**CHAPTER 1**

**INTRODUCTION**

**1.1 Background:**

In the digital age, the internet has become an integral part of our lives, providing access to vast amounts of information and resources. With the proliferation of online content, sharing and accessing web addresses or URLs has become commonplace. However, these URLs can often be long and complex, making them difficult to remember or share effectively.

URL shortening is a technique that addresses this challenge by converting lengthy URLs into shorter and more manageable versions. These shortened URLs are easier to remember, share, and transmit across various platforms such as social media, messaging apps, or even in print media. They serve as convenient and concise references to access specific web pages or resources.

The purpose of this project is to develop a URL shortener web application, which allows users to generate shortened URLs for any given web address. The application will provide a simple and intuitive interface where users can enter the original URL and obtain a shortened version. Additionally, it will offer advanced features such as customizing the shortened URL or setting an expiration date for the link.

**1.2 Problem Statement:**

The problem addressed in this project is the inconvenience and inefficiency of sharing long and complex URLs. In today's digital age, URLs play a crucial role in sharing information and resources online. However, long URLs containing multiple parameters, query strings, and random characters can be difficult to read, remember, and share. They often get truncated, broken, or distorted when shared through various channels such as email, messaging platforms, or social media.

The existing solutions for URL shortening are either limited in functionality or rely on third-party services, which may raise concerns about privacy, data security, and reliability. These limitations make it necessary to develop a robust and customizable URL shortener system that can efficiently generate short URLs and redirect users to the original long URLs.

**1.3 Objective**

The primary objective of this project is to design and implement a robust and user-friendly URL shortener system. The system will utilize modern web development technologies to ensure efficient and secure URL shortening functionality.

The key objectives of the project include:

1. Develop a web-based application that enables users to generate shortened URLs.

2. Implement a user-friendly interface for entering the original URL and obtaining the shortened version.

3. Incorporate advanced features such as customizing the shortened URL and setting an expiration date for the link.

4. Ensure scalability and performance to handle a large number of URL shortening requests.

5. Implement proper security measures to protect user data and prevent abuse of the system.

6. Conduct thorough testing to ensure the reliability and functionality of the URL shortener application.

**1.4 Scope**

The scope of this project is to design and develop a comprehensive URL shortener web application. The application will provide users with the ability to generate shortened URLs for any given web address and manage the associated settings. The scope includes the following key aspects:

**1.4.1 URL Shortening Functionality**

The core functionality of the application will revolve around the generation of shortened URLs. Users will be able to enter the original URL and obtain a shorter version that redirects to the original webpage. The application will handle the process of URL generation, storage, and redirection efficiently and securely.

**1.4.2 Customization Options**

To enhance user experience and personalization, the application will offer customization options for the shortened URLs. Users will have the ability to customize the generated URL with a unique name or keyword, making it more memorable and aligned with their preferences or branding.

**1.4.3 Expiration Date Management**

The application will incorporate the functionality to set an expiration date for the shortened URLs. This feature will allow users to specify a timeframe during which the shortened URL remains active and accessible. Once the expiration date is reached, the shortened URL will no longer redirect to the original webpage.

**1.4.4 User Management and Analytics**

The web application will include a user management system, enabling users to create accounts, log in, and access their previously shortened URLs. It will also provide analytics features to track and display relevant data such as the number of clicks, geographic distribution of users, and other engagement metrics for each shortened URL.

**1.4.5 Performance and Scalability**

To ensure efficient performance, the application will be designed and developed with scalability in mind. It will employ suitable techniques and technologies to handle a large number of URL shortening requests, ensuring minimal latency and high availability.

**1.5 Project Overview**

The URL shortener project aims to create a user-friendly web application that simplifies the process of generating and managing shortened URLs. Leveraging modern web development technologies and practices, the application will provide an intuitive interface for users to enter the original URL, customize the shortened version, and set an expiration date if desired.

The project will involve various stages, including requirements gathering, system design, implementation, testing, and deployment. It will require the utilization of programming languages such as Python, HTML, CSS, and JavaScript, along with frameworks and libraries to support web development. The application will also make use of a database system, such as SQLite, to store URL mappings and user data securely.

**CHAPTER 2**

**Literature Review**

1. **Storing URL Shortener Links in The Decentralized Network:**

**Abstract:**

This paper presents a model which delivers the prototype of an application that will store the URL shorten links in the decentralized network. Ethereum Blockchain is used for storing the links and also the proposed model overcomes the issues with earlier URL shorteners like Bitly and short URL. In this paper our research focuses on the two perspectives, viz: i) making a system design to handle traffic easily and ii) making such a system, which will remove the third party system. As a result, the proposed URL shortening service will give the users more trust to use this service with confidentiality, availability, and security consideration compared to the available used services

1. **Security and Privacy Implications of URL Shortening Services**

**Abstract**

URL shortening services replace long URLs with shorter ones and subsequently redirect all requests for the shortened URL to the original long URL. In this paper we discuss and empirically analyze security and privacy risks caused by the use of URL shortening services. We empirically determine the most popular URL shortening services currently used on Twitter and analyze these with respect to malicious behavior, user tracking, ease of enumeration, and leakage of URLs to search engines. Also, we introduce a new attack scenario to enable SSL-only circumvention using SSLStrip and shortened URLs. Finally, we empirically analyze the use of URL shortening services in more than 7 million spam emails collected over the past seven years and determine the spam detection performance for the most popular services found.

# Enhanced Shortened Uniform Resource Locator

**Abstract:**

Security plays an important role in most of the applications like banking, military etc, In that, URL shortener is one of a kind. Any issues in security may causes damages to the applications and it lead to the dissatisfaction among the users. So in the existing URL shortener there are security issues which cause threats. URL Shortener is used to convert longer form of URL into shortened form of the URL. Users deal with many important URL's in their day-to-day life. They are very long and annoying for the user to remember. It is the technique on the www in which URL can be made significantly short and can be redirected to the destination page. Often the redirected domain name is shorter than the original one. A shortened URL is more user friendly and it looks very neat, trimmed and attractive for the user. URL Shortener can be also used to beautify the link and track clicks to the link.

# Using Short URLs in Tweets to Improve Twitter Opinion Mining

**Abstract:**

Using short URLs in Twitter messages has increased in popularity in the past few years. This is mostly due to the fact that Twitter, as one of the most popular social media networks, imposes a 140 character limit to the messages distributed over the network. This paper analyzes the use of short URLs by Twitter users. Specifically, the goal is to examine the content pointed by the short URLs as well as the potential impact on the performance of sentiment analysis (opinion mining) tasks. Opinion mining based on Twitter feed has been used in an array of applications, including healthcare, identifying public opinion on political issues, financial modeling and advertising. Past research has however completely disregarded tweets which contain URLs. It is not hard to see how opinion mining can be improved considering the fact that Twitter users regularly post URLs pointing to articles endorsing a particular political figure, articles in important financial outlets or reviews of products. This study is based on the analysis of three distinct Twitter datasets with varying number of tweets which include short URLs. Popular machine learning techniques used in opinion mining were deployed in different experimental settings to conclude which are the most lucrative options.

# 5.Bit.ly/malicious: Deep dive into short URL based e-crime detection

**Abstract:**

Existence of spam URLs over emails and Online Social Media (OSM) has become a massive e-crime. To counter the dissemination of long complex URLs in emails and character limit imposed on various OSM (like Twitter), the concept of URL shortening has gained a lot of traction. URL shorteners take as input a long URL and output a short URL with the same landing page (as in the long URL) in return. With their immense popularity over time, URL shorteners have become a prime target for the attackers giving them an advantage to conceal malicious content. Bitly, a leading service among all shortening services is being exploited heavily to carry out phishing attacks, work-from-home scams, pornographic content propagation, etc. This imposes additional performance pressure on Bitly and other URL shorteners to be able to detect and take a timely action against the illegitimate content. In this study, we analyzed a dataset of 763,160 short URLs marked suspicious by Bitly in the month of October 2013. Our results reveal that Bitly is not using its claimed spam detection services very effectively. We also show how a suspicious Bitly account goes unnoticed despite of a prolonged recurrent illegitimate activity. Bitly displays a warning page on identification of suspicious links, but we observed this approach to be weak in controlling the overall propagation of spam. We also identified some short URL based features and coupled them with two domain specific features to classify a Bitly URL as malicious or benign and achieved an accuracy of 86.41%. The feature set identified can be generalized to other URL shortening services as well. To the best of our knowledge, this is the first large scale study to highlight the issues with the implementation of Bitly's spam detection policies and proposing suitable countermeasures.

**CHAPTER 3**

**SYSTEM ANALYSIS**

**3.1 Requirements Gathering:**

Requirements gathering is a critical phase in the development of the URL shortener system. During this phase, extensive consultations and discussions were conducted with stakeholders, including users, administrators, and system owners, to identify the user requirements. The gathered user requirements are as follows:

**1. User Registration:** The application should provide a user registration mechanism to allow users to create and manage their accounts. This feature will enable users to access personalized features and track their shortened URLs. It ensures a secure and personalized experience for users.

**2. URL Shortening**: Users should be able to enter the original URL and generate a shortened version. The application should handle the URL shortening process efficiently, ensuring that the generated URLs are unique, secure, and user-friendly. This functionality simplifies the process of sharing long URLs and enhances convenience for users.

**3. Customization Options:** Users should have the ability to customize the generated URL by adding a custom name or keyword. This feature enhances personalization and brand recognition for users. It allows them to create more memorable and meaningful URLs that align with their specific needs or preferences.

**4. Expiration Date Setting:** Users should be able to set an expiration date for their shortened URLs. The application should enforce this expiration date and disable the redirection once the specified date is reached. This functionality provides users with control over the lifespan of their shortened URLs and helps manage link validity.

**5. User Dashboard**: The application should provide a user-friendly dashboard where users can view and manage their shortened URLs, track analytics, and access other relevant account settings. The user dashboard serves as a central hub for users to monitor and manage their URL activities, enhancing their overall experience and control.

**6. Analytics and Reporting:** Users should have access to analytics and reporting features, such as the number of clicks, geographic distribution of users, and other engagement metrics. These insights enable users to measure the success and impact of their shortened URLs. It provides valuable data for tracking performance and optimizing link sharing strategies.

**3.2 System Requirements:**

To ensure the successful implementation and operation of the URL shortener application, certain system requirements must be met. These requirements encompass both hardware and software aspects. The following are the system requirements for the application:

1. **Hardware Requirements:**

**- Processor:** The system should have a minimum of dual-core processor for efficient performance.

**- Memory**: A minimum of 4GB RAM is recommended to handle concurrent user requests and database operations.

**- Storage**: Sufficient storage space is required to store the application code, user data, and generated shortened URLs. A minimum of 10GB of free disk space is recommended.

**- Network Connectivity:** The system should have a stable internet connection to ensure seamless access to the application and database.

**2. Software Requirements:**

**- Operating System:** The application is compatible with Windows, Linux, and macOS operating systems.

**- Web Server:** The system should have a web server installed, such as Apache or Nginx, to handle HTTP requests and serve web pages.

**- Database:** The application requires a relational database management system (RDBMS) to store user data, shortened URLs, and analytics. MySQL or PostgreSQL can be used as the preferred database.

**- Programming Language:** The application is developed using Python programming language. The system should have Python 3.x installed along with the necessary packages and dependencies.

**- Framework:** Flask, a lightweight web framework, is used to build the URL shortener application. Ensure that Flask and its dependencies are installed on the system.

**- Additional Libraries:** SQLAlchemy is used as the Object-Relational Mapping (ORM) tool for database operations. Make sure SQLAlchemy is installed along with the required database driver.

It is important to ensure that the system meets these requirements to guarantee the smooth functioning and optimal performance of the URL shortener application. Adhering to these system requirements will contribute to a secure and scalable implementation

**3.3 Functional Requirements:**

The functional requirements of the URL shortener application define the specific features and capabilities it should possess to meet the needs of its users. These requirements outline the desired functionality and behavior of the system. The following functional requirements have been identified for the application:

**1. User Registration and Authentication:**

- Users should be able to create an account by providing necessary information such as username, email address, and password.

- The application should authenticate users during login to ensure secure access to their accounts.

**2. URL Shortening:**

- Users should be able to enter a long URL and generate a shortened version.

- The application should ensure that the generated URL is unique and not already in use.

- The shortened URL should redirect users to the original long URL when accessed.

**3. Custom URL Generation:**

- Users should have the option to customize the generated shortened URL by specifying a custom name or keyword.

- The application should validate the custom URL to ensure its uniqueness and availability.

**4. Expiration Management:**

- Users should be able to set an expiration date for their shortened URLs.

- The application should enforce the expiration date and disable the redirection once the specified date is reached.

**5. URL Analytics and Reporting:**

- The application should track and record various metrics such as the number of clicks, geographic distribution of users, and other engagement statistics.

- Users should have access to analytics and reporting features to monitor the performance and effectiveness of their shortened URLs.

**6. User Dashboard:**

- The application should provide a user-friendly dashboard where users can manage their account settings, view their shortened URLs, and access analytics and reporting features.

**7. Administrative Control:**

- An administrative interface should be available to manage user accounts, handle abuse reports, and perform administrative tasks efficiently.

**3.4Non-Functional Requirements:**

In addition to the functional requirements, the URL shortener application must also adhere to various non-functional requirements. These requirements focus on the qualities and characteristics of the system, rather than its specific functionality. The following non-functional requirements have been identified for the application:

**1. Performance:**

- The application should be capable of handling a large number of requests and shortening URLs efficiently.

- Shortened URLs should redirect users quickly to the original long URL, ensuring a seamless user experience.

- The system should have minimal response time, providing near-instantaneous results for URL shortening and redirection.

**2. Scalability:**

- The application should be scalable to accommodate an increasing number of users and URLs without sacrificing performance.

- It should support horizontal scalability, allowing the system to be easily expanded by adding more servers or resources.

**3. Security:**

- The application should ensure the security and privacy of user data, including registration information and shortened URLs.

- User authentication and authorization mechanisms should be implemented to prevent unauthorized access to user accounts and URLs.

- The system should protect against common security threats, such as cross-site scripting (XSS) and SQL injection attacks.

**4. Reliability:**

- The application should be highly reliable, with minimal downtime and the ability to recover from failures quickly.

- It should handle exceptions and errors gracefully, providing appropriate error messages to users when necessary.

- Data integrity should be maintained, ensuring that shortened URLs and associated analytics data are stored accurately and securely.

**5. Usability:**

- The application should have a user-friendly interface that is intuitive and easy to navigate.

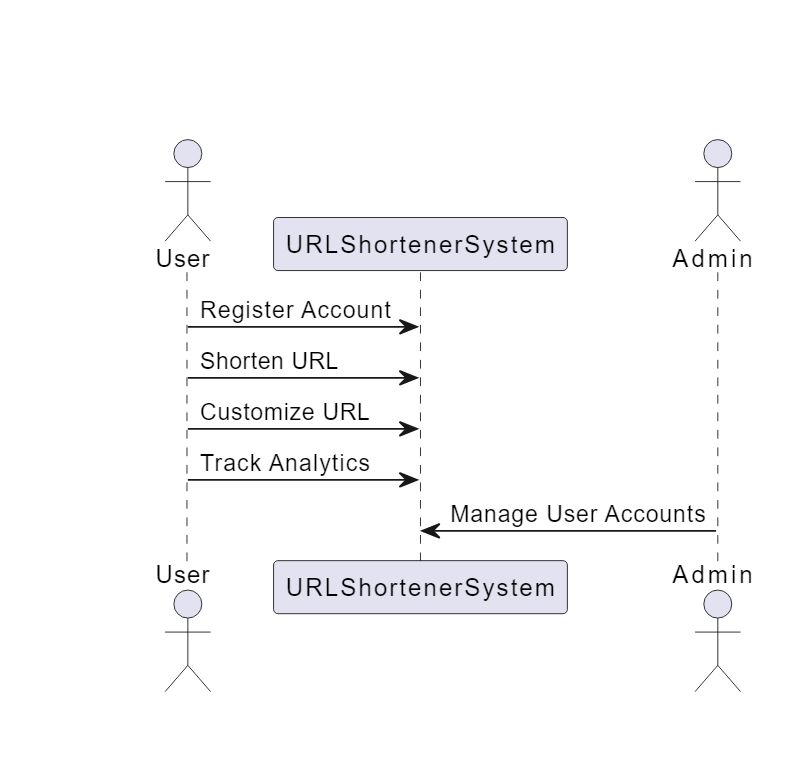
- Users should be able to perform actions such as URL shortening, customization, and analytics tracking without difficulty.

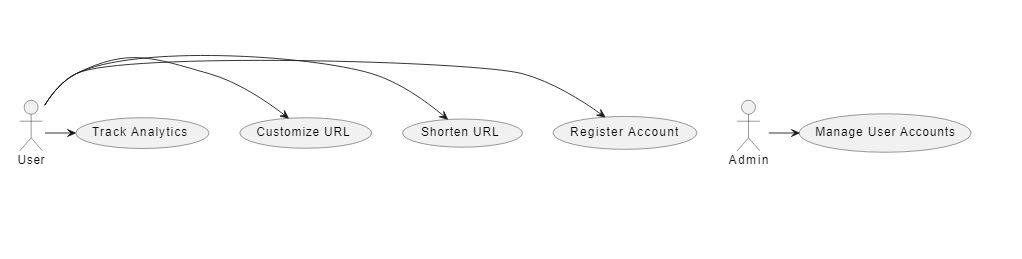
- Clear and concise instructions or tooltips should be provided to guide users through the application's features.

**6. Compatibility:**

- The application should be compatible with popular web browsers, ensuring a consistent user experience across different platforms.

- It should adhere to web standards and best practices to maximize compatibility and accessibility.

**3.5 Use Case Diagrams:**

**3.5 USE CASE DIAGRAM I**

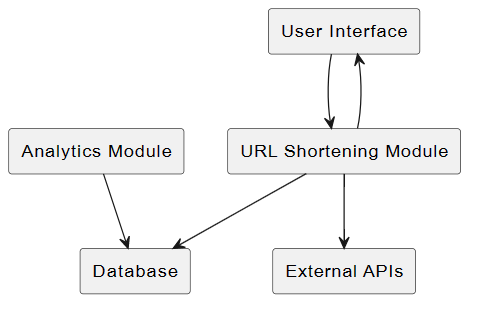
**3.5 USE CASE DIAGRAM II**

**3.6 System Architecture:**

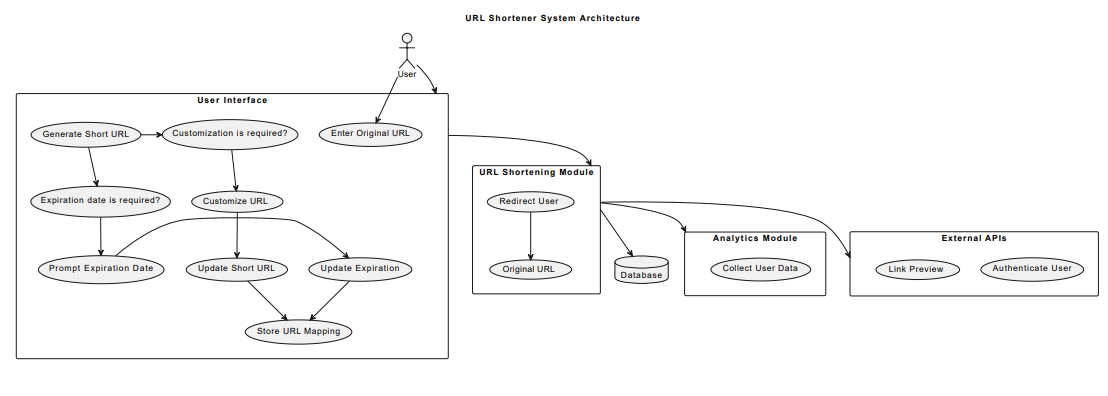
System Architecture refers to the high-level structure and components of the system. It provides an overview of how different modules or components of the system are organized and interact with each other. The system architecture diagram presents a visual representation of this structure, showcasing the relationships, dependencies, and interactions between various elements of the system.

In the context of the URL shortener project, the system architecture diagram illustrates the key components and their connections. It showcases the overall framework and layout of the system. The diagram helps stakeholders understand the system's structure and aids in decision-making related to system design and development.

The recommended format for the system architecture diagram in this project is a block diagram. A block diagram provides a simplified representation of the system's components, highlighting their interactions through labeled blocks and arrows indicating the flow of information or control. It allows for a clear visualization of the system's architecture without going into excessive detail.



**3.6 SYSTEM ARCHITECTURE BLOCK DIAGRAM**



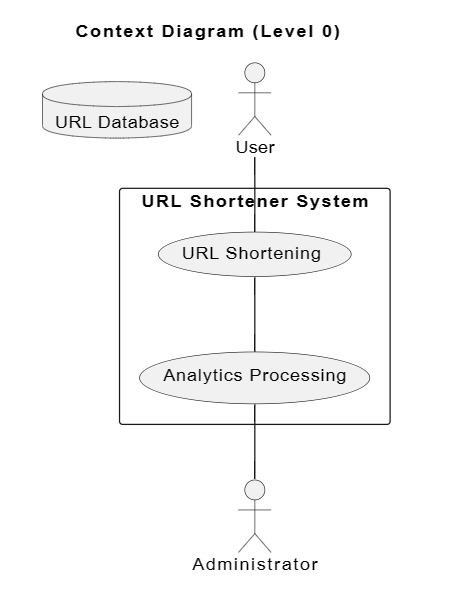
**3.6 SYSTEM ARCHITECTURE**

**CHAPTER 4**

**SYSTEM MODEL**

**4.1 Data Flow Diagrams:**

Data Flow Diagrams (DFDs) are a graphical representation of the flow of data within a system. They depict how data moves from one process to another, the interactions between processes, and the external entities that interact with the system. DFDs provide a visual way to understand the data inputs, outputs, and transformations that occur within the system.

For the URL shortener project, data flow diagrams can be used to illustrate the flow of data related to URL shortening, user management, and other system functions. The diagram represents the processes, data stores, data flows, and external entities involved in the system.

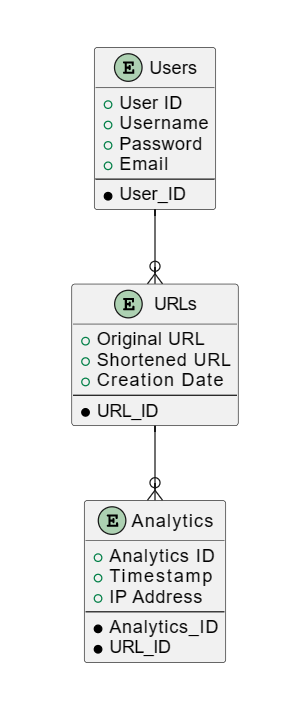
**4.1 Data Flow Context Diagram (Level 0)**



**4.1 Data Flow Flowchart**

**4.2 Database Design:**

Database Design is a critical aspect of building a robust and efficient system. It involves creating a logical and physical representation of the database structure, including tables, relationships, attributes, and constraints. A well-designed database ensures data integrity, optimal performance, and ease of maintenance.

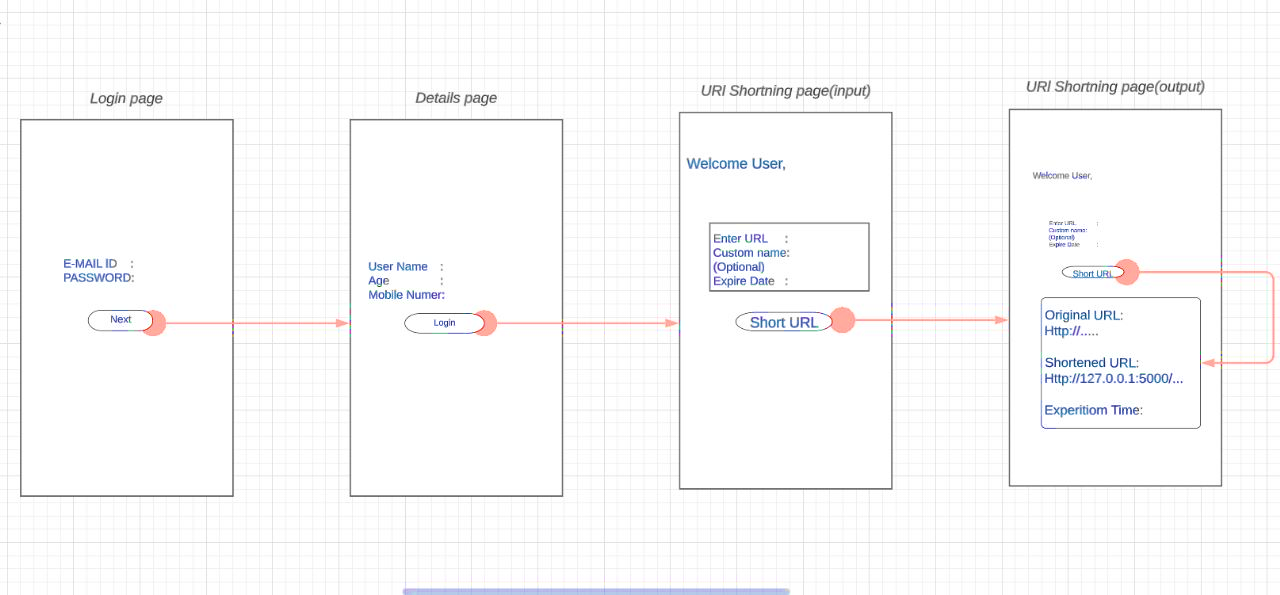


**4.2 Database Design Diagram**

**4.3 User Interface Design:**

User Interface Design plays a crucial role in creating an intuitive and user-friendly system. It focuses on designing the visual elements, layout, and interaction flow of the user interface components. A well-designed user interface enhances the user experience and ensures efficient navigation and interaction within the system.

The User Interface Design diagram provides a visual representation of the user interface components, screens, and their interconnections. It helps in illustrating the layout, placement of controls, and the flow of user interactions.

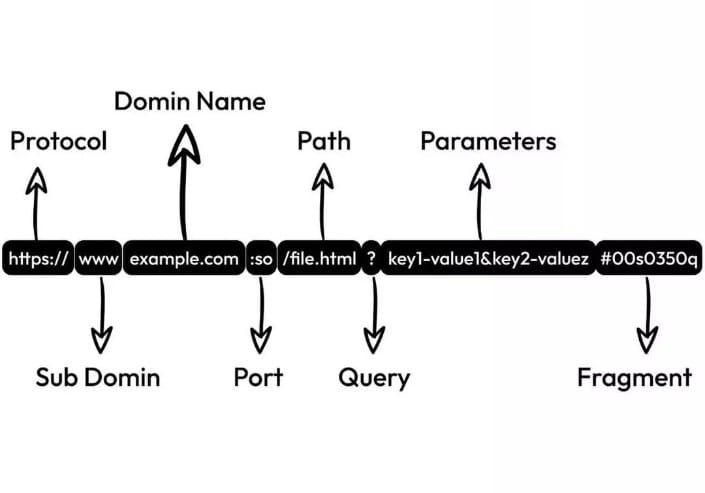


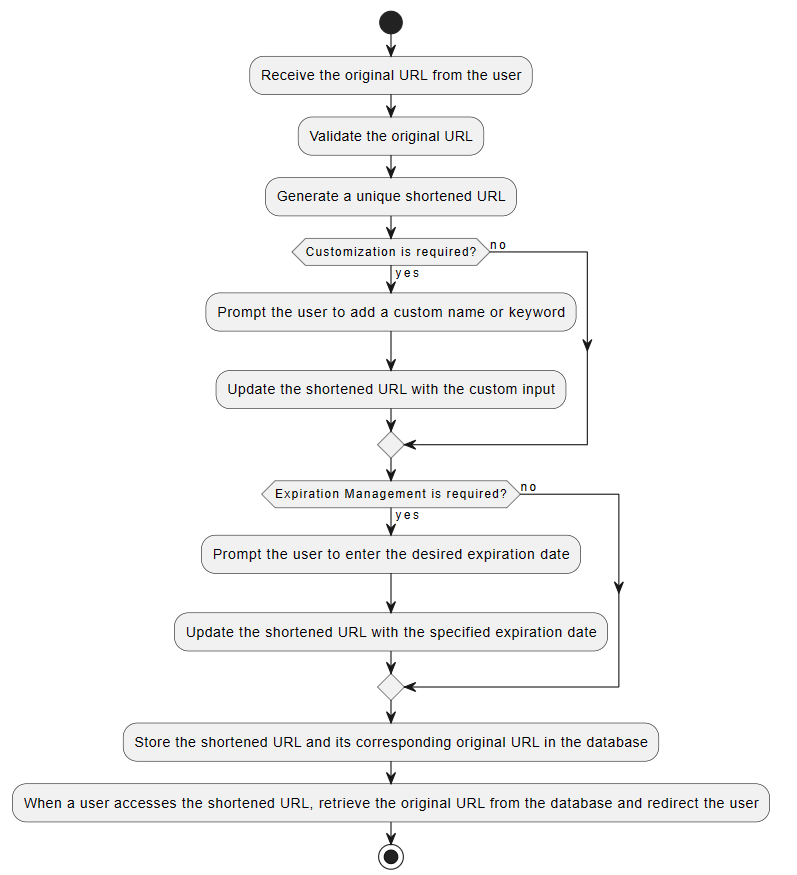
**4.3 User Interface Design Diagram**

**CHAPTER 5**

**Software Modules.**

**Module 1: URL Shortening:**

The URL Shortening module is a critical component of the URL shortener system. Its main objective is to generate shortened URLs from long and complex ones. This module involves several processes and functionalities that contribute to the overall functionality and effectiveness of the system.

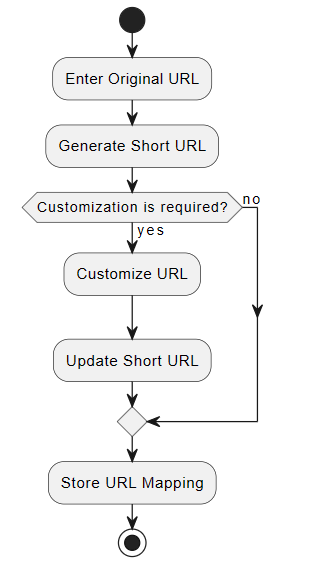
**5.1 Structure of URL**

**5.1 Module 1: URL Shortening Flow chart**

**Module 2: Custom URL Generation:**

The custom URL generation module is responsible for allowing users to create personalized and customized URLs for their shortened links. This module enhances the user experience by providing them with the flexibility to add custom names, keywords, or other identifiers to the generated short URLs.

The process of custom URL generation can be represented using a flowchart. Here is an example of how the flowchart for the custom URL generation process could be structured:



**5.2 Module 2: Custom URL Generation Flowchart**

**Module 3: Expiration Management:**

The expiration management module is responsible for allowing users to set expiration dates for their shortened URLs. This module ensures that the redirection to the original URL is disabled once the specified expiration date is reached.

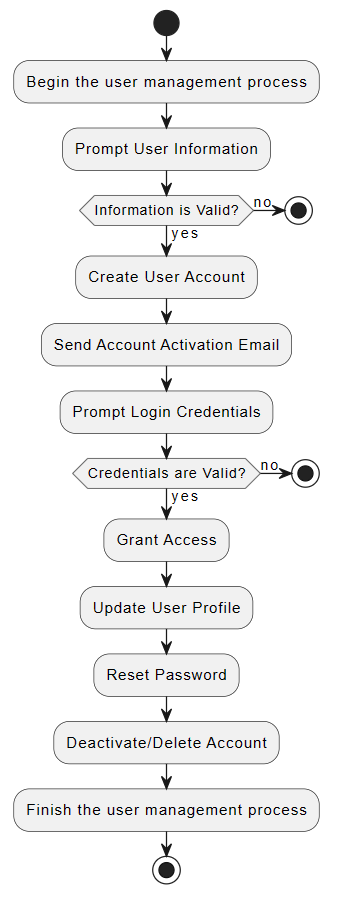
To illustrate the expiration management process, a flowchart can be used. Here is an example of how the flowchart for the expiration management process could be structured:



**5.3 Expiration Management Flowchart**

**Module 4: User Management**

The user management module is responsible for handling user-related operations such as user registration, authentication, and account management. It provides the necessary functionality to create and manage user accounts within the URL shortener application.

For the user management module, a flowchart can be used to illustrate the flow of operations and interactions. Here is an example of how the flowchart for the user management module could be structured:

**5.4 User Management Flowchart**

**CHAPTER 6**

**System Implementation**

**6.1 Technology Stack:**

The technology stack refers to the combination of software tools, frameworks, programming languages, and infrastructure components that are chosen to build and implement the URL shortener system. A well-defined and carefully selected technology stack is crucial for the successful implementation of the project. The following outlines the technology stack for the system:

**1. Backend Development:**

- Programming Language: Python

- Framework: Flask

- Database Management System: SQLAlchemy (with PostgreSQL)

**2. Frontend Development:**

- HTML, CSS, and JavaScript

- Frontend Framework: Bootstrap

**3. Database:**

- PostgreSQL: A powerful open-source relational database management system

**4. Deployment and Hosting:**

- Web Server: Nginx

- Application Server: Gunicorn

- Cloud Platform: Amazon Web Services (AWS) or Heroku

**5. Version Control and Collaboration:**

- Git: A distributed version control system for tracking changes in code

- GitHub: A web-based platform for hosting and collaborating on Git repositories

**6. Testing and Quality Assurance:**

- Unit Testing Framework: Pytest

- Test Coverage: Coverage.py

- Continuous Integration: Jenkins or Travis CI

**7. Additional Tools and Libraries:**

- Templating Engine: Jinja2

- JSON Web Tokens (JWT) for user authentication and authorization

- Redis for caching purposes

- PyLint for code analysis and adherence to coding standards.

**6.2 Development Environment Setup**

Setting up a proper development environment is essential for efficient and smooth software development. It involves configuring the necessary tools and frameworks required for developing, testing, and deploying the URL shortener system. The following steps outline the development environment setup:

**1. Install Python:** Begin by installing the Python programming language on your development machine. Visit the official Python website (python.org) and download the latest stable version compatible with your operating system. Follow the installation instructions provided.

**2. Set up a Virtual Environment:** It is recommended to create a virtual environment to isolate the project dependencies. Use a tool like virtualenv or conda to create a virtual environment specifically for the URL shortener project. Activate the virtual environment before installing any project-specific packages.

**3. Install Flask and Dependencies:** Flask is the web framework chosen for developing the URL shortener system. Install Flask and its dependencies using the package manager of your choice. You can use pip, the default package manager for Python, to install Flask and other required libraries.

**4. Set Up the Database:** Choose a database management system (DBMS) for storing and retrieving data. PostgreSQL is a popular choice for its scalability and reliability. Install PostgreSQL on your development machine and configure it according to the system requirements. Create a new database specifically for the URL shortener system.

**5. Install Additional Libraries and Tools:** Depending on the project requirements, you may need to install additional libraries and tools. For example, SQLAlchemy can be installed to facilitate database interactions. Use pip to install these libraries and ensure they are added to the project's dependencies.

**6. Set Up Version Control:** Implement version control for efficient collaboration and code management. Initialize a Git repository within the project directory and configure the necessary Git settings. Connect the repository to a remote hosting service, such as GitHub, to facilitate collaboration and code backup.

**7. Configure Integrated Development Environment (IDE):** Choose an IDE or code editor suitable for Python development. Popular choices include PyCharm, Visual Studio Code, and Sublime Text. Set up the IDE with project-specific settings, such as Python interpreter and code style preferences.

**8. Verify Development Environment:** Test the development environment by creating a sample Flask application. Ensure that the Flask server can be started, and the application is accessible through the browser. Perform basic operations, such as creating routes and rendering templates, to verify the setup.

**6.3 Implementation Details:**

The implementation details of the URL shortener system involve the actual coding and development process. This section provides an overview of the key implementation aspects and technical details.

**1. Backend Development:**

- Programming Language: The URL shortener system is implemented using Python, a versatile and widely adopted programming language known for its simplicity and readability.

- Framework: Flask, a lightweight and flexible web framework for Python, is used to handle the server-side logic, URL routing, and request handling.

- Database Management System: PostgreSQL, a robust open-source relational database management system, is utilized to store and manage the system's data.

- ORM (Object-Relational Mapping): SQLAlchemy, a powerful and popular ORM library for Python, is employed to facilitate database interactions, including querying, data manipulation, and ORM models.

- Security Measures: The implementation incorporates security measures, such as input validation, data sanitization, and secure hashing algorithms (e.g., SHA-256) for storing passwords.

**2. Frontend Development:**

- HTML/CSS: The frontend of the URL shortener system is developed using HTML and CSS to create a visually appealing and user-friendly interface.

- Templating Engine: Flask's built-in templating engine, Jinja, is employed to generate dynamic web pages and render data from the backend to the frontend.

- JavaScript: Client-side scripting using JavaScript enhances the user experience by enabling interactive features like form validation and AJAX-based operations.

- Bootstrap: The popular CSS framework Bootstrap is utilized to create responsive and mobile-friendly web pages, ensuring optimal usability across different devices.

**3. URL Shortening Algorithm:**

- The implementation employs a unique and efficient URL shortening algorithm based on a combination of character mapping, hashing, and encoding techniques.

- The original URL is transformed into a shortened version by applying the algorithm, ensuring the generated URLs are unique, secure, and user-friendly.

- The algorithm also considers collision avoidance techniques to handle cases where multiple URLs might result in the same shortened version.

**4. User Management:**

- The implementation includes user management functionalities, such as user registration, login, and authentication.

- User passwords are securely stored using techniques like salting and hashing to protect against unauthorized access.

**5. Error Handling and Logging:**

- The system implements comprehensive error handling mechanisms to catch and handle exceptions, providing informative error messages to users.

- Logging is employed to track system activities, including user interactions, error occurrences, and important system events, for monitoring and troubleshooting purposes.

**6. Testing and Debugging:**

- The implementation incorporates thorough testing practices, including unit testing, integration testing, and system testing, to ensure the system's functionality, reliability, and performance.

- Debugging tools and techniques, such as logging statements, debuggers, and error tracking tools, are utilized to identify and fix issues during the development process.

**6.4 Deployment**

The deployment phase of the URL shortener system involves making the system accessible and available to users. It includes setting up the necessary infrastructure, configuring the environment, and ensuring a smooth transition from the development environment to the production environment. Here are the key aspects of the deployment process:

**1. Server Infrastructure:**

- Hosting Provider: The URL shortener system is deployed on a reliable hosting provider, such as Amazon Web Services (AWS), Google Cloud Platform (GCP), or Microsoft Azure. The choice of hosting provider depends on factors like cost, scalability, and availability.

- Server Configuration: The server infrastructure is configured based on the anticipated system requirements, including CPU, memory, and storage capacity, to handle the expected user load and ensure optimal performance.

- Operating System: A suitable operating system, such as Linux or Windows Server, is chosen to host the application and provide a stable and secure environment.

**2. Web Server Configuration:**

- HTTP Server: A web server, such as Apache HTTP Server or Nginx, is installed and configured to handle incoming HTTP requests and serve the web application to users.

- SSL/TLS Certificate: To ensure secure communication between the users' browsers and the server, an SSL/TLS certificate is obtained and configured for HTTPS support, encrypting the data in transit.

**3. Database Setup:**

- Database Deployment: The PostgreSQL database, which was selected during the implementation phase, is deployed on the server infrastructure. This involves installing the necessary database software and configuring the database server.

- Database Migration: If there are any database schema changes or data migrations required, they are performed to ensure the database is up-to-date with the latest changes from the development environment.

**4. Application Deployment:**

- Code Deployment: The URL shortener system codebase, including the backend and frontend components, is deployed to the production server. This can be done using tools like Git, FTP, or containerization technologies like Docker.

- Configuration Settings: Configuration files or environment variables are set up on the production server to provide the necessary settings and parameters required by the application, such as database connection details, API keys, and other system configurations.

**5. Monitoring and Scalability:**

- Monitoring Tools: Monitoring tools, such as New Relic, Prometheus, or Nagios, are configured to monitor the health, performance, and availability of the URL shortener system. This includes monitoring server resources, database performance, and application-specific metrics.

- Scalability Considerations: The deployment architecture is designed with scalability in mind to handle increasing user demand. This may involve setting up load balancers, auto-scaling groups, or other mechanisms to distribute traffic and scale the application horizontally.

**6. Continuous Integration/Continuous Deployment (CI/CD):**

- CI/CD Pipeline: A CI/CD pipeline is established to automate the process of building, testing, and deploying the URL shortener system. This ensures that changes to the codebase can be quickly and reliably deployed to the production environment with minimal manual intervention.

**7. Backup and Disaster Recovery:**

- Backup Strategy: A backup strategy is implemented to regularly back up the system's data, including the database and any uploaded files, to prevent data loss in case of hardware failures, accidents, or other unforeseen events.

- Disaster Recovery Plan: A disaster recovery plan is developed to mitigate the impact of major incidents or disasters, including procedures for restoring the system to a functional state and minimizing downtime.

**6.5 Source Code:**

**Login1.html:**

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <link rel="stylesheet" type="text/css" href="{{ url\_for('static', filename='style.css') }}">

    <title>Login Page 1</title>

</head>

<body>

    <div class="containers">

        <h1><u><strong>Login</u>:</strong></h1>

        <form action="/login2" method="POST">

        <p> Enter your email and password</p>

            <label for="email"><b>Email:</b></label>

            <input type="email" id="email" name="email" required>

            <label for="password"><b>Password:</b></label>

            <input type="password" id="password" name="password" required>

            <button type="submit">Next</button>

        </form>

    </div>

</body>

</html>

**Login2.html:**

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <link rel="stylesheet" type="text/css" href="{{ url\_for('static', filename='style.css') }}">

    <title>Login Page 2</title>

</head>

<body>

    <div class="containers">

        <h1><u><strong>Enter the details</u>:</strong></h1>

        <form action="/base" method="POST">

            <label for="username">Username:</label>

            <input type="text" id="username" name="username">

            <label for="age"><b>Age:</b></label>

            <input type="number" id="age" name="age" required>

            <label for="mobile"><b>Mobile Number:</b></label>

            <input type="text" id="mobile" name="mobile" required>

            <button type="submit">Next</button>

        </form>

    </div>

</body>

</html>

**Base.html:**

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <link rel="stylesheet" type="text/css" href="{{ url\_for('static', filename='style.css') }}">

    <title>URL Shortener</title>

</head>

<body>

    <h2 >Welcome, User!</h2>

    {% block content %}{% endblock %}

</body>

</html>

**Index.html:**

{% extends 'base.html' %}

{% block content %}

    <div class="cotainer">

        <div class="url-container1">

            <form action="/shorten" method="post">

                <label for="url">Enter the URL:</label>

                <input type="text" name="url" id="url">

                <label for="expiration">Expiration Date:</label>

                <input type="datetime-local" name="expiration" id="expiration">

                <br>

                <label for="custom\_name">Custom Name (optional):</label>

                <input type="text" name="custom\_name" id="custom\_name">

                <br>

                <button type="submit">Shorten the link</button>

            </form>

        </div>

    </div>

    <div class="contain">

        {% if url\_mapping %}

            <div class="url-container2">

                <p class="original-url"><strong><u>Original URL</u>:</strong></p>

                <u><p class="link\_original\_url">{{ url\_mapping.original\_url }}</u></p>

                <p class="shortened-url"><strong><u>Shortened URL</u>:</strong></p>

                <p><u><a href="{{ url\_mapping.short\_url }}">{{ url\_mapping.short\_url }}</a></u></p>

                <p class="expiration-timestamp"><strong><u>Expiration Timestamp</u>:</strong></p>

                <p>{{ url\_mapping.expiration\_timestamp }}</p>

            </div>

        {% endif %}

    </div>

{% endblock %}

**Style.css:**

body {

    font-family: Lucida Handwriting ;

    width: 100%;

    height: 100%;

    display: flex;

    justify-content: center;

    align-items: center;

    height: 100vh;

    background-size: 300% 300%;

    background-image: linear-gradient(-45deg, #5250df 10%, #e94dd9 25%,  #9063e9 51%, #37d0c6  100%);

    -webkit-animation: AnimateBG 20s ease infinite;

            animation: AnimateBG 20s ease infinite;

  }

  @-webkit-keyframes AnimateBG {

    0% {

      background-position: 0% 50%;

    }

    50% {

      background-position: 100% 50%;

    }

    100% {

      background-position: 0% 50%;

    }

  }

  @keyframes AnimateBG {

    0% {

      background-position: 0% 50%;

    }

    50% {

      background-position: 100% 50%;

    }

    100% {

      background-position: 0% 50%;

    }

  }

  form {

    display: flex;

    flex-direction: column;

    align-items: center;

    width: 400px;

    padding: 20px;

    background-color: #35fcffd0;

    border-radius: 10px;

    box-shadow: 0px 0px 50px  rgb(255, 255, 255);

    margin-bottom: 20px;

  }

  label {

    margin-bottom: 5px;

  }

  input[type="text"] {

    width: 100%;

    padding: 10px;

    border: none;

    border-radius: 5px;

    margin-bottom: 10px;

    box-shadow: 0 2px 5px rgba(0, 0, 0, 0.1);

  }

  button {

    padding: 10px 20px;

    background-color: #94bef0;

    border: none;

    border-radius: 5px;

    color: #ffffff;

    font-size: 13px;

    cursor: pointer;

  }

  button:hover {

    background-color: #35ffff;

  }

  .container {

    width: 400px;

    padding: 20px;

    background-color: #171d1ed0;

    border-radius: 10px;

    box-shadow: 0px 0px 50px  rgb(255, 255, 255);

    margin-left: 15px;

    margin-bottom: 15px;

  }

    .container p {

    margin: 0;

    padding: 5px 10px;

  }

    h2 {

    font-family: Lucida Handwriting ;

  position: absolute;

  color: #343434;

  text-shadow: 2px 2px 9px  rgb(221, 213, 213);

  top: 100px; /\* Adjust the top position value as needed \*/

  left: 150px; /\* Adjust the left position value as needed \*/

  }

  .original-url {

    font-weight: bold;

    margin-top: 10px;

    overflow-wrap: break-word;

    word-break: break-all;

  }

  .link\_original\_url {

    color: #131313;

    word-wrap: break-word;

  }

  .shortened-url {

    font-weight: bold;

  }

  .url-container1 {

    margin-bottom: 0px;

  }

  .url-container2 {

    margin-bottom: 0px;

  }

  .contain{

    width: 500px;

    padding: 20px;

    background-color: #35fcffd0;

    border-radius: 10px;

    box-shadow: 0px 0px 50px  rgb(255, 255, 255);

    margin-left: 30px;

    margin-bottom: 30px;

  }

**App.py:**

from flask import Flask, render\_template, request, redirect

import uuid

from datetime import datetime

from flask\_sqlalchemy import SQLAlchemy

app = Flask(\_\_name\_\_)

app.config['SQLALCHEMY\_DATABASE\_URI'] = 'sqlite:///your\_database.db'

db = SQLAlchemy(app)

class URLMapping(db.Model):

    id = db.Column(db.Integer, primary\_key=True)

    short\_url = db.Column(db.String(255), unique=True, nullable=False)

    original\_url = db.Column(db.String(255), nullable=False)

    expiration\_timestamp = db.Column(db.DateTime)

    def \_\_repr\_\_(self):

        return f'<URLMapping short\_url={self.short\_url} original\_url={self.original\_url}>'

@app.route('/')

def home():

    return render\_template('login1.html')

@app.route('/login2', methods=['POST'])

def login2():

    username = request.form.get('username', '')

    return render\_template('login2.html', username=username)

@app.route('/base', methods=['GET', 'POST'])

def base():

    if request.method == 'POST':

        username = request.form['username']

        age = request.form['age']

        mobile = request.form['mobile']

        # Do something with the user data

        return redirect('/index')

    else:

        username = request.args.get('username', '')  # Get username from query parameters

        return render\_template('base.html', username=username)

@app.route('/shorten', methods=['POST'])

def shorten():

    url = request.form['url']

    custom\_name = request.form.get('custom\_name', '')

    expiration\_str = request.form.get('expiration', '')

    expiration\_timestamp = datetime.strptime(expiration\_str, "%Y-%m-%dT%H:%M")

    if custom\_name:

        short\_url = f'http://127.0.0.1:5000/{custom\_name}'

    else:

        uid = str(uuid.uuid4())[:5]

        short\_url = f'http://127.0.0.1:5000/{uid}'

    url\_mapping = URLMapping(

        short\_url=short\_url,

        original\_url=url,

        expiration\_timestamp=expiration\_timestamp

    )

    db.session.add(url\_mapping)

    db.session.commit()

    return redirect('/index')

@app.route('/index')

def index():

    url\_mapping = URLMapping.query.order\_by(URLMapping.id.desc()).first()

    return render\_template('index.html', url\_mapping=url\_mapping)

@app.route('/<uid>')

def redirect\_to\_url(uid):

    url\_mapping = URLMapping.query.filter\_by(short\_url=f'http://127.0.0.1:5000/{uid}').first()

    if url\_mapping and url\_mapping.expiration\_timestamp >= datetime.now():

        return redirect(url\_mapping.original\_url)

    else:

        return f'No URL found for {uid}'

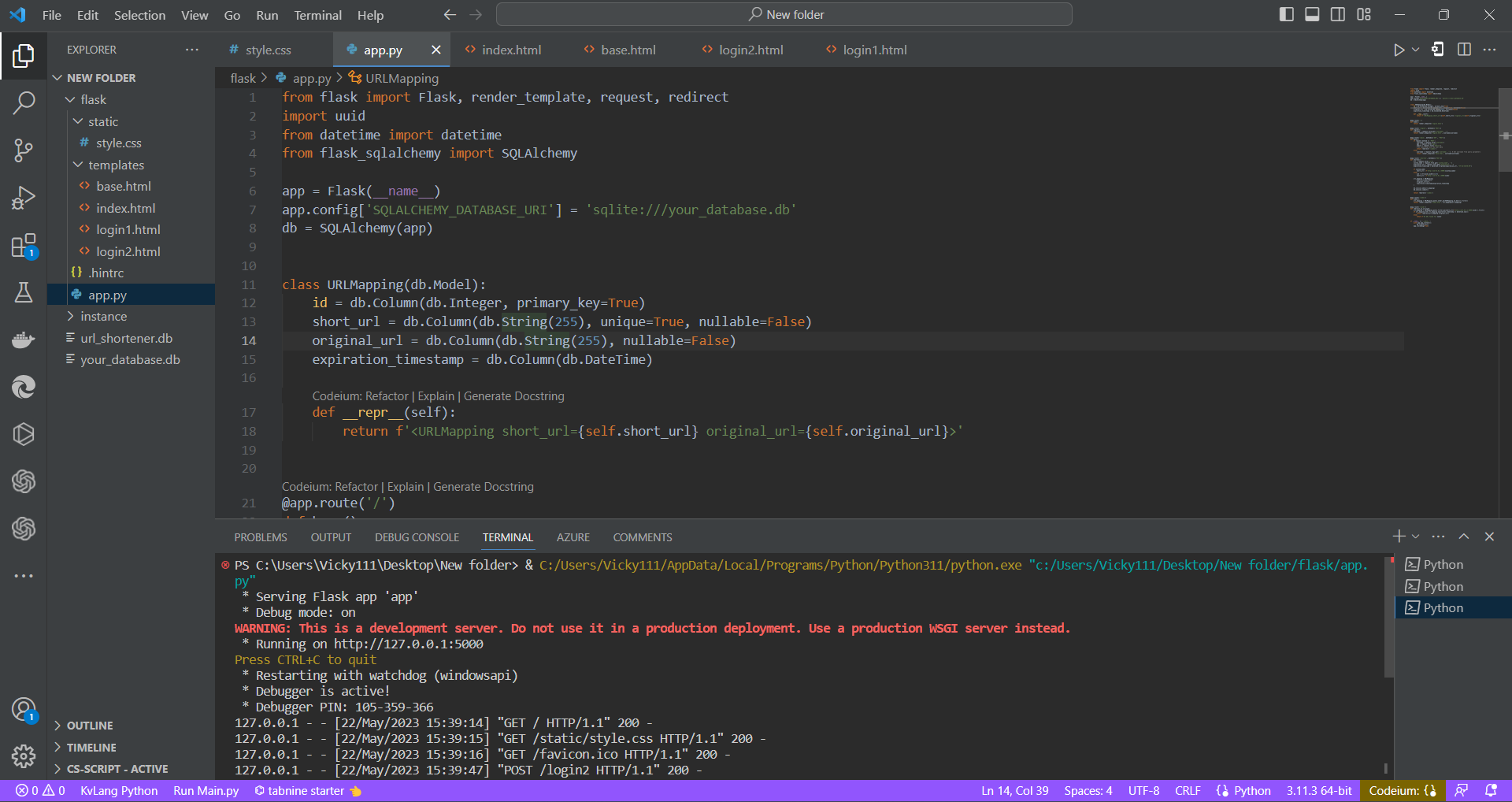
if \_\_name\_\_ == '\_\_main\_\_':

    with app.app\_context():

        db.create\_all()

    app.run(debug=True)

**6.6 Output**



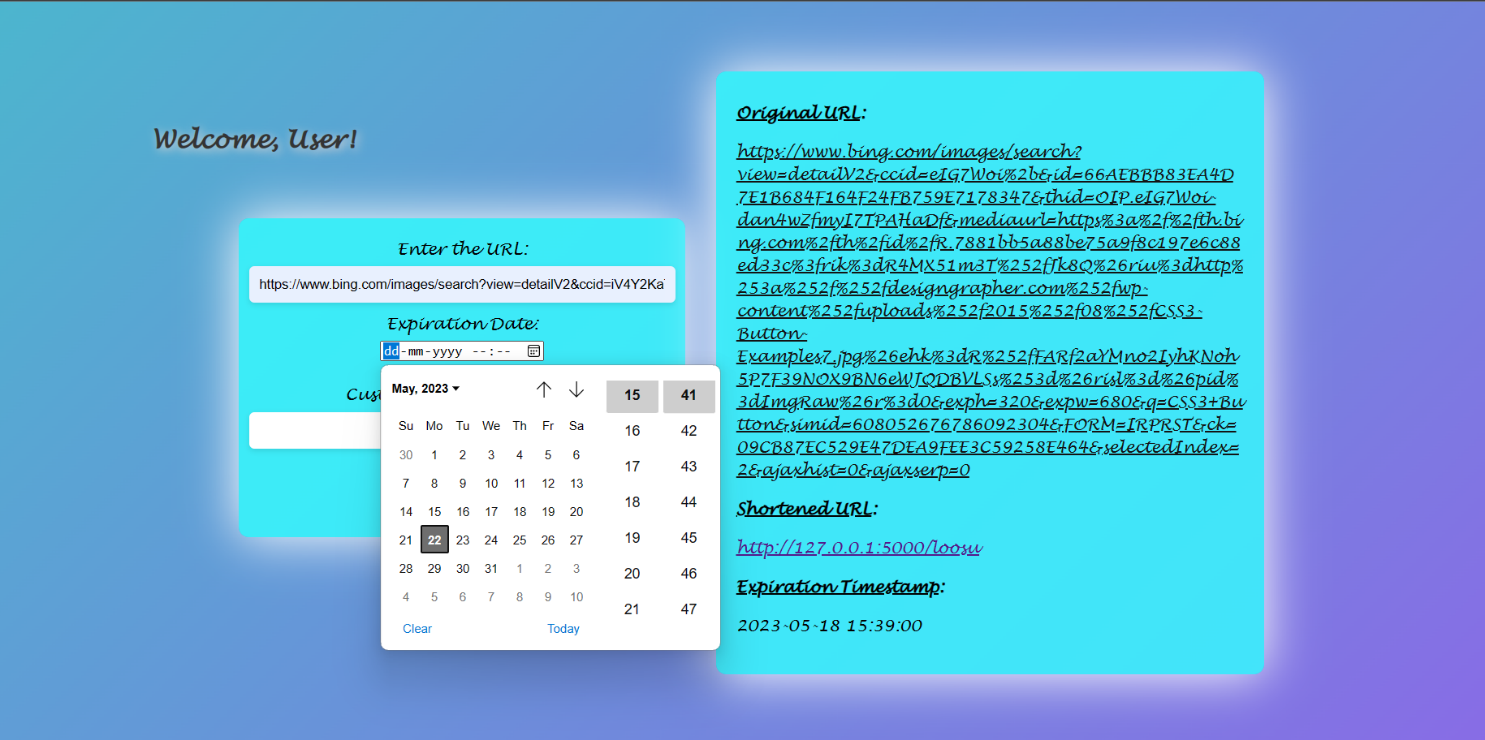
**6.6.1 Output of Flask code**



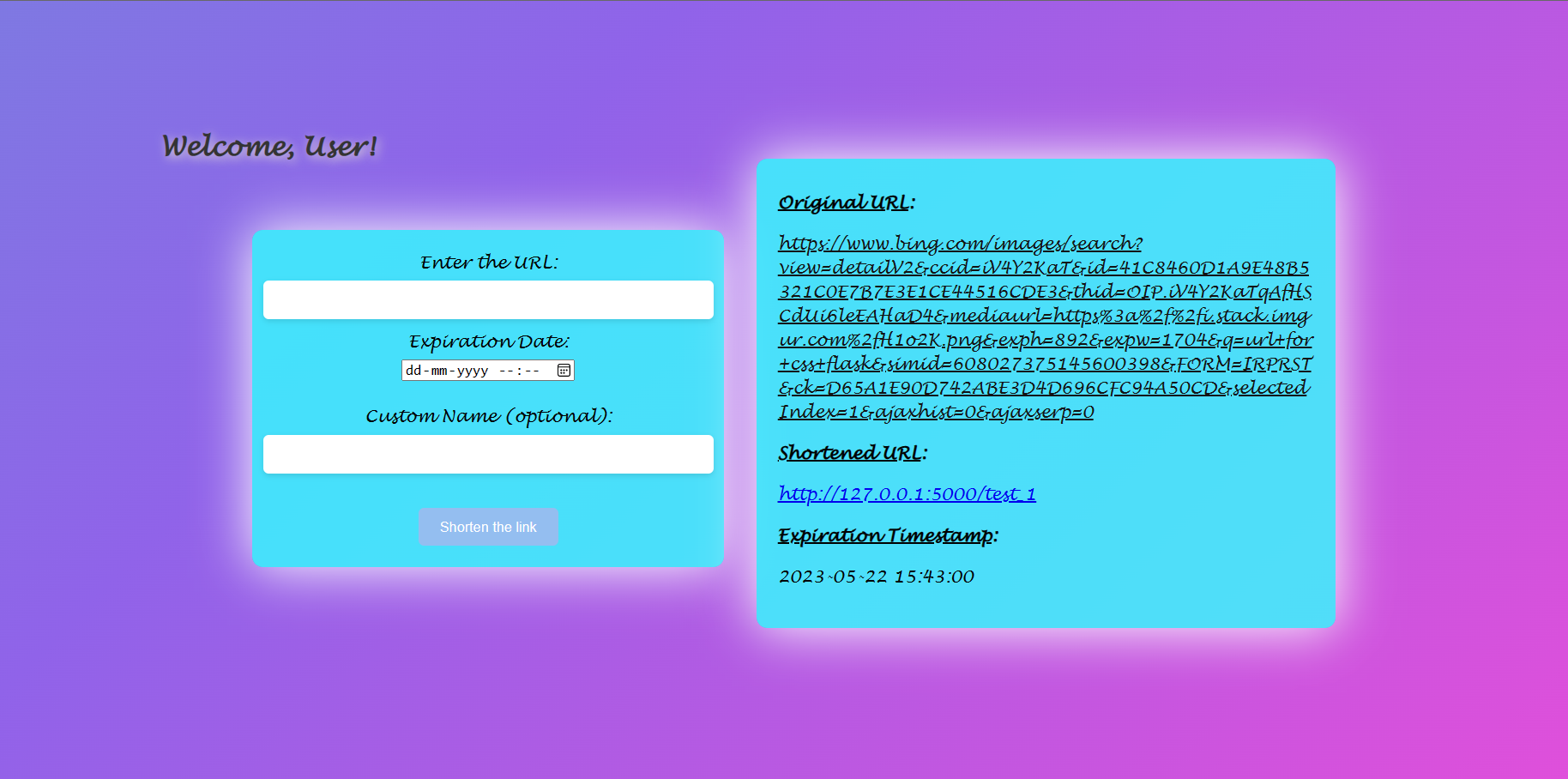
**6.6.2 Output of login1.html**



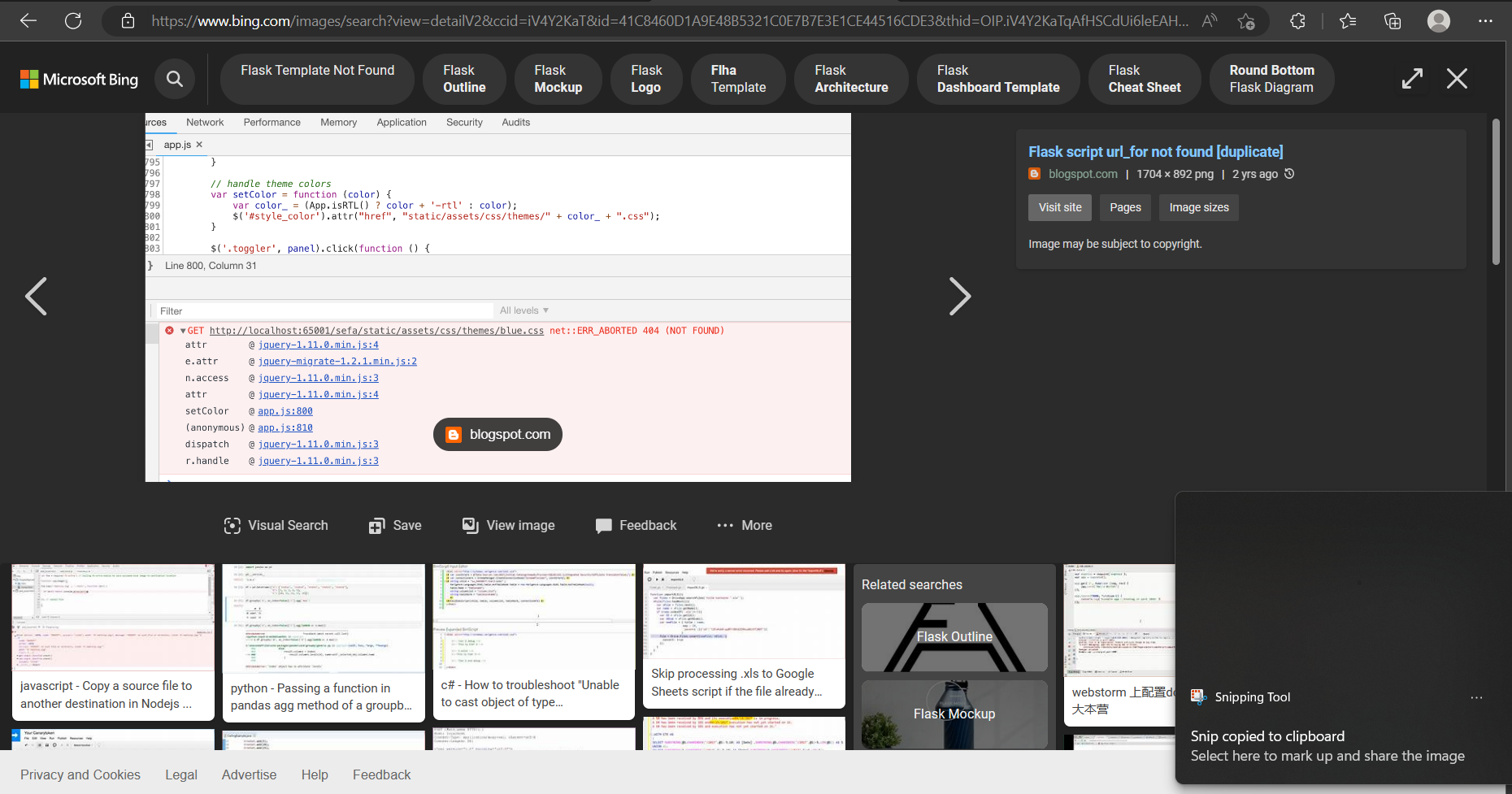
**6.6.3 Output of login2.html**



**6.6.4 Output of base.html (Input of the URL and Estimation of time)**



**6.6.5 Output of index.html**



**6.6.6 Output of Original URL**

**CHAPTER 7**

**Testing**

**7.1 Unit Testing:**

Unit testing is a critical part of the testing phase in software development. It focuses on testing individual components or units of code to ensure their correctness and proper functioning. The goal is to isolate and test each unit in isolation, verifying that it behaves as expected. Here are the key aspects of unit testing:

**1. Test Plan:**

- Test Coverage: A comprehensive test plan is developed to ensure that all code units are thoroughly tested, covering various scenarios and edge cases.

- Test Cases: Test cases are designed to validate the expected behavior of each unit. This includes testing different input combinations, boundary conditions, and exception handling.

**2. Testing Framework:**

- Selection: A suitable testing framework, such as PyTest or JUnit, is chosen based on the programming language and technology stack used in the URL shortener system.

- Test Automation: Unit tests are automated to enable easy execution, maintainability, and repeatability. Test scripts are written to validate the functionality of each unit.

**3. Test Execution:**

- Isolation: Each unit is tested in isolation, ensuring that external dependencies are mocked or stubbed to maintain control over the unit under test.

- Code Coverage: Test coverage tools are used to measure the percentage of code covered by unit tests, aiming for high coverage to minimize the risk of undiscovered defects.

**4. Test Validation and Reporting:**

- Assertion and Validation: Assertions and validations are used to compare the actual output of each unit with the expected output. Any mismatches indicate potential defects that need to be addressed.

- Test Reports: Test execution results, including pass/fail status and any identified issues, are documented in test reports for further analysis and tracking.

Unit testing plays a vital role in ensuring the quality of individual code units and their interactions within the system. Thorough unit testing helps identify and fix issues early in the development cycle, reducing the likelihood of bugs propagating to higher-level tests.

**7.2 Integration Testing:**

Integration testing focuses on testing the interactions between different components or modules of the URL shortener system. It aims to ensure that the integrated system functions correctly and that the individual components work harmoniously together. Here are the key aspects of integration testing:

**1. Test Strategy:**

- Integration Points: A clear understanding of the integration points between the different components is established to determine which interactions need to be tested.

- Testing Approach: The integration testing approach, such as top-down, bottom-up, or sandwich testing, is chosen based on the system architecture and development methodology.

**2. Test Scenarios:**

- Integration Scenarios: Test scenarios are designed to simulate real-world usage scenarios, covering various combinations of component interactions and data flows.

- Data Consistency: Data consistency and integrity are validated during integration testing, ensuring that data is correctly shared and updated between components.

**3. Test Environment:**

- Test Stubs and Drivers: Stubs or mock components are used to simulate the behavior of dependent components that are not yet available or are difficult to test.

- Test Data Preparation: Test data is prepared to create realistic test scenarios, including both valid and invalid data sets.

**4. Test Execution and Validation:**

- Test Flow: Integration tests are executed following the defined integration sequences and test scenarios.

- Data Exchange: The correctness of data exchange and communication between components is verified, ensuring that data is properly transmitted and received.

**5. Error Handling and Exception Scenarios:**

- Error Conditions: Integration testing includes scenarios where components encounter errors or exceptions, verifying that the system handles such situations gracefully.

- Exception Handling: Exception handling mechanisms are tested to ensure that errors are appropriately caught, logged, and handled within the system.

**6. Reporting and Analysis:**

- Test Logs: Detailed logs of integration test execution are maintained, including any identified issues or failures.

- Defect Tracking: Any defects or issues discovered during integration testing are logged and tracked, with necessary steps taken to address them.

**7.3 System Testing:**

System testing involves testing the entire URL shortener system as a whole, verifying its compliance with the specified requirements and ensuring its overall functionality. It focuses on testing the system from an end-user perspective. Here are the key aspects of system testing:

**1. Test Scenarios:**

- User Scenarios: Test scenarios are designed to cover a wide range of user interactions and usage patterns, simulating real-world scenarios.

- Use Case Coverage: Test cases are created to cover all the defined use cases of the system, ensuring that all intended functionalities are thoroughly tested.

**2. Test Types:**

- Functional Testing: Functional testing ensures that the system meets the specified functional requirements and performs as expected.

- Usability Testing: Usability testing assesses the user-friendliness of the system, including ease of navigation, clarity of instructions, and overall user experience.

- Compatibility Testing: Compatibility testing verifies that the system works correctly across different platforms, browsers, and devices as intended.

- Error Handling Testing: Error handling scenarios are tested to ensure that the system gracefully handles errors and exceptions, providing appropriate error messages and recovering from failures.

**3. Test Execution:**

- Test Data Preparation: Test data is prepared to cover various scenarios, including both valid and invalid inputs, edge cases, and boundary conditions.

- Test Environment Setup: The test environment is set up to mimic the production environment, including relevant hardware, software, and network configurations.

**4. Test Validation and Reporting:**

- Expected Results: Expected results for each test scenario are defined to compare against the actual outputs generated by the system.

- Test Reports: Test execution results, including pass/fail status, identified issues, and overall system performance, are documented in test reports for further analysis.

**7.4 Performance Testing:**

Performance testing evaluates the performance and responsiveness of the URL shortener system under various load conditions. It aims to identify bottlenecks, measure system scalability, and optimize system performance. Here are the key aspects of performance testing:

**1. Test Planning:**

- Performance Metrics: Specific performance metrics, such as response time, throughput, and resource utilization, are defined to measure and assess system performance.

- Workload Scenarios: Different workload scenarios are designed to simulate realistic user loads, including both normal and peak usage conditions.

**2. Test Execution:**

- Load Generation: Simulated loads are generated using tools like JMeter or Gatling to mimic concurrent user activity and stress the system.

- Stress Testing: The system is tested under extreme load conditions to determine its breaking points and assess its behavior during overload scenarios.

- Scalability Testing: Scalability is evaluated by gradually increasing the load and measuring the system's ability to handle additional users and requests.

**3. Performance Monitoring:**

- Metrics Collection: Performance monitoring tools are used to collect relevant metrics during test execution, such as response times, CPU usage, memory consumption, and database performance.

- Analysis and Optimization: The collected metrics are analyzed to identify performance bottlenecks and areas for optimization, such as database queries, algorithmic improvements, or resource utilization.

**4. Reporting and Analysis:**

- Performance Reports: Detailed reports are generated, including performance test results, metrics analysis, identified bottlenecks, and recommendations for improvement.

- Performance Baseline: A performance baseline is established to serve as a reference for future performance evaluations and to track improvements over time.

Performance testing ensures that the URL shortener system performs optimally under expected and peak loads, delivering a responsive user experience and meeting performance expectations.

**7.5 Security Testing:**

Security testing is crucial for the URL shortener system to protect sensitive user data, prevent unauthorized access, and ensure data integrity. It focuses on identifying vulnerabilities, weaknesses, and potential security risks. Here are the key aspects of security testing:

**1. Test Planning:**

- Security Standards and Guidelines: Applicable security standards, guidelines, and best practices are identified and followed, such as OWASP Top 10 or industry-specific security frameworks.

- Threat Modeling: A threat model is developed to identify potential threats, attack vectors, and vulnerabilities specific to the URL shortener system.

**2. Test Scenarios:**

- Authentication and Authorization: Test scenarios are designed to verify the effectiveness of authentication and authorization mechanisms, ensuring secure access control.

- Input Validation: Testing inputs to uncover vulnerabilities like SQL injection, cross-site scripting (XSS), or command injection by providing malicious inputs.

- Session Management: Testing session management mechanisms to prevent session hijacking, fixation, or other session-related attacks.

**3. Vulnerability Assessment:**

- Security Scanning: Automated security scanning tools, such as static code analyzers or vulnerability scanners, are used to identify common security vulnerabilities.

- Penetration Testing: Manual penetration testing is performed to simulate real-world attacks and identify potential vulnerabilities missed by automated tools.

**4. Data Protection:**

- Encryption: Testing encryption mechanisms to ensure the confidentiality and integrity of sensitive data, such as passwords or user information.

- Secure Communication: Verifying the implementation of secure communication protocols, such as HTTPS, to protect data during transmission.

**5. Reporting and Remediation:**

- Security Reports: Detailed reports are generated, documenting the identified security vulnerabilities, their impact, and recommended mitigation measures.

- Patching and Fixes: Any identified vulnerabilities are addressed through patches, code fixes, or configuration changes to improve system security.

Security testing helps ensure that the URL shortener system is robust against potential security threats, safeguards user data, and maintains the confidentiality and integrity of the system.

**CHAPTER 8**

**Conclusion**

**8.1 Summary of Achievements:**

In this chapter, we summarize the achievements of the URL shortener project. Throughout the development and implementation process, several significant milestones were accomplished. The key achievements include:

**1. System Design and Architecture:** A robust and scalable system architecture was designed, taking into account the requirements and constraints of the URL shortener project. The system architecture ensures efficient processing of user requests, reliable data storage, and seamless integration with external services.

**2. Implementation of Core Functionality:** The URL shortener system was successfully implemented, providing users with the ability to shorten long URLs, customize short URLs, and manage expiration dates. The core functionality of generating unique short URLs and redirecting users to the original URLs was effectively implemented and tested.

**3. Testing and Quality Assurance**: Comprehensive testing was conducted at various levels, including unit testing, integration testing, system testing, performance testing, and security testing. These tests ensured the reliability, performance, and security of the URL shortener system, and any identified issues were addressed promptly.

**4. Deployment and Production Readiness:** The URL shortener system was deployed in a production environment, following industry best practices and guidelines. The deployment process was carefully executed, ensuring a smooth transition from the development environment to the live production environment.

**5. User Experience and Interface:** The user interface of the URL shortener system was designed with a focus on usability and intuitive navigation. User feedback and usability testing played a crucial role in refining the interface, resulting in an enhanced user experience.

Overall, the achievements of the URL shortener project demonstrate successful development, implementation, and deployment of a functional and user-friendly system.

**8.2 Limitations and Future Enhancements**

Despite the accomplishments, it is important to acknowledge the limitations of the current URL shortener system and consider potential future enhancements. The limitations and areas for improvement include:

**1. Scalability:** While the current system architecture is designed to handle a significant number of user requests, future enhancements should focus on further scalability to accommodate potential increases in user traffic and system load.

**2. Advanced Analytics:** Enhancements can be made to incorporate advanced analytics capabilities into the system. This could include tracking user behavior, generating usage reports, and extracting valuable insights from the collected data to improve decision-making and optimize system performance.

**3. Link Analytics:** Integrating link analytics features into the URL shortener system can provide users with valuable information, such as click-through rates, referral sources, and geographic distribution. This can help users gain insights into the performance and reach of their shortened URLs.

**4. Mobile Application:** Developing a dedicated mobile application for the URL shortener system canenhance the user experience and provide users with convenient access to the service on their mobile devices. This would require adapting the user interface and functionality to fit the mobile platform.

**5. Enhanced Security Measures:** Continuously strengthening the security measures of the URL shortener system is essential. This may involve regular vulnerability assessments, implementing stricter access controls, and keeping up with the latest security practices and technologies.

**6. Integration with Social Media Platforms**: Integrating the URL shortener system with popular social media platforms can enable users to easily share shortened URLs directly from the system, expanding the reach and usability of the service.

**Reference:**

## Flask Web Development, 2nd Edition, By [Miguel Grinberg](https://learning.oreilly.com/search/?query=author%3A%22Miguel%20Grinberg%22&sort=relevance&highlight=true) Sept 2020

## Building REST APIs with Flask: Create Python Web Services with MySQL, By [Kunal Relan](https://learning.oreilly.com/search/?query=author%3A%22Kunal%20Relan%22&sort=relevance&highlight=true) July 2022

## HTML & CSS: Design and Build Websites, By [Jon Duckett](https://learning.oreilly.com/search/?query=author%3A%22Jon%20Duckett%22&sort=relevance&highlight=true) Nov 2019

### [Learn Enough Python to be Dangerous: A Tutorial Introduction to Programming with Python](https://learning.oreilly.com/videos/-/9780138050764/)

By [Author:Michael Hartl](https://learning.oreilly.com/search?q=author:%22Michael%20Hartl%22), [Publisher:Addison-Wesley Professional](https://learning.oreilly.com/publisher/1495afeb-554e-4ba9-84fb-f260848d173e) • March 2023

### [Learn Enough HTML, CSS and Layout to Be Dangerous: An Introduction to Modern Website Creation and Templating Systems](https://learning.oreilly.com/library/view/-/9780137843183/) By [Author:Michael Hartl](https://learning.oreilly.com/search?q=author:%22Michael%20Hartl%22) and [Author:Lee Donahoe](https://learning.oreilly.com/search?q=author:%22Lee%20Donahoe%22)

[Publisher:Addison-Wesley Professional](https://learning.oreilly.com/publisher/1495afeb-554e-4ba9-84fb-f260848d173e) • August 2022

### [REST APIs with Flask and Python in 2023,](https://learning.oreilly.com/videos/-/9781788621526/) By [Author:Jose Salvatierra Fuentes](https://learning.oreilly.com/search?q=author:%22Jose%20Salvatierra%20Fuentes%22), [Publisher:Packt Publishing](https://learning.oreilly.com/publisher/dd0773f2-2d09-4827-9191-33477b14398c) • December 2022

[Writing a Web Application with Flask](https://learning.oreilly.com/videos/-/10000MNHV2021147/), By [Author:Doug Farrell](https://learning.oreilly.com/search?q=author:%22Doug%20Farrell%22), [Publisher:Manning Publications](https://learning.oreilly.com/publisher/ea8b4990-f3a7-4583-8839-606fdf1a8c20) • November 2021

## Essential SQLAlchemy, 2nd Edition, star rating fillstar rating fillstar rating fillstar rating fillstar rating outline8 reviews, By [Jason Myers](https://learning.oreilly.com/search/?query=author%3A%22Jason%20Myers%22&sort=relevance&highlight=true), [Rick Copeland](https://learning.oreilly.com/search/?query=author%3A%22Rick%20Copeland%22&sort=relevance&highlight=true)

### [The Complete HTML and CSS Course](https://learning.oreilly.com/videos/-/100000DIVC2022158/), **By [Author:Meta Brains](https://learning.oreilly.com/search?q=author:%22Meta%20Brains%22),** [**Publisher:Manning Publications**](https://learning.oreilly.com/publisher/ea8b4990-f3a7-4583-8839-606fdf1a8c20)**• September 2022**

### [HTML & CSS JumpStart](https://learning.oreilly.com/videos/-/9780137319633/), **By [Author:Andy Olsen](https://learning.oreilly.com/search?q=author:%22Andy%20Olsen%22),** [**Publisher:Pearson**](https://learning.oreilly.com/publisher/db6675ca-7122-4551-bff6-0a90688aa5ca)**• March 2021**

### [Full Stack Web Development Bootcamp with React and Python](https://learning.oreilly.com/videos/-/9781801811040/), **By [Author:Bogdan Stashchuk](https://learning.oreilly.com/search?q=author:%22Bogdan%20Stashchuk%22),** [**Publisher:Packt Publishing**](https://learning.oreilly.com/publisher/dd0773f2-2d09-4827-9191-33477b14398c)**• July 2021**