Design & Architecture Document

Project: URL Shortener Microservice

Candidate: Boya Mallesh

Technology: Python + FastAPI

1. Objective

To build a scalable, secure, and maintainable HTTP URL Shortener Microservice with:

- Custom & auto-generated shortcodes
- Redirection support
- URL expiry handling
- Click tracking with metadata
- Robust logging via custom middleware

2. Architecture Overview

The application follows a modular microservice architecture, structured into:

- main.py – API routing and business logic
- storage.py – In-memory data store
- middleware.py – Custom logging middleware
- Future-ready for database integration and load balancing

3. Technology Selections

Web Framework: FastAPI	High performance (async), automatic docs, Pydantic validation
Server: Uvicorn	Lightweight ASGI server suitable for async web apps
Data Storage: Python dict	Simple in-memory DB for evaluation; easy to replace with Redis/SQL
Logging: Custom Middleware	Required by evaluation; gives full control over format and scope
Time & Parsing: datetime	To handle expiry and ISO 8601 timestamp formatting

4. Key Code Design Choices

- URL Shortening: Supports custom or autogenerated shortcode, validated and ensured unique
- Expiry Handling: Defaults to 30 minutes; stored as UTC datetime
- Redirection: Checks shortcode existence & expiry, then redirects
- Analytics: Tracks timestamp, referrer, and mocked geo location
- Logging: Custom middleware logs every request (method, URL, timestamp)

5. Data Modeling

```
db = {
  "abcd1": {
    "original_url": str,
    "created_at": datetime,
    "expires_at": datetime,
    "clicks": int
  }
}

click_stats = {
  "abcd1": [
    {
      "timestamp": str,
      "referrer": str,
      "geo": str
    }
  ]
}
```

6. API Summary

POST /shorturls – Create shortened URL

GET /shorturls/{shortcode} – Retrieve click stats and metadata

GET /{shortcode} – Redirect to original URL

7. Assumptions Made

- All users are pre-authorized (no login/auth required)
- Validity input is always in minutes
- If the link is expired, it returns 410 Gone
- Geolocation is mocked as "IN" for all users
- In-memory DB suffices for this test; can be swapped for Redis/PostgreSQL

8. Scalability Considerations

- Storage Layer: Easily swappable with Redis/PostgreSQL
- Auto-scaling: FastAPI + Uvicorn supports containerization
- Caching: Can use Redis TTL for expired shortcodes
- Rate Limiting: Addable via middleware or API gateway

9. Future Enhancements

- Add user accounts & authentication
- Real-time analytics dashboard
- Geo-IP detection via external APIs
- Persistent DB with ORM
- CLI tools for admin tasks