Test Strategy

1. Functionality Testing

This involves testing the basic functionality of the URL shortening service.

- **Positive Test Cases**: Test with valid URLs to ensure they are shortened correctly. Verify that the shortened URL redirects to the original URL when accessed.
- **Negative Test Cases**: Test with invalid URLs (e.g., “abcd”, “http//missingdotcom”) to ensure appropriate error messages are displayed.
- **Edge Cases**: Test with extremely long URLs, URLs with special characters, and URLs from different protocols (http, https, ftp) to ensure the service can handle a variety of inputs.

2. User Experience Testing

This involves testing the service from a user’s perspective.

- Ensure the website is user-friendly/mobile friendly in terms of ui scaling and intuitive. The process to shorten a URL should be straightforward and the shortened URL should be easy to copy.
- Check the response time when a URL is shortened. It should be quick to provide a good user experience.
- Test on various devices (desktop, mobile) and browsers (Chrome, Firefox, Safari) for compatibility.
- Minimal use of ads, proper search engine optimisation of the website can help too.
- Accessibility testing can be done for users with disabilities like color blindedness etc.
- Website should have dark and ui design trends like glassmorphism and bento grids can be employed to make the website look more modern.

3. Security Testing

This involves testing the service for potential security vulnerabilities.

- Ensure that the shortened URLs are secure and cannot be tampered with. The service should generate a unique key for each URL that is difficult to guess.
- Test for common web vulnerabilities such as SQL injection or hunting down crack websites whose links are shared on youtube.
- Tampermonkey chrome extension allows user to install custom scripts to bypass paywalls which should be hindered by modifying the code.

4. Performance Testing

This involves testing the service under high load.

- Test the system's behavior under normal and peak loads to ensure it can handle high traffic. This can be done using load testing tools that simulate multiple users accessing the service at the same time.
- Monitor the service's performance over time to identify any potential bottlenecks or performance issues.

5. Abuse Handling

This involves testing the service's ability to handle potential abuse.

- Implement rate limiting to prevent a single user from overloading the service with requests. This can help prevent denial-of-service attacks.
- Users should be given an option to purchase premium to bypass rate limits
- Monitor for any suspicious activities, such as a large number of requests from a single IP address or use of vpns like psiphon, nord or using fake temporary email ids.
- Also serial number verification, in case a premium is charge should be done strictly to prevent reuse of serial numbers by multiple users.

TEST CASES

Positive Test Cases

Test Case 1: URL shortening with valid HTTPS URL Steps:

1. Enter a valid HTTPS URL in the input field.
2. Click the "Shorten" button.
3. Verify that a shortened URL is generated.
4. Click on the shortened URL.
5. Verify that the user is redirected to the original HTTPS URL.

Expected Outcome: The user should be successfully redirected to the original URL.

Test Case 2: Custom short URL generation Steps:

1. Enter a valid URL and a desired custom short URL.
2. Click the "Shorten" button.

3. Verify that the desired custom short URL is generated.

Expected Outcome: The desired custom short URL should be generated and work correctly.

Test Case 3: Viewing URL information with "info." prefix Steps:

1. Prepend "info." to a shortened URL.
2. Visit the modified URL.
3. Verify that the URL information and metadata are displayed.

Expected Outcome: The URL information and metadata should be accurately displayed.

Negative Test Cases

Test Case 4: URL shortening with invalid URL Steps:

1. Enter an invalid URL in the input field.
2. Click the "Shorten" button.
3. Verify that an error message is displayed.

Expected Outcome: An informative error message should be displayed, indicating the invalid URL format.

Test Case 5: Non-existent shortened URL Steps:

1. Visit a randomly generated URL.
2. Verify that a 404 Not Found error message is displayed.

Expected Outcome: A 404 error message should be displayed, indicating that the shortened URL does not exist.

Test Case 6: XSS attack attempt Steps:

1. Enter a URL containing malicious JavaScript code.
2. Verify that the JavaScript code is not executed.

Expected Outcome: The malicious JavaScript code should not be executed, ensuring user safety.

Edge Cases

Test Case 7: URL shortening with long URLs Steps:

1. Enter a very long URL in the input field.

2. Verify that the URL is successfully shortened.
3. Verify that the shortened URL redirects to the original URL.

Expected Outcome: The service should handle long URLs effectively and redirect users correctly.

Test Case 8: URL shortening with special characters Steps:

1. Enter a URL containing special characters.
2. Verify that the URL is successfully shortened.
3. Verify that the shortened URL redirects to the original URL.

Expected Outcome: The service should handle special characters correctly and redirect users to the intended website.

Performance Test Cases

Test Case 9: High traffic load Steps:

1. Simulate a large number of concurrent users using load testing tools.
2. Monitor the service's performance metrics such as response times and throughput.
3. Verify that the service remains stable and responsive under high traffic.

Expected Outcome: The service should maintain acceptable performance under high traffic loads without significant delays or errors.

Test Case 10: URL shortening latency Steps:

1. Measure the time it takes to shorten a URL from user input to redirection.
2. Verify that the latency is within acceptable limits.

Expected Outcome: The URL shortening process should be fast and efficient, minimizing user wait times.

Automation Testing

Tools and Frameworks:

- Cypress for UI automation
- JMeter for performance testing
- Selenium for WebDriver integration
- Pytest or Jest for test execution

Automation Test Cases:

1. Positive Test Cases:

1. Shorten Valid URL:
    - Input a valid URL and ensure a shortened URL is generated.
    - Assert the shortened URL redirects to the original URL.
2. Access Shortened URL:
    - Use the shortened URL and verify redirection to the original URL.

2. Negative Test Cases:

3. Shorten Invalid URL:
    - Input an invalid URL and validate the service's response.
4. Exceed URL Length Limit:
    - Attempt to shorten a URL exceeding the maximum length and verify system response.

3. Edge Cases:

5. Test with Extremely Long URL:
    - Input a very long URL and check if the service successfully shortens it.
6. Special Character URL:
    - Test with a URL containing special characters and verify successful shortening.

4. Performance Test Cases:

7. Load Testing:
    - Simulate multiple concurrent requests and measure response times.
8. Stress Testing:
    - Push the service beyond its limits with excessive requests and monitor behavior.

Automation Tools/Frameworks and Maintenance:

Tools/Frameworks:

- Selenium: For UI testing.
- JMeter: For performance and load testing.
- OWASP ZAP: For security testing.
- Python/Java + TestNG/JUnit: For writing automation test scripts.

Maintenance Considerations:

- Regular Updates: Keep automation scripts updated with changes in the application.
- Version Control: Use version control systems (e.g., Git) to manage and track changes in test scripts.
- Code Reviews: Regularly review and refactor test scripts to maintain quality.
- Continuous Integration: Integrate tests into CI/CD pipelines for automated testing on each build.
- Logging and Reporting: Implement logging mechanisms and generate detailed test reports for analysis.