Industrial Internship Report on

”PyShortLink - Python URL Shortener”

Prepared by

Arom Dhabe

Executive Summary
This report provides details of the Industrial Internship provided by upskill Campus and The IoT Academy in collaboration with Industrial Partner UniConverge Technologies Pvt Ltd (UCT). This internship was focused on a project/problem statement provided by UCT. We had to finish the project including the report in 6 weeks’ time. My project was to develop a user-friendly web application that allows users to convert long and unwieldy URLs into shortened and more manageable links. The platform will offer an efficient and secure solution for individuals and businesses to share links on social media, messaging apps, and other digital platforms with character limitations. This internship gave me a very good opportunity to get exposure to Industrial problems and design/implement solution for that. It was an overall great experience to have this internship.

TABLE OF CONTENTS

1 Preface

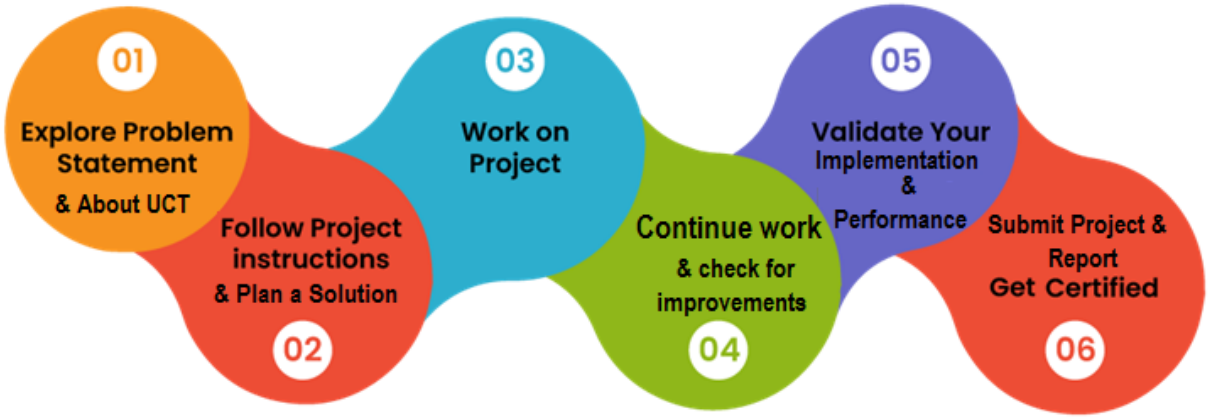
Summary of the whole 6 weeks' work.

About need of relevant Internship in career development.

Brief about Your project/problem statement.

Opportunity given by USC/UCT.

How Program was planned



Your Learnings and overall experience.

Thank to all (with names), who have helped you directly or indirectly.

Your message to your juniors and peers.

2 Introduction

2.1 About UniConverge Technologies Pvt Ltd

A company established in 2013 and working in Digital Transformation domain and providing Industrial solutions with prime focus on sustainability and RoI.

For developing its products and solutions it is leveraging various **Cutting Edge Technologies e.g. Internet of Things (IoT), Cyber Security, Cloud computing (AWS, Azure), Machine Learning, Communication Technologies (4G/5G/LoRaWAN), Java Full Stack, Python, Front end** etc.

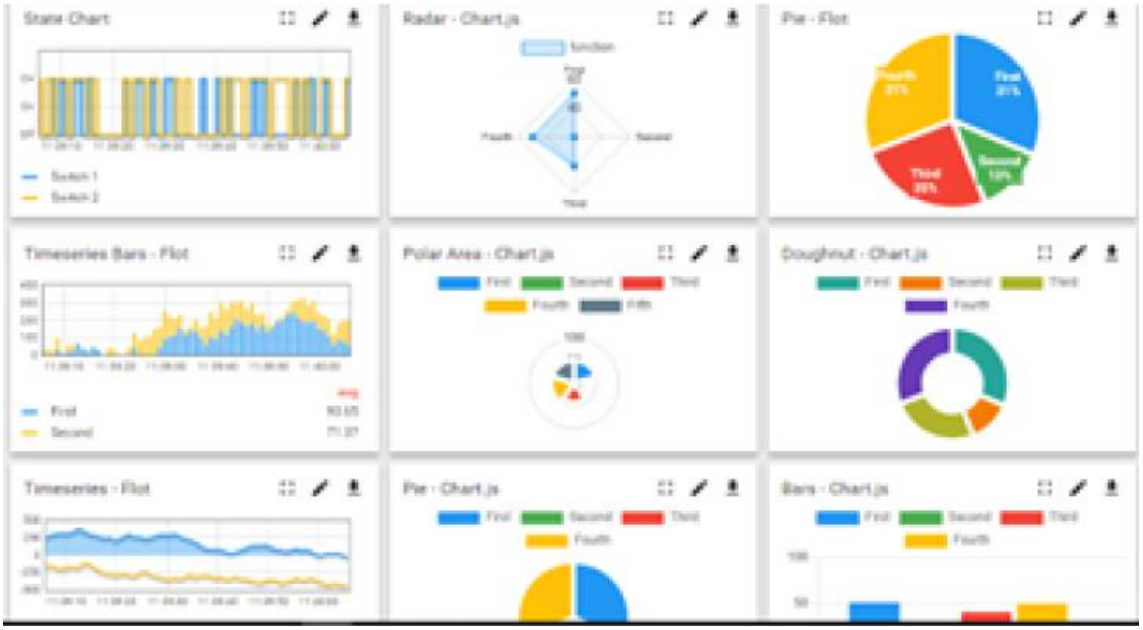
i. UCT IoT Platform (UCT Insight)

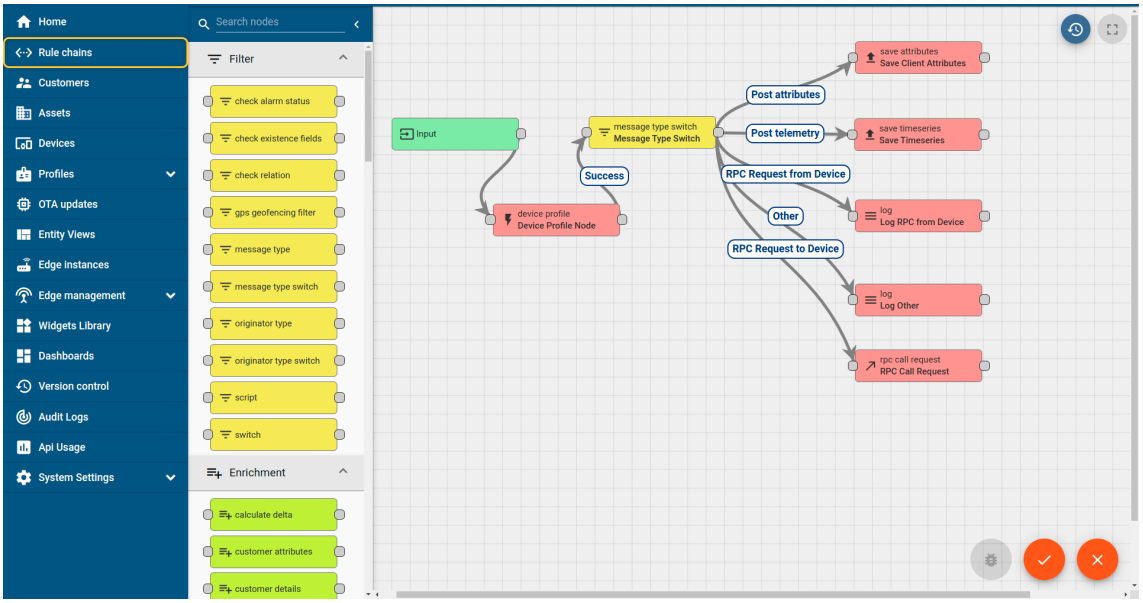
UCT Insight is an IOT platform designed for quick deployment of IOT applications on the same time providing valuable "insight" for your process/business. It has been built in Java for backend and ReactJS for Front end. It has support for MySQL and various NoSql Databases.

- It enables device connectivity via industry standard IoT protocols - MQTT, CoAP, HTTP, Modbus TCP, OPC UA
- It supports both cloud and on-premises deployments.

It has features to

- Build Your own dashboard
- Analytics and Reporting
- Alert and Notification
- Integration with third party application(Power BI, SAP, ERP)
- Rule Engine





ii. Smart Factory Platform (FACTORY WATCH)

Factory watch is a platform for smart factory needs.

It provides Users/ Factory

- with a scalable solution for their Production and asset monitoring
- OEE and predictive maintenance solution scaling up to digital twin for your assets.
- to unleased the true potential of the data that their machines are generating and helps to identify the KPIs and also improve them.
- A modular architecture that allows users to choose the service that they what to start and then can scale to more complex solutions as per their demands.

Its unique SaaS model helps users to save time, cost and money.

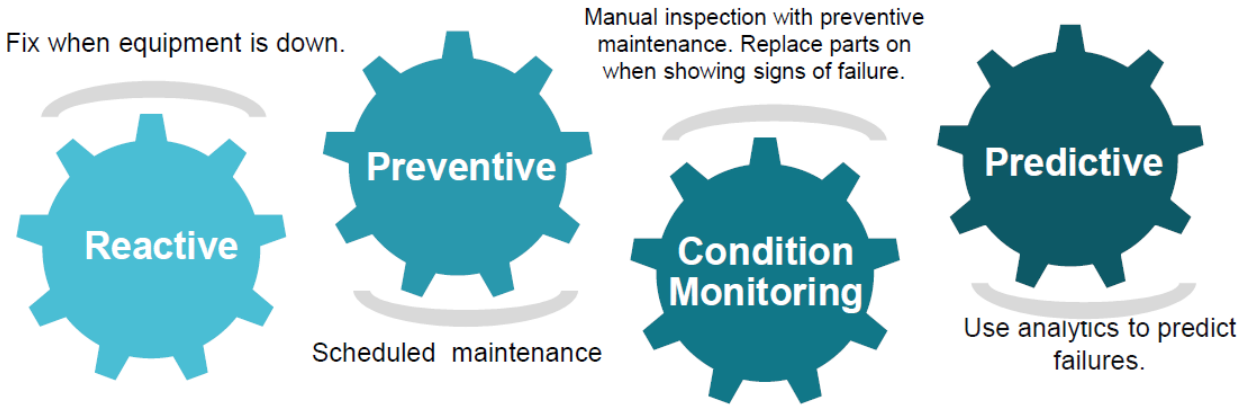
Machine	Operator	Work Order ID	Job ID	Job Performance	Job Progress		Output		Rejection	Time (mins)				Job Status	End Customer
					Start Time	End Time	Planned	Actual		Setup	Pred	Downtime	Idle		
CNC_S7_81	Operator 1	WO0405200001	4168	58%	10:30 AM		55	41	0	80	215	0	45	In Progress	i
CNC_S7_81	Operator 1	WO0405200001	4168	58%	10:30 AM		55	41	0	80	215	0	45	In Progress	i

iii. LoRaWAN based Solution

UCT is one of the early adopters of LoRAWAN teschnology and providing solution in Agritech, Smart cities, Industrial Monitoring, Smart Street Light, Smart Water/ Gas/ Electricity metering solutions etc.

iv. Predictive Maintenance

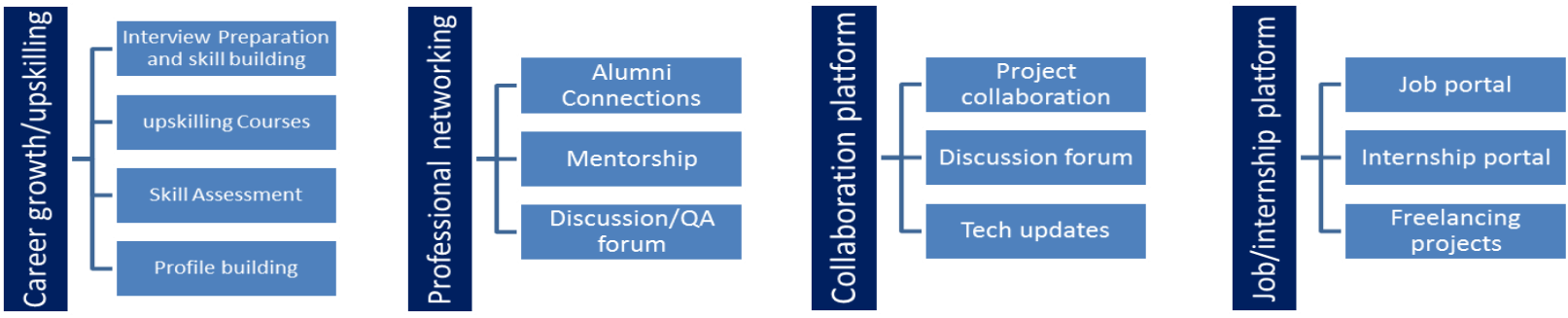
UCT is providing Industrial Machine health monitoring and Predictive maintenance solution leveraging Embedded system, Industrial IoT and Machine Learning Technologies by finding Remaining useful life time of various Machines used in production process.



2.2 About upskill Campus (USC)

upskill Campus along with The IoT Academy and in association with Uniconverge technologies has facilitated the smooth execution of the complete internship process.

USC is a career development platform that delivers **personalized executive coaching** in a more affordable, scalable and measurable way.



2.3 The IoT Academy

The IoT academy is EdTech Division of UCT that is running long executive certification programs in collaboration with EICT Academy, IITK, IITR and IITG in multiple domains.

2.4 Objectives of this Internship program

The objective for this internship program was to

- ☛ get practical experience of working in the industry.
- ☛ to solve real world problems.
- ☛ to have improved job prospects.
- ☛ to have Improved understanding of our field and its applications.
- ☛ to have Personal growth like better communication and problem solving.

2.5 Reference

[1] Python Documentation : https://docs.python.org/3/

[2] TinyURL : http://www.tinyurl.com/

[3] Popular Services: Bitly, TinyURL, and Ow.ly are examples of well-known URL shortener services that offer features like link customization, tracking, and analytics to help users manage their shortened links effectively.

2.6 Glossary

Terms	Acronym
URL	Uniform Resource Locator
DNS	Domain Name System
IP	Internet Protocol
HTTP	Hypertext Transfer Protocol
HTTPS	HTTP Secure

3 Problem Statement

In the assigned problem statement

1. Design a simple URL shortener application using Python and Tkinter: This sets the context for the project, indicating that the goal is to create a software application (URL shortener) using the Python programming language with the help of the Tkinter library to build the graphical user interface.
2. The application should provide a graphical user interface (GUI) that allows users to enter a long URL and receive a shortened URL as output using the TinyURL service: The primary functionality of the application is to allow users to interact with the GUI. Users should have the ability to input a long URL into a text entry field provided by the GUI. When they click a specific button ("Click Me"), the application will process the input URL and generate a shortened URL using the TinyURL service. The shortened URL should then be displayed on the GUI.
3. Requirements: This section lists specific criteria and features that the URL shortener application must fulfill:
    - The application should be built using Python and Tkinter for the GUI: This sets the programming language and library requirements for the project.
    - Users should be able to enter a long URL into a text entry field: It specifies the input mechanism for users.
    - The program should generate the shortened URL using the TinyURL service: The choice of the TinyURL service as the URL shortening method is given.
    - Display the shortened URL in a separate text entry field: It defines how the output should be presented to the user.
    - The GUI should have an aesthetically pleasing design with appropriate font styles and colors: This requirement emphasizes the need for an attractive and user-friendly interface.
    - The application should work offline once the GUI is loaded, but an internet connection is required to generate the shortened URL: The application should be functional without an internet connection for GUI operations, but internet connectivity is necessary for actual URL shortening using TinyURL.
4. Note: This section provides additional information or clarifications related to the problem statement. It mentions that the **pyshorteners** library is used in this implementation for simplicity, but other URL shortener APIs or custom URL shortening services can also be explored.

In summary, the problem statement provides a clear understanding of what needs to be accomplished—the creation of a Python-based URL shortener application with a user-friendly GUI using the Tkinter library, enabling users to input long URLs and receive shortened URLs through the TinyURL service.

4 Existing and Proposed solution

Provide summary of existing solutions provided by others, what are their limitations?

4.1. Shorte.st:

- Features: Shorte.st is a URL shortener that also offers earning opportunities through link sharing.
- Limitations: Some users might find the inclusion of advertisements and the requirement to view ads before reaching the destination URL inconvenient.

4.2. is.gd:

- Features: is.gd is a simple and fast URL shortener service with a straightforward interface.
- Limitations: The service lacks advanced link management features and analytics. Additionally, link customization options are limited compared to other solutions.

What is your proposed solution?

1. Dynamic Short Links: The AI-powered URL shortener can support dynamic short links that adapt their destination based on specific parameters or user attributes, providing personalized link experiences.
2. Offline Functionality: The solution might have limited offline shortening capabilities, allowing users to create shortened links even without internet connectivity and syncing data when they regain access.
3. Link Preview: Users can preview the destination URL before clicking on shortened links to ensure transparency and mitigate the risk of clicking on malicious links.

What value addition are you planning?

1. Increased Knowledge Base: Expanding the model's training data with up-to-date information allows it to stay relevant and provide more comprehensive and accurate information on various topics.
2. Real-Time Learning: Implementing mechanisms for AI models to learn from new data and user interactions in real-time helps them adapt and improve their responses over time.
3. User Interface Enhancements: Integrating AI language models into user-friendly interfaces enhances the user experience and makes AI technology more approachable for users.
4. Accessibility Features: Implementing accessibility features ensures that AI language models can be used by people with diverse abilities and needs.

4.1 Code submission (Github link) : https://github.com/DevArom/upskillcampus

4.2 Report submission (Github link) :

https://drive.google.com/drive/u/0/folders/1d9JK7UT8ax33ulXU9eQYuVsTXsTZ-ND1

5 Proposed Design/ Model

Given more details about design flow of your solution. This is applicable for all domains. DS/ML Students can cover it after they have their algorithm implementation. There is always a start, intermediate stages and then final outcome.

5.1 Interfaces (if applicable)

Update with Block Diagrams, Data flow, protocols, FLOW Charts, State Machines, Memory Buffer Management.

6 Performance Test

This is very important part and defines why this work is meant of Real industries, instead of being just academic project.

Here we need to first find the constraints.

How those constraints were taken care in your design?

How those constraints were taken care in your design?

Constraints can be e.g. memory, MIPS (speed, operations per second), accuracy, durability, power consumption etc.

In case you could not test them, but still you should mention how identified constraints can impact your design, and what are recommendations to handle them.

1. Length Limitation: In the design, a function could be implemented to validate the length of the input URL before attempting to shorten it using the TinyURL service. If the URL exceeds the supported length, the application could display an error message to inform the user about the limitation and provide guidance on what to do next.
2. TinyURL Service Dependency: The application could handle the TinyURL service dependency by checking the availability of the service before attempting to shorten URLs. If the service is unavailable, the application could provide an informative message to the user, suggesting the use of a different URL shortener or trying again later.
3. Internet Connectivity: The application could include a check for internet connectivity before offering the URL shortening feature. If an internet connection is not available, the application could disable the "Click Me" button and display a message informing the user that internet access is required for URL shortening.
4. User Input Validation: The design could include robust input validation to ensure that users enter valid URLs. If an invalid URL is entered, the application could display an error message, guiding the user to enter a properly formatted URL.
5. Error Handling: Proper error handling could be implemented throughout the application to catch and manage various types of errors, including errors related to the TinyURL service or unexpected network issues. User-friendly error messages could be displayed to inform users about any problems encountered during URL shortening.
6. Performance and Scalability: To address performance and scalability concerns, the application design could consider load testing the URL shortening service and optimizing the code to handle multiple requests simultaneously without significant delays.
7. Security: The design could include security measures to prevent abuse and misuse of the URL shortener. This might include rate-limiting requests from a single IP address, implementing CAPTCHA for suspicious activities, and providing a warning message to users about the potential risks of clicking on shortened links from unknown sources.
8. Third-Party Library Reliability: The design could regularly check for updates to the pyshorteners library and ensure that it remains compatible with the underlying URL

shortening service. Additionally, a contingency plan could be prepared in case the library becomes unsupported or experiences issues.

9. Legal and Ethical Considerations: The application design could include a privacy policy and terms of service to inform users about data handling practices and any legal obligations. The design should also comply with relevant copyright and content ownership laws.
10. Platform Compatibility: To ensure platform compatibility, the design could be tested on different operating systems and Python versions. Any platform-specific issues could be addressed during the development and testing phase.

Remember that the specific implementation of these considerations may vary based on the application's requirements and the developer's choices. The design should prioritize user experience, security, and reliability to create a successful URL shortener application.

6.1 Test Plan/ Test Cases

Test Cases:

6.1.1 Test Case: Valid URL Shortening

- Description: Enter a valid long URL and click the "Click Me" button.
- Expected Result: The application should generate a shortened URL using the TinyURL service and display it in the output field.

6.1.2 Test Case: Invalid URL Input

- Description: Enter an invalid or malformed URL and click the "Click Me" button.
- Expected Result: The application should display an error message indicating that the URL is invalid.

6.1.3 Test Case: URL Length Limitation

- Description: Enter a very long URL that exceeds the supported length for the URL shortening service and click the "Click Me" button.
- Expected Result: The application should display an error message indicating that the URL is too long to be shortened.

6.1.4 Test Case: Internet Connectivity

- Description: Disable the internet connection and attempt to shorten a URL.
- Expected Result: The application should display a message indicating that an internet connection is required for URL shortening.

6.1.5 Test Case: Error Handling

- Description: Simulate a server error from the TinyURL service by providing a non-functional URL shortening API endpoint.
- Expected Result: The application should display an informative error message indicating that there was an issue with the URL shortening service.

6.1.6 Test Case: Security Measures

- Description: Enter a high number of URL shortening requests in a short period from the same IP address.
- Expected Result: The application should limit the number of requests and display a CAPTCHA or similar challenge to prevent abuse.

6.2 Test Procedure

1. Test Setup: Ensure that the URL shortener application is installed and running on the test environment with Python and Tkinter properly set up.
2. Test Environment: Verify that the test environment meets the specified requirements, including the operating system and Python version.
3. Input Validation Tests:
   3.1. Valid URL Shortening:
      - Enter a valid long URL in the input field.
      - Click the "Click Me" button.
      - Verify that the application generates a shortened URL using the TinyURL service.
      - Check that the shortened URL is displayed in the output field.
   3.2. Invalid URL Input:
      - Enter an invalid or malformed URL in the input field.
      - Click the "Click Me" button.
      - Verify that the application displays an error message indicating that the URL is invalid.
      - Ensure that the output field remains empty.
   3.3. URL Length Limitation:
      - Enter a very long URL that exceeds the supported length for the URL shortening service in the input field.
      - Click the "Click Me" button.
      - Verify that the application displays an error message indicating that the URL is too long to be shortened.
      - Ensure that the output field remains empty
4. Internet Connectivity Tests:
   4.1. Online URL Shortening:
      - Enable internet connectivity on the test machine.
      - Enter a valid long URL in the input field.

- Click the "Click Me" button.
- Verify that the application generates a shortened URL using the TinyURL service and displays it in the output field.

4.2. Offline URL Shortening:

- Disable the internet connection on the test machine.
- Enter a valid long URL in the input field.
- Click the "Click Me" button.
- Verify that the application displays a message indicating that an internet connection is required for URL shortening.
- Ensure that the output field remains empty.

5. Error Handling Tests:

5.1. Server Error Handling:

- Simulate a server error from the TinyURL service by providing a non-functional URL shortening API endpoint.
- Enter a valid long URL in the input field.
- Click the "Click Me" button.
- Verify that the application displays an informative error message indicating the issue with the URL shortening service.
- Ensure that the output field remains empty.

6. Security Measures Test:

6.1. Abuse Prevention:

- Enter a high number of URL shortening requests in a short period from the same IP address.
- Verify that the application limits the number of requests and displays a CAPTCHA or similar challenge to prevent abuse.

7. Usability Tests:

7.1. GUI Usability:

- Check the GUI layout, font styles, and colors for an aesthetically pleasing design.
- Verify that the input and output fields are appropriately labeled and visually distinct.
- Ensure that buttons are responsive and provide visual feedback on click.

8. Performance Test:
   - Perform load testing by entering multiple valid long URLs and clicking the "Click Me" button repeatedly.
   - Verify that the application handles multiple requests simultaneously without significant delays.
9. Test Reporting: Record the test results, including pass/fail status and any issues encountered during testing.
10. Test Completion: Complete the test procedure when all test cases have been executed, and the application meets the specified acceptance criteria

6.3 Performance Outcome

The performance outcome of the URL shortener application refers to how well it performs in terms of responsiveness, speed, and resource usage during various scenarios and under different loads. The performance outcome is typically measured against predefined performance criteria to determine if the application meets the expected performance standards.

Performance outcomes can be assessed through various metrics and tests, such as:

1. Responsiveness: This measures how quickly the application responds to user interactions. A responsive application should provide immediate feedback to users upon clicking buttons or entering URLs.

2. URL Shortening Speed: This metric evaluates how quickly the application generates the shortened URL after the user clicks the "Click Me" button. Faster URL shortening times are desirable to ensure a smooth user experience.

3. Internet Connectivity Handling: The performance outcome should consider how well the application handles cases where there is no internet connection or when the TinyURL service is temporarily unavailable. The application should provide appropriate messages to users, indicating the need for an internet connection and informing them of any service unavailability.

4. Error Handling: The application's performance outcome should be evaluated based on how well it handles various error scenarios, such as invalid URLs, server errors, or issues with the URL shortening service. Proper error messages and handling contribute to a better user experience.

5. Load Testing: Load testing involves subjecting the application to a simulated workload to assess its performance under various user loads. It can help identify potential performance bottlenecks, such as slow response times or resource exhaustion.

6. Resource Usage: Monitoring the application's resource usage, such as CPU, memory, and network, can reveal how efficiently it utilizes system resources during URL shortening operations. Efficient resource usage contributes to the application's overall performance.

The performance outcome should align with the application's defined performance objectives and acceptance criteria. It is crucial to validate the performance against expected values and to address any performance issues that arise during testing.

Ultimately, a positive performance outcome indicates that the URL shortener application is capable of delivering a smooth and efficient user experience, handling various scenarios gracefully, and meeting its performance expectations. However, if performance issues are identified, they should be addressed through optimization, code improvements, or by selecting alternative components that better meet the performance requirements.

7 My learnings

My training allows me to provide coherent responses to a wide range of queries. I can assist with research, answer questions, and support decision-making. However, I am not conscious and have no personal goals. My purpose is to be a helpful tool for providing information and assisting users in a responsible manner.

7.1. Knowledge: The overall internship was self-learning and I like that means if we've some query we've to solve on ourselves i.e. problem solving. This will keep us motivated and dedicated to our project.

7.2. Research: We have to do deep research and quality research about that particular project that solidify our problem solving, error handling.

7.3. Creative Support: In creative field, we've to do creative work that user will come and test our project in their environment.

7.4. Decision Support: In strategic decision-making processes, we can assist by analyzing data, providing insights, and offering potential outcomes, supporting better-informed decisions for business growth.

8 Future work scope

From my side future scope of URL shorteners is likely to evolve to meet the changing needs of users and advancements in technology. Here are some potential areas of development and improvements for URL shorteners in the future if I want to do that in future:

1. Link Management and Tracking: Future URL shorteners may provide more robust link management features, allowing users to organize, categorize, and track their shortened links more efficiently. Advanced analytics and reporting tools could offer deeper insights into link performance.
2. Security Enhancements: To address security concerns, future URL shorteners might implement measures to detect and prevent the creation of short links to malicious or phishing websites. This could include link scanning and validation before shortening.
3. Customization and Branding: The scope for customization could expand, allowing users to have more control over the appearance of their shortened links, including custom slugs, branded domains, and personalized link previews.
4. Dynamic and Conditional Short Links: Future URL shorteners might support dynamic and conditional short links that change their destination based on specific conditions, user attributes, or geolocation, providing personalized experiences.

5. Multi-Platform Integration: URL shorteners may integrate with various platforms and applications to enable seamless sharing of shortened links across different devices and channels, including mobile apps and smart devices.

6. Offline Functionality: Future URL shorteners could explore limited offline shortening capabilities, allowing users to create shortened links even without internet connectivity and syncing data when they regain access.

7. Blockchain-Based Shortening: The use of blockchain technology could offer enhanced transparency, traceability, and security for shortened links, ensuring the integrity of the link history.

8. URL Expanding Services: To counter the issue of link rot and broken links, URL shorteners might offer companion services that allow users to expand and restore shortened links even if the shortening service becomes unavailable.

9. Integration with Voice Assistants: Future URL shorteners may integrate with voice-activated devices and virtual assistants, enabling users to create and share shortened links using voice commands.

10. Emphasis on Privacy: URL shorteners could prioritize user privacy, offering features that allow users to control the data collected through link clicks and providing options for anonymous link shortening.

11. Social Media Integration: To streamline social media sharing, URL shorteners might offer direct integration with popular social media platforms, making link shortening and sharing more convenient.

12. Multilingual Support: Expanding language support could enable users to create shortened links in various languages, catering to diverse global audiences.

The future of URL shorteners will depend on user preferences, technological advancements, and the industry's response to security and privacy concerns. As with any technology, responsible development and thoughtful consideration of the impact on users will be essential in shaping the future of URL shorteners.