolved by verifying database commits and database queries. Another challenge was validating user input, which was solved using a Python validation library.

11. Final Outcome:

The final application successfully shortens URLs, stores them in a database, redirects users to original URLs, and displays URL history. The project meets all the requirements given in the task and remains simple and beginner-friendly.

12. Conclusion:

This project helped me understand the basics of web application development using Flask and databases. The step-by-step approach made it easier to implement the application correctly. This project provided valuable hands-on experience and improved my confidence in backend development.