URL SHORTENER WEB APPLICATION FOR ADVANCE USERS

Project Report

Intern Name: Nikhil Somnath Borade

Internship Program: Data Science with GenAI

Organization: Innomatics Research Labs

Technology Stack:

- **Frontend:** HTML, CSS, Bootstrap
- **Backend:** Flask (Python)
- **Authentication:** Flask-Login
- **ORM:** SQLAlchemy
- **Database:** SQLite

---

1. Introduction

In the modern digital world, sharing web links is a daily activity. However, long URLs are often inconvenient to share, difficult to remember, and prone to copying errors. A URL Shortener application provides a solution by converting long URLs into short, easy-to-share links while maintaining correct redirection.

The **URL Shortener Web Application for Advance Users** was developed as part of my **Data Science with GenAI internship**. This advanced version of the project extends the basic URL shortener by introducing **user authentication**, allowing each user to log in, shorten URLs securely, and maintain a personal history of their shortened links.

This project focuses on real-world web application development concepts such as authentication, database relationships, ORM usage, and secure handling of user data.

---

2. Project Objectives

The main objectives of this project are:

1. To develop a URL shortener web application with user authentication.
2. To allow users to sign up with a unique username and password.
3. To validate username constraints during signup.
4. To allow registered users to log in securely.
5. To enable logged-in users to shorten URLs.
6. To store URLs in a database associated with individual users.
7. To display user-specific URL history.
8. To implement URL validation to prevent incorrect data storage.
9. To ensure secure password handling.
10. To design a clean and responsive frontend interface.

---

3. Why an Advanced URL Shortener Is Needed

While a basic URL shortener is useful, an advanced version provides additional benefits:

- Personalized URL history for each user
- Improved security through authentication
- Better data organization
- Scalability for future enhancements
- Real-world applicability

This project demonstrates how authentication and database relationships can be integrated into a web application to make it more secure and user-friendly.

---

4. System Architecture

The application follows a **three-tier architecture**:

4.1 Frontend Layer

The frontend is built using:

- HTML for structure
- CSS for styling
- Bootstrap for responsive and consistent UI design

The frontend consists of three main pages:

- **Login Page**
- **Signup Page**
- **Dashboard Page**

4.2 Backend Layer

The backend is implemented using **Flask**, which handles:

- Routing
- Authentication logic
- URL shortening logic
- URL redirection
- Database communication

Flask was chosen for its simplicity, flexibility, and suitability for learning full-stack development.

4.3 Database Layer

The database layer uses:

- **SQLite** as the database
- **SQLAlchemy ORM** for database operations

ORM allows interaction with the database using Python objects instead of raw SQL, improving maintainability and security.

---

5. Frontend Description

5.1 Login Page

The login page allows registered users to enter their username and password. If the credentials are valid, the user is redirected to the dashboard. If invalid, an error message is displayed.

5.2 Signup Page

The signup page allows new users to register. The following validations are applied:

- Username must be unique
- Username length must be between **5 and 9 characters**
- An error message is displayed if the username already exists or is invalid

After successful signup, the user is redirected to the login page.

5.3 Dashboard Page

The dashboard is accessible only after login and provides:

- URL input field
- Shortened URL output
- List of previously shortened URLs by the logged-in user
- Logout functionality

---

6. Project Workflow

The workflow of the application is as follows:

1. User visits the application.
2. User signs up with a valid username and password.
3. Username validation is performed.
4. User logs in using registered credentials.
5. User is redirected to the dashboard.
6. User enters a long URL.
7. URL is validated.
8. A short code is generated.
9. Original URL and short code are stored in the database.

10. Shortened URL is displayed.
11. User can view previously shortened URLs.
12. Visiting a shortened URL redirects to the original URL.

---

7. Database Design

The database contains two main tables:

7.1 User Table

- **id** – Primary key
- **username** – Unique username
- **password** – Hashed password

7.2 URL Table

- **id** – Primary key
- **original_url** – Long URL entered by user
- **short_code** – Generated short identifier
- **user_id** – Foreign key linking to User table
- **created_at** – Timestamp of URL creation

This design ensures each URL is associated with a specific user.

---

8. Authentication and Security

User authentication is implemented using **Flask-Login**.
Key security practices include:

- Password hashing using Werkzeug
- Session management
- Login-required route protection
- Logout functionality

Passwords are never stored in plain text, ensuring data security.

---

9. URL Validation

URL validation ensures that only properly formatted URLs are accepted. This prevents:

- Invalid redirects
- Database pollution
- Poor user experience

Validation improves application reliability and correctness.

---

10. Short URL Generation Logic

Short URLs are generated using random alphanumeric characters. This approach:

- Produces compact URLs
- Reduces collision probability
- Ensures uniqueness
- Keeps URLs readable

This logic is simple and efficient for basic to medium-scale applications.

---

11. Features Implemented

- User signup with validation
- Secure login and logout
- URL shortening functionality
- URL validation
- User-specific URL history
- Database persistence
- ORM-based database interaction
- Bootstrap-based responsive UI
- Redirection from short URLs

---

12. Challenges Faced

Some challenges encountered during development include:

- Implementing authentication correctly
- Managing database relationships
- Handling username validation
- Associating URLs with specific users
- Ensuring secure password storage

---

13. Solutions and Learnings

These challenges were resolved by:

- Using Flask-Login for authentication
- Using SQLAlchemy ORM for database handling
- Applying validation logic during signup and URL entry
- Testing each feature incrementally

Key Learnings:

- Authentication workflows
- Secure password handling
- ORM-based database design
- Full-stack development practices
- Debugging and testing

---

14. Conclusion

The **URL Shortener Web Application for Advance Users** successfully meets all project objectives. It extends basic URL shortening functionality by adding secure user authentication and personalized URL history. This project strengthened my understanding of Flask, SQLAlchemy ORM, authentication systems, and real-world web application architecture.

Completing this project as part of the **Data Science with GenAI internship** provided valuable hands-on experience and improved my confidence in building full-stack applications.

---

15. Future Enhancements

Potential future improvements include:

- Click analytics
- Custom short URLs
- URL expiration
- User profile management
- Cloud deployment
- Role-based access control

End of Report