**Shortened URL Validator**Made By: Apon Das

Table Of Contents

[1. Project Overview**:** 3](#_Toc203123765)

[2. Functional Requirements 4](#_Toc203123766)

[3. Non – functional Requirements 5](#_Toc203123767)

[4. Technology Stack: 6](#_Toc203123768)

[5. Folder Structure & Files: 7](#_Toc203123769)

[6. File – by – file explanation: 7](#_Toc203123770)

[7. Error Handling: 16](#_Toc203123771)

[8. Security Considerations: 18](#_Toc203123772)

[9. Future Enhancements: 19](#_Toc203123773)

[10. Testing Strategy 20](#_Toc203123774)

[11. Flowchart Diagrams: 22](#_Toc203123775)

# 1. Project Overview**:**

I am working on creating a shortened URL Validator, which is a backend service designed to validate short URLs / URLs which have been shortened, without users needing to click or visit them directly.  
  
 **How it works:**

* It expands the short URLs
* It tracks redirect chains
* Checks against blacklists
* Evaluates domain reputation
* Caches validated URLs for faster future checkups

This application will help users assess how safe and legitimate a short URL is likely to be, before they visit them manually.

# 2. Functional Requirements

* Accepting user input – It receives shortened URL via JSON POST request
* Expand short URL – Uses ‘requests’ to follow redirects
* Track redirect chain – Shows all the destination hops from the short link to the final destination
* Domain reputation check – Checks if the final domain is trusted using WHOIS lookup
* Blacklist check – Compares the final URL against known malicious domains
* Confidence scoring – Returns a basic score based on trust and blacklist status
* Cache results – Stores validated URLs in a local JSON file (urls.json)
* Return clean JSON – Outputs structured, easy – to – read response

# 3. Non – functional Requirements

* Secure data handling – No sensitive data stored
* Using minimal dependencies (Flask, WHOIS, Python)
* Easy to run locally
* Handles timeouts, DNS errors, malformed inputs
* New features can be added

# 4. Technology Stack:

|  |  |
| --- | --- |
| **Layer** | **Technology Used** |
| Backend | Flask (Python web framework) |
| HTTP Requests | ‘requests’ library |
| Domain Reputation | ‘python – whois’ |
| Datastore | Local JSON file (urls.json’) |
| Deployment Ready | Can be hosted on cloud platforms |
| Testing Tools | Powershell |
| Hosting Platform | Github (private repo), deployable via Render or Heroku |

# 5. Folder Structure & Files:

**A screenshot of a computer program

AI-generated content may be incorrect.**

# 6. File – by – file explanation:

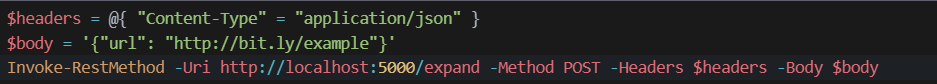
* **“production\_ready\_app.py”**

This is the main Flask application that runs the Validator service

**Key Features:**

* Accepts POST requests at “/expand” endpoint
* Expands shortened URLs and follows redirect chain
* Checks final URL against blacklist
* Evaluates domain reputation using WHOIS
* Assigns a confidence score (100 = safe, 0 = unsafe)
* Saves results in ‘urls.json’ to avoid re – checking the same URL

**Example Request:**

****

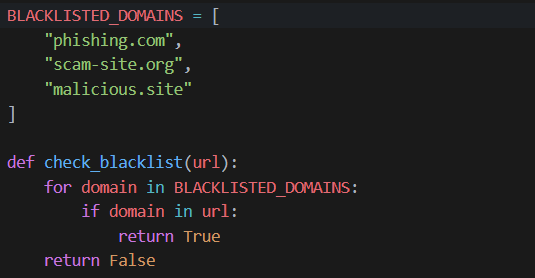
**Sample Response:**

**A screen shot of a computer code

AI-generated content may be incorrect.**

* **utils/blacklist.py**

Maintains a list of **known phishing / malware domains  
  
Code Snippet:**

****

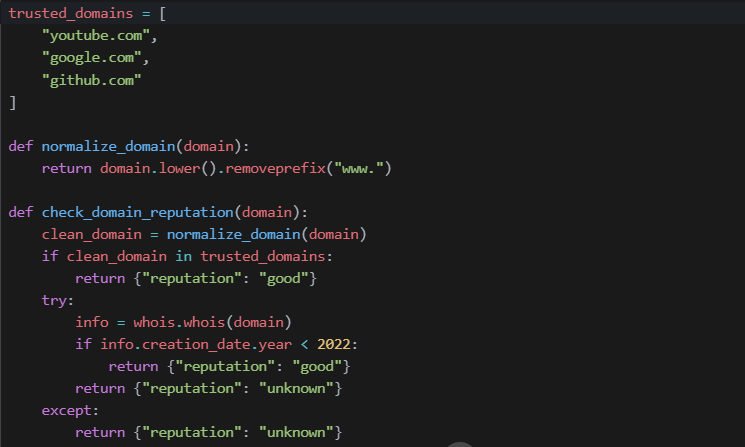
Checks if any blacklisted domain appears in the final URL.

* utils/reputation.py

This evaluates whether the final domain is **trusted**, based on:

* A list of well – known good companies (like YouTube, Google, GitHub)
* WHOIS lookup to check domain age and ownership

Code Snippet:



If the domain is not in the trusted list, it checks the WHOIS creation date as an extra safety measure.

* **data/urls.json**

Stores the cached results of previously validated URLs.

**Why it’s useful:**

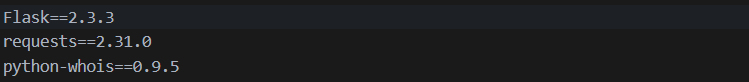
* Prevents repeated expansion of the same short URL
* Improves performance
* Reduces external API calls

Example content:



* **“requirements.txt”**

Lists all required Python libraries:



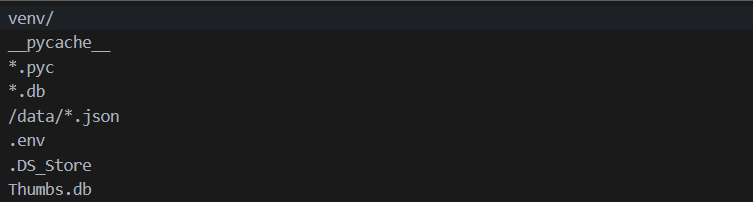
Install with:



* **“gitignore”**

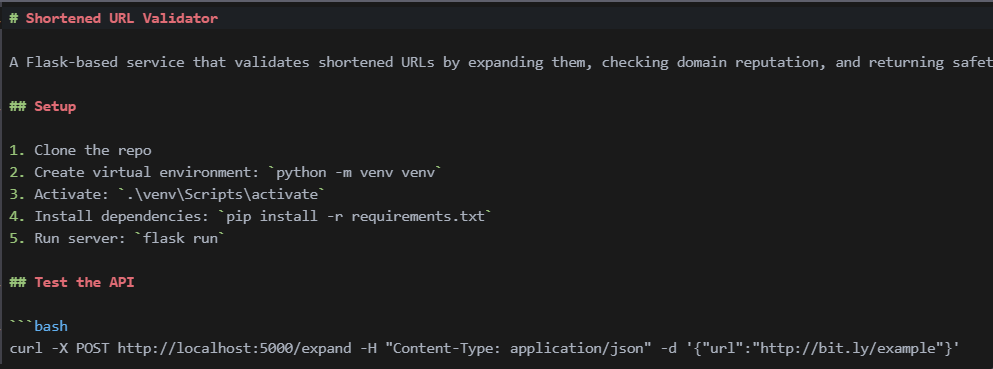
Tells Git which files not to upload to GitHub.

Sample contents:



* README.md

A basic “README.md” should include:



A black screen with white text

AI-generated content may be incorrect.

2. Create Virtual Environment:



3. Install Dependencies



4. Ensure Required Files Exist

A black background with red text

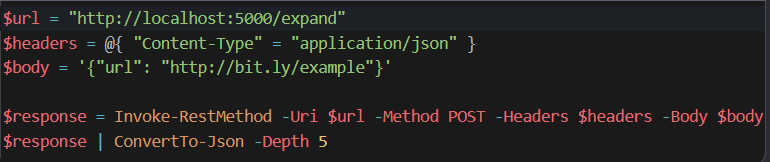
AI-generated content may be incorrect.

5. Run Flask Server

A screen shot of a computer

AI-generated content may be incorrect.

6. In Another Terminal Window, Test the API



# 7. Error Handling:

Current Error Handling Features

|  |  |
| --- | --- |
| Error Type | Description |
| Empty or missing URL | Returns {“error”: “No URL provided”} |
| Malformed JSON input | Caught by “request.get\_json()” |
| Timeout during expansion | requests.exceptions.Timeout  returns “Request timed out” |
| Too many redirects | “TooManyRedirects”  Returns “Too many redirects” |
| DNS or connection failure | ConnectionError  Returns “Failed to connect to destination" |
| WHOIS lookup failures | Wrapped in try/except -> returns (“reputation”: “unknown”) |
| Unexpected errors | General “Exception” catch – all -> “Internal server error” |

These features mean that while the app will not break down from malformed inputs or bad links, it can still benefit from improvements like logging and retry logic.

# 8. Security Considerations:

|  |  |
| --- | --- |
| Feature | Status |
| Safe URL Expansion | Only follows redirects, does not open the final page |
| No Sensitive Data | No cookies, tokens or secrets used |
| Domain Reputation Check | Helps to identify suspicious sites |
| WHOIS lookup | Provides more information about the domain owner and age |
| Future Enhancement | Integrates APIs like Google Safe Browsing for better security |

# 9. Future Enhancements:

These are some of the ideas I am planning to work on this application in the future.

|  |  |
| --- | --- |
| Enhancement | Description |
| Structured Error Codes | Return standard HTTP status codes and structured error objects |
| Configurable Timeouts | Allow users to set timeout limits via config |
| Redirecting Chain Safety | Limiting maximum amount of redirect depth and detect loops |
| Valid Domain Format | Ensuring domain names are valid before processing |

# 10. Testing Strategy

While the current code doesn’t include an automated test suite, it’s **testable**, and I am working on writing different kinds of tests to ensure reliability and preventing bugs when adding new features.

**Types of tests to add:**

|  |  |  |
| --- | --- | --- |
| **Test Type** | **What it does** | **Why it’s useful** |
| Unit Tests | Test individual functions (ex- “check\_blacklist()”, “check\_domain\_reputation()”) | Ensures all small parts work correctly |
| Integration Tests | Test full /expand endpoint with real or mock requests | Ensures all components work together |
| Functional Tests | Simulate user interaction (ex-sending real POST requests) | Verifies system works end – to – end |
| Regression Tests | Re-run old tests after updates | Makes sure that the new code doesn’t break old functionality |
| Security Tests | Try edge cases (ex- infinite redirects, malicious domains) | Helps to harden against abuse |
| Performance Tests | Benchmark how fast it validates URLs under load | Prepares for future scalability |

# 11. Flowchart Diagrams:

Below, are some flowchart diagrams to visualize the process.

* Validate Input & Check Cache

A screenshot of a computer flowchart

AI-generated content may be incorrect.

* **Expand Shortened URL & Handle Errors**

**A diagram of a company

AI-generated content may be incorrect.**

* **Analyze Final URL:**

**A screenshot of a computer

AI-generated content may be incorrect.**