System Design – Task 1: URL Shortener Lite

Name: [Student Name] **Date:** December 10, 2025 **Assignment Title:** URL Shortener Lite Implementation

1. Overview

What is a URL Shortener?

A URL shortener is a service that converts long, unwieldy URLs into shorter, manageable aliases. When a user clicks the short link, the service automatically redirects them to the original, longer URL. This is commonly used to save space in social media posts, track clicks, or simply beautify links.

Project Implementation

For this task, I designed and implemented a lightweight URL shortener API. The system allows users to submit a valid HTTP URL and receive a 6-character unique alphanumeric code (e.g., `http://localhost:8000/a7B2x9`).

Technology Stack

- **Language:** Python 3.14
- **Framework:** FastAPI (chosen for its speed and automatic documentation)
- **Storage:** In-memory Python Dictionary (for simplicity and O(1) fast lookups)
- **Validation:** Pydantic (ensures input URLs are valid)

2. Detailed Step-by-Step Implementation

Step 1: Setting up the Application

We begin by importing the necessary libraries. `FastAPI` is the core framework, `HTTPException` handles errors (like 404s), and `Pydantic` models ensure data integrity.

```
 from fastapi import FastAPI, HTTPException, Request
from fastapi.responses import RedirectResponse
from pydantic import BaseModel, HttpUrl
import string
import random

app = FastAPI()
```

Step 2: Data Storage (In-Memory)

In a production system, we would use a database like PostgreSQL or Redis. For this "Lite" version, I used a Python dictionary `url_db`. This provides extremely fast key-value lookups.

```
 # In-memory storage
url_db = {}
```

- **Design Choice:** A dictionary is the simplest form of a hash map. It offers O(1) time complexity for saving and retrieving URLs.

Step 3: Input Validation

Using Pydantic's `BaseModel` and `HttpUrl`, the system automatically validates input. If a user tries to shorten "hello" instead of "http://google.com", the API will reject it before it even reaches our logic.

```
class ShortenRequest(BaseModel):
    long_url: HttpUrl
```

Step 4: Short ID Generation (Base62)

I implemented a standard Base62 encoding scheme (A-Z, a-z, 0-9) to generate short IDs.

```
def generate_short_id(length=6):
    characters = string.ascii_letters + string.digits
    while True:
        sid = ''.join(random.choice(characters) for _ in range(length))
        if sid not in url_db:
            return sid
```

- **Logic:** The function picks 6 random characters.

- **Collision Handling:** The `while True` loop checks `url_db` to ensure we never overwrite an existing ID (Collision Check). 6 characters give us roughly 56 billion possibilities ($62^6$).

Step 5: The Shorten Endpoint (POST)

This endpoint accepts the long URL, generates an ID, stores it, and returns the full short URL.

```
 @app.post("/shorten")
async def shorten_url(req: ShortenRequest, request: Request):
    short_id = generate_short_id()
    url_db[short_id] = req.long_url

    # Dynamically build the full URL based on where the server is running
    host = request.url.hostname or "127.0.0.1"
    port = request.url.port or 8000
    full_short = f"http://{host}:{port}/{short_id}"

    print("Current URL DB:", url_db) # For debugging/logging
    return {"short_url_id": short_id, "short_url": full_short, "original_url": req.
```

Step 6: The Redirect Endpoint (GET)

This is the core functional part. When a short ID is accessed, the system looks it up and redirects the browser.

```
 @app.get("/{short_id}")
async def redirect(short_id: str):
    if short_id not in url_db:
        raise HTTPException(status_code=404, detail="Short URL not found")
    return RedirectResponse(url_db[short_id])
```

---

3. Process Description & Design Choices

1. **Why FastAPI?** FastAPI was chosen over Flask because it supports asynchronous programming (`async/await`) out of the box, which is better for handling high concurrency. It also provides the Swagger UI (`/docs`) automatically, which made testing significantly easier.

2. **Why Base62?** We use Base62 because it is URL-safe (no special characters like `/` or `?` that would break the link) and dense (it stores more information per character than random numbers alone).

3. **Why In-Memory Storage?** The requirement was for a "Lite" task. Setting up a SQL database would add overhead. The in-memory dictionary demonstrates the core logic of key-value mapping without infrastructure complexity, though it is not persistent (data is lost on restart).

---

4. Testing & Screenshots

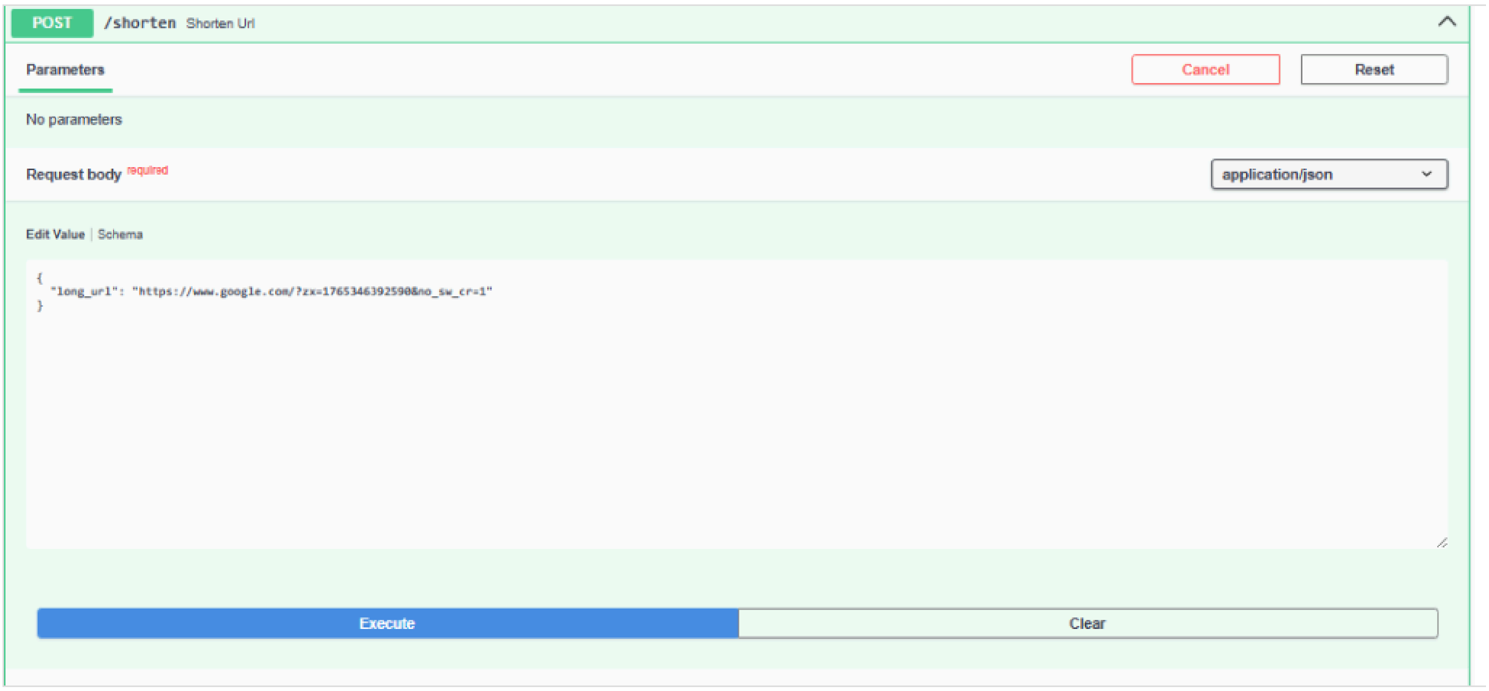
4.1 Server Running

The server is initialized using Uvicorn. (Placeholder: Your terminal showing `Uvicorn running on http://127.0.0.1:8000`)

4.2 Creating a Short URL

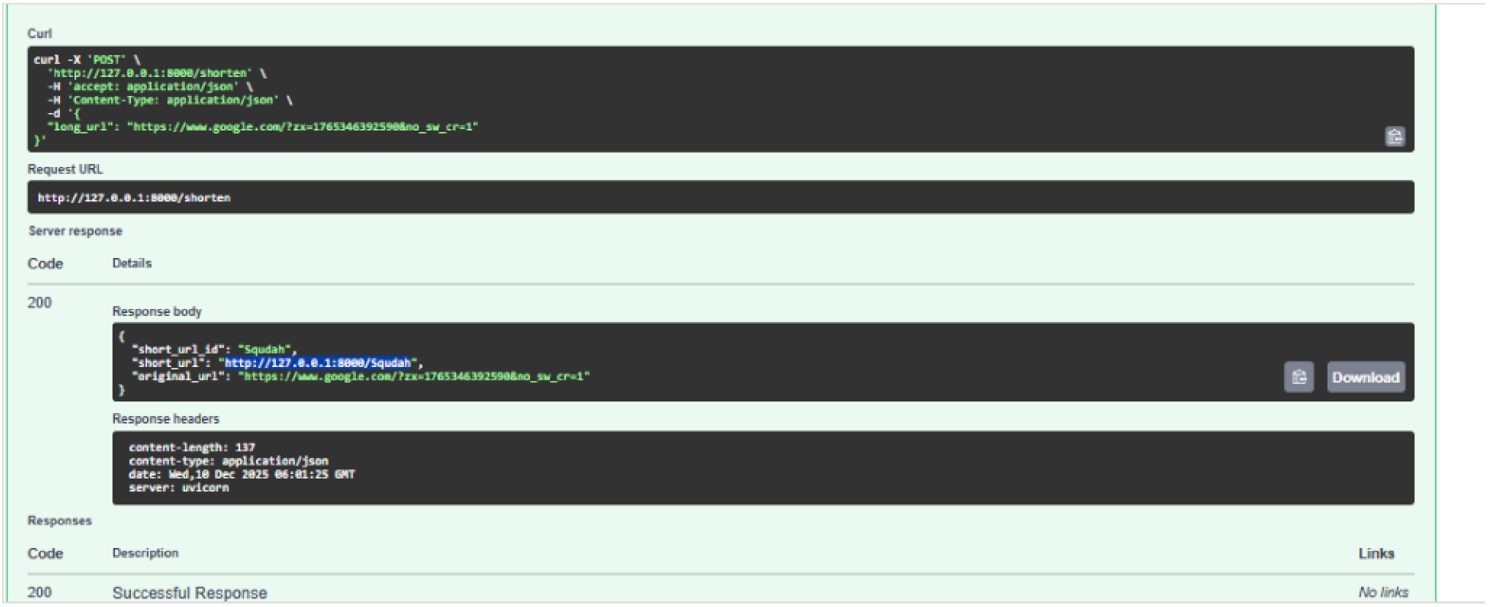
Using the Swagger UI to POST a request.

Input:



```
{
  "long_url": "https://www.google.com/?zx=1765346392590&no_sw_cr=1"
}
```

Output:



Curl

```
curl -X 'POST' \
  'http://127.0.0.1:8000/shorten' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{
  "long_url": "https://www.google.com/?zx=1765346392590&no_sw_cr=1"
}'
```

Request URL

```
http://127.0.0.1:8000/shorten
```

Server response

Code Details

200

Response body

```
{
  "short_url_id": "Squdah",
  "short_url": "http://127.0.0.1:8000/Squdah",
  "original_url": "https://www.google.com/?zx=1765346392590&no_sw_cr=1"
}
```

Response headers

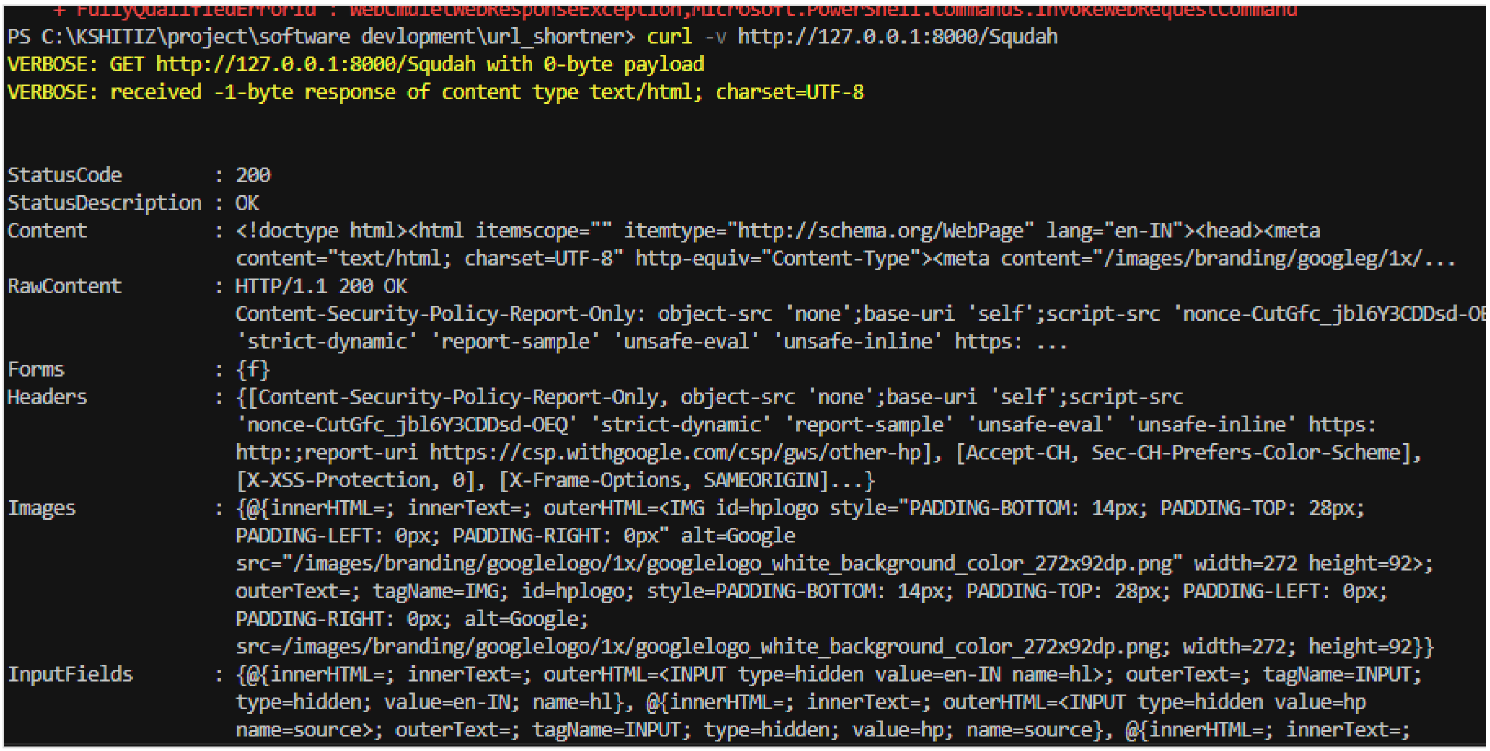
```
 content-length: 137
 content-type: application/json
 date: Wed,10 Dec 2025 06:01:25 GMT
 server: uvicorn
```

Responses

Code	Description	Links
200	Successful Response	*No links*

4.3 Testing the Redirect

Using `curl` *or browser to verify the 307 Redirect.*



```
    + FullyQualifiedErrorId : WebCmdletWebResponseException,Microsoft.PowerShell.Commands.InvokeWebRequestCommand
PS C:\KSHITIZ\project\software devlopment\url_shortner> curl -v http://127.0.0.1:8000/Squdah
VERBOSE: GET http://127.0.0.1:8000/Squdah with 0-byte payload
VERBOSE: received -1-byte response of content type text/html; charset=UTF-8


StatusCode        : 200
StatusDescription : OK
Content           : <!doctype html><html itemscope="" itemtype="http://schema.org/WebPage" lang="en-IN"><head><meta
                    content="text/html; charset=UTF-8" http-equiv="Content-Type"><meta content="/images/branding/googleg/1x/...
RawContent        : HTTP/1.1 200 OK
                    Content-Security-Policy-Report-Only: object-src 'none';base-uri 'self';script-src 'nonce-CutGfc_jbl6Y3CDDsd-OE
                    'strict-dynamic' 'report-sample' 'unsafe-eval' 'unsafe-inline' https: ...
Forms             : {f}
Headers           : {[Content-Security-Policy-Report-Only, object-src 'none';base-uri 'self';script-src
                    'nonce-CutGfc_jbl6Y3CDDsd-OEQ' 'strict-dynamic' 'report-sample' 'unsafe-eval' 'unsafe-inline' https:
                    http:;report-uri https://csp.withgoogle.com/csp/gws/other-hp], [Accept-CH, Sec-CH-Prefers-Color-Scheme],
                    [X-XSS-Protection, 0], [X-Frame-Options, SAMEORIGIN]...}
Images            : {@{innerHTML=; innerText=; outerHTML=<IMG id=hplogo style="PADDING-BOTTOM: 14px; PADDING-TOP: 28px;
                    PADDING-LEFT: 0px; PADDING-RIGHT: 0px" alt=Google
                    src="/images/branding/googlelogo/1x/googlelogo_white_background_color_272x92dp.png" width=272 height=92>;
                    outerText=; tagName=IMG; id=hplogo; style=PADDING-BOTTOM: 14px; PADDING-TOP: 28px; PADDING-LEFT: 0px;
                    PADDING-RIGHT: 0px; alt=Google;
                    src=/images/branding/googlelogo/1x/googlelogo_white_background_color_272x92dp.png; width=272; height=92}}
InputFields       : {@{innerHTML=; innerText=; outerHTML=<INPUT type=hidden value=en-IN name=hl>; outerText=; tagName=INPUT;
                    type=hidden; value=en-IN; name=hl}, @{innerHTML=; innerText=; outerHTML=<INPUT type=hidden value=hp
                    name=source>; outerText=; tagName=INPUT; type=hidden; value=hp; name=source}, @{innerHTML=; innerText=;
```

---

5. Final Notes

Challenges Faced

- **URL Validation:** Initially, any string could be sent. I solved this by adding Pydantic's `HttpUrl` type, which strictly validates that inputs are real URLs (including protocol http/https).
- **Host Construction:** Hardcoding `localhost` works for testing, but I wanted it to be robust. I used `request.url.hostname` to dynamically generate the short link regardless of where the server is hosted.

Key Learnings

This task solidified my understanding of RESTful API design. I learned how `RedirectResponse` works under the hood (sending a 307 status code to the browser). It also demonstrated how powerful Python's dictionaries are for O(1) retrieval operations, serving as a perfect mental model for how a real Redis cache would operate in a production URL shortener.