Design Document

HTTP URL Shortener Microservice

Company: Afford Medical Technologies Private Limited

Project: Campus Hiring Evaluation Task

1. Architecture Overview

The project is a single FastAPI microservice implementing an HTTP URL Shortener.

It supports:

- Creating globally unique short links
- Redirecting to the original URLs
- Managing URL expiry
- Logging application events via a reusable middleware.

2. Key Design Decisions

Aspect	Choice	Reason
Architecture	Microservice	Simpler to deploy & manage as a single unit
Framework	FastAPI (Python)	Asynchronous, fast to develop, widely adopted
Shortcode	Auto-generated or user-defined	Flexibility + uniqueness
Storage	Python Dictionary (in-memory)	Lightweight for this assignment
Logging	Custom API-call-based middleware	Meets mandatory test requirement
Expiry	TTL-based (datetime + validity)	Lightweight, no database scheduler needed
Redirection	HTTP 307 Temporary Redirect	Standard for temporary short links
Security	No authentication for demo	Focus is on core functionality

3. Technology Stack

Component	Technology
Language	Python 3.11+
Framework	FastAPI
HTTP Server	Uvicorn
Logging Client	httpx (Optional external API call)
Validation	Pydantic models
Data Storage	Python in-memory dict
Deployment	Localhost (Cloud ready)

4. Data Modeling

In-memory Store:

```
url_store = {
    "shortcode": {
        "url": "<original_long_url>",
        "expires": <expiry_datetime>
    }
}
```

Request Model:

```
{
    "url": "<long URL>",
    "validity": <minutes, optional>,
    "shortcode": "<custom short code, optional>"
}
```

Response Model:

```
{
    "shortlink": "<base_url>/<shortcode>",
    "expiry": "<ISO8601 timestamp>"
}
```

5. API Endpoints

Method	Endpoint	Auth	Description
POST	/shorturl	No	Create a shortened URL
GET	/{shortcode}	No	Redirect to original URL

6. Logging Strategy

All important actions are logged using a custom log() function, not the built-in Python logger.

Example log:

```
[BACKEND] [INFO] [shortener] - Created shortlink http://localhost:8000/abcd for
https://example.com
```

Log Levels:

- INFO: Successful operations
- ERROR: Client-side issues (e.g., shortcode not found)
- FATAL: Unexpected exceptions

7. Error Handling Approach

Case	HTTP Code	Response
Shortcode collision	400	Shortcode already in use
Invalid shortcode	404	Shortcode not found
Expired link	410	Shortcode expired
Invalid inputs	400	Validation error
Internal error	500	Internal Server Error

8. Assumptions

- System will run in single-process mode (no horizontal scaling in test setup).
- Shortened links expire after the specified validity period.
- No persistent database needed for the evaluation.
- Redirection should be open/publicly accessible.

9. Future Scalability Suggestions

Area	Suggested Upgrade
Storage	Replace dict with Redis / PostgreSQL
Logging	Integrate with ELK Stack / Datadog
Auth	Add OAuth2 / JWT if needed
Deployment	Dockerize & deploy to AWS / GCP
Analytics	Track click counts, popular links

Conclusion

This microservice follows modular design principles, separating:

- API routing
- Business logic
- Logging
- Data modeling

It fulfills the problem requirements while being ready for production expansion with minimal refactoring.