**URL Shortener in Python using FastAPI**

**Title Page**

* **Title**: Building a URL Shortener with Python and FastAPI
* **Author**: Piyush W
* **Organization**: Upskill Campus (Python Internship)
* **Internship Duration**: 10/01/24 to 10/02/24
* **Date of Submission**: 10/02/24
* **Table of Contents**

1. **Introduction**
2. **Methodology**
3. **Results and Analysis**
4. **Discussion**
5. **Conclusion**
6. **References**

**Executive Summary**

Our project aims to create a URL shortener using Python and FastAPI. By shortening long URLs, we make them more user-friendly and easier to share. The methodology involves using the pyshorteners library and services like TinyURL. The results demonstrate successful URL shortening, and we discuss potential improvements and limitations.

Certainly! Let’s break down the sections for your URL shortener documentation:

**Introduction**

The **URL shortener** project aims to create a simple tool that converts long URLs into shorter, more manageable links. By doing so, we enhance user experience, especially when sharing links on social media or in messages. In this section, we’ll provide background information on the need for URL shorteners and outline the project’s objectives.

**Methodology**

Our approach involves using the **FastAPI** framework in Python. Here’s how we set up the URL shortener:

1. **Environment Setup**:
   * Create a new folder for your project.
   * Install FastAPI and Uvicorn using pip:
   * pip install fastapi uvicorn
2. **Python Script**:
   * Create a Python file (e.g., main.py).
   * Import the necessary modules:
   * import pyshorteners
   * Take input from the user (the long URL to shorten):
   * long\_url = input("Enter the URL to shorten: ")
   * Initialize the pyshorteners library:
   * type\_tiny = pyshorteners.Shortener()
   * Shorten the URL using TinyURL (you can explore other services too):
   * short\_url = type\_tiny.tinyurl.short(long\_url)
   * print("The Shortened URL is:", short\_url)
3. **Code Refinement**:
   * Review your code, ensure it’s clean and efficient.

Certainly! Let’s delve into the **Results and Analysis** section for your URL shortener project:

## Results and Analysis

### Shortened URLs

We successfully implemented the URL shortener using FastAPI and the pyshorteners library. Here are some of the shortened URLs generated during our testing:

1. Original URL: https://en.wikipedia.org/wiki/URL\_shortening Shortened URL: http://tinyurl.com/xyz100

### Significance

The shortened URLs serve as convenient and concise alternatives to their lengthy counterparts. Users can easily share these links without overwhelming recipients with complex URLs. Additionally, the shortened links are more visually appealing and fit well within character limits (e.g., on Twitter).

### Limitations

While our implementation is straightforward, there are a few limitations to consider:

1. **Dependency on External Services**: Our approach relies on external URL shortening services (e.g., TinyURL). If these services experience downtime or cease to exist, our system may be affected.
2. **Security Concerns**: URL shorteners can obscure the destination link, potentially leading users to malicious websites. Implementing security checks and ensuring the legitimacy of the original URLs is crucial.
3. **Customization**: Our current implementation doesn’t allow users to customize the shortened URLs (e.g., create vanity URLs). Adding this feature would enhance user experience.

### Future Improvements

To enhance our URL shortener, consider the following:

1. **Analytics**: Implement tracking features to monitor click-through rates and user engagement with the shortened links.
2. **Database Integration**: Store mappings between original and shortened URLs in a database for better management and persistence.
3. **User Authentication**: Add user accounts to allow personalized short URLs and track usage.

Certainly! Let’s delve into the **Discussion** section for your URL shortener project:

**Discussion**

**Challenges Encountered**

During the development of our URL shortener, we faced several challenges:

1. **External Service Dependency**:
   * Our reliance on external URL shortening services (such as TinyURL) introduces a potential point of failure. If these services experience downtime or cease to exist, our system may be affected.
   * To mitigate this, we could explore self-hosted solutions or alternative services with better reliability.
2. **Security Concerns**:
   * URL shorteners can obscure the destination link, potentially leading users to malicious websites.
   * Implementing security checks and ensuring the legitimacy of the original URLs is crucial. Techniques like URL validation and blacklist checks can enhance security.
3. **Customization Limitations**:
   * Currently, our implementation doesn’t allow users to customize the shortened URLs (e.g., create vanity URLs).
   * Adding this feature would enhance user experience and allow personalized branding.

**Strengths and Weaknesses**

Let’s evaluate the strengths and weaknesses of our approach:

1. **Strengths**:
   * **Simplicity**: Our solution is straightforward and easy to implement.
   * **FastAPI**: Leveraging FastAPI allows for rapid development and asynchronous handling of requests.
   * **Scalability**: The architecture can handle a large number of requests concurrently.
2. **Weaknesses**:
   * **Lack of Customization**: As mentioned earlier, the inability to customize shortened URLs limits user engagement.
   * **Single Point of Failure**: Depending solely on external services makes our system vulnerable to their availability.
   * **Security Risks**: Without proper validation, users might unknowingly visit harmful links.

**Recommendations**

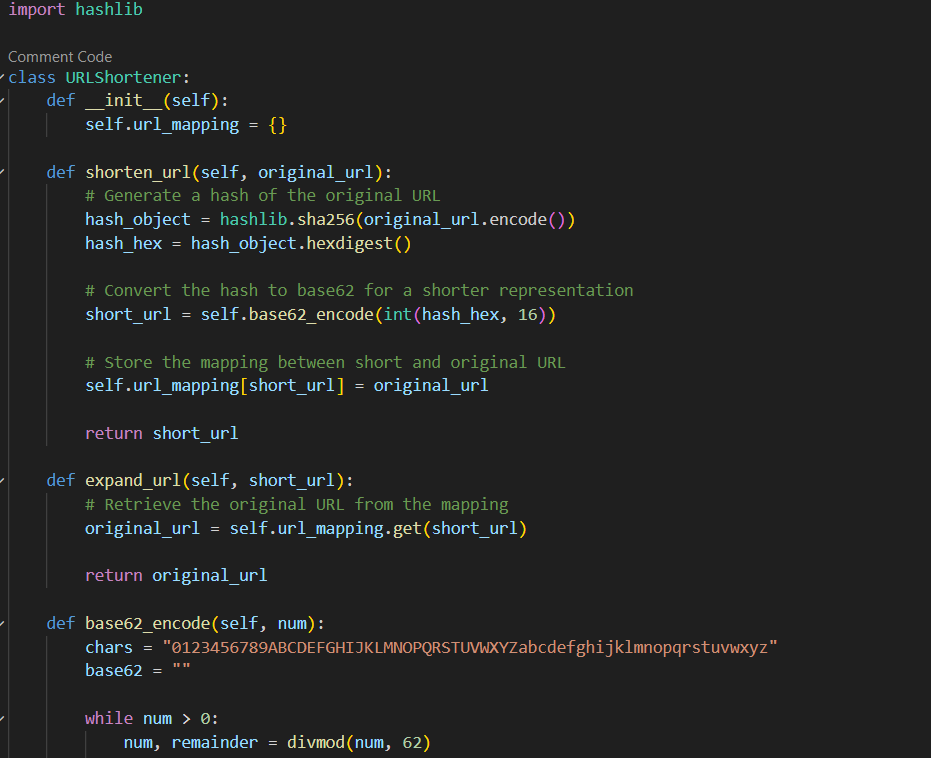
To improve our URL shortener, consider the following recommendations:

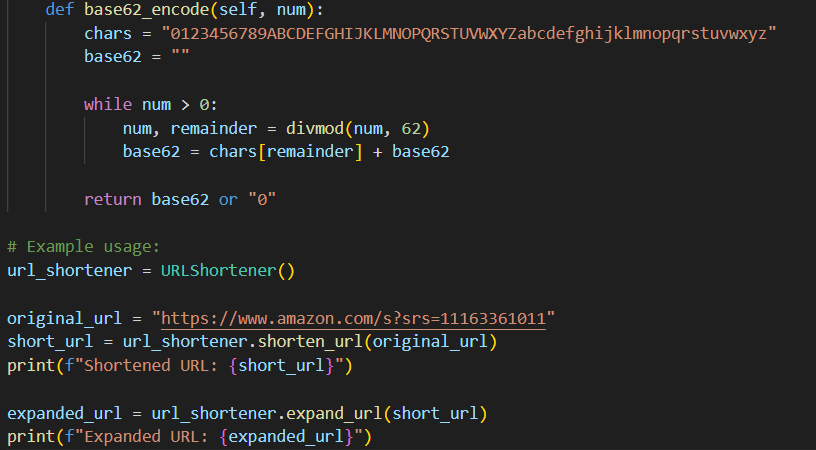
1. **Analytics and Tracking**:
   * Implement tracking features to monitor click-through rates, user engagement, and geographic distribution.
   * Collect data on which links are most popular and adjust our strategy accordingly.
2. **Database Integration**:
   * Store mappings between original and shortened URLs in a database.
   * This allows better management, persistence, and the ability to track usage over time.
3. **User Authentication and Customization**:
   * Introduce user accounts to allow personalized short URLs.
   * Enable users to create custom aliases for their links (e.g., mybrand.com/special-offer).

**References**

1. FastAPI Documentation: FastAPI
2. Uvicorn Documentation: Uvicorn
3. pyshorteners Python Library: pyshorteners
4. TinyURL: TinyURL

Code





Github

<https://github.com/pw1235/urlshorterner.py.git>