# Name/ Moataz salah ezzat ahmed معتز بالله صلاح عزت احمد ID/240101499

# Python-Based Web Proxy Server – Technical Documentation

## Project Title

**Python-Based Web Proxy Server with Caching, Filtering, Throttling, and Persistence**

## Purpose

This project implements a custom HTTP/HTTPS proxy server using Python and Flask. The server provides the following functionalities:

* Forwarding client web requests to destination websites
* Caching responses to improve performance
* Filtering and blocking specific websites based on keywords
* Throttling bandwidth to simulate slow network conditions

## Technologies & Libraries

The following libraries are utilized in the project:

| **Library** | **Purpose** |
| --- | --- |
| **Flask** | Web framework for creating HTTP routes to expose the proxy server |
| **requests** | Fetches external web content, as Flask does not handle this natively |
| **time** | Manages cache expiration and introduces delays for throttling |
| **request** | used to access data from incoming HTTP requests (e.g., query parameters ) |
| **Response** | class provided by Flask for constructing HTTP responses to send back to the client. |

## 

## How to Run

## 1. Install Dependencies

Run the following command to install required Python libraries:

*pip install flask requests*

## 2. Start the Server

Execute the server script:

*python proxy\_server.py*

## 3. Access the Proxy

Use the proxy by making requests to:

*http://127.0.0.1:5050/proxy?url=https://example.com*

## Key Features

## 1. Request Forwarding

The proxy forwards client requests to the target URL and returns the response:



## 2. In-Memory Caching

Responses are cached in memory to serve repeated requests faster:

## C:\Users\win10\Downloads\ProxyServerWithCachingC.png

## 4. Content Filtering

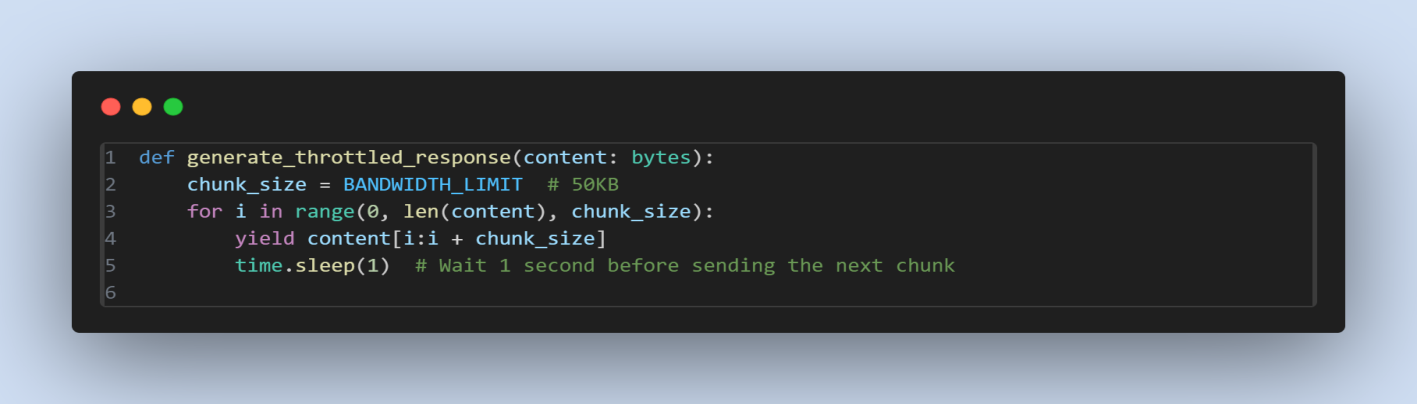
The proxy blocks websites containing specific keywords:

## C:\Users\win10\Downloads\filtering.png

## Also the same in domains blocking: C:\Users\win10\Downloads\domBlocked.png

## 5. Bandwidth Throttling

Simulates slow network conditions by limiting bandwidth:

****

## 6. Generator Usage (in Throttling)

The use of Python generators (yield) enables chunk-by-chunk data streaming, which is critical for implementing bandwidth throttling efficiently.  
  
**return** : give the whole pizza in one go.

**yield** : deliver **one slice at a time**, with a pause between each.

## Future Enhancements

* Handle persistent caching.
* Implement user authentication for proxy access
* Develop an admin dashboard for monitoring and configuration
* Introduce logging and analytics for request tracking