NAME: VIDUSHI DWIVEDI

USN:22BTRCL167

System Design Document: HTTP URL Shortener Microservice

Problem Overview:

This project implements a URL Shortener microservice using FastAPI (Python) and SQLite (with SQLAlchemy ORM). The service provides RESTful APIs to shorten long URLs and redirect users from short codes to the original URLs. The design emphasizes simplicity, scalability, and maintainability.

Architectural Choices:

Framework: FastAPI was chosen for its high performance, asynchronous support, and automatic OpenAPI documentation, enabling rapid development and easy testing.

Database: SQLite is used for persistent storage in this implementation, with SQLAlchemy ORM for database abstraction. This allows for easy migration to more robust databases (e.g., PostgreSQL) in production.

Asynchronous I/O: All endpoints and database operations are asynchronous, supporting high concurrency and responsiveness.

Containerization (Optional): The service can be easily containerized for deployment using Docker.

Key Design Decisions:

API Endpoints: These are the points that mark the initial beginning and terminal ends of the API. (mid-break points can also be included to suggests the intermediate changes made (if any)).

POST /shorten: This accepts a long URL and returns a unique short code.

GET /{short_code}: Redirects to the original URL if the code exists.

Code Snippets:

- Short codes are generated using random alphanumeric strings, checked for uniqueness in the database.

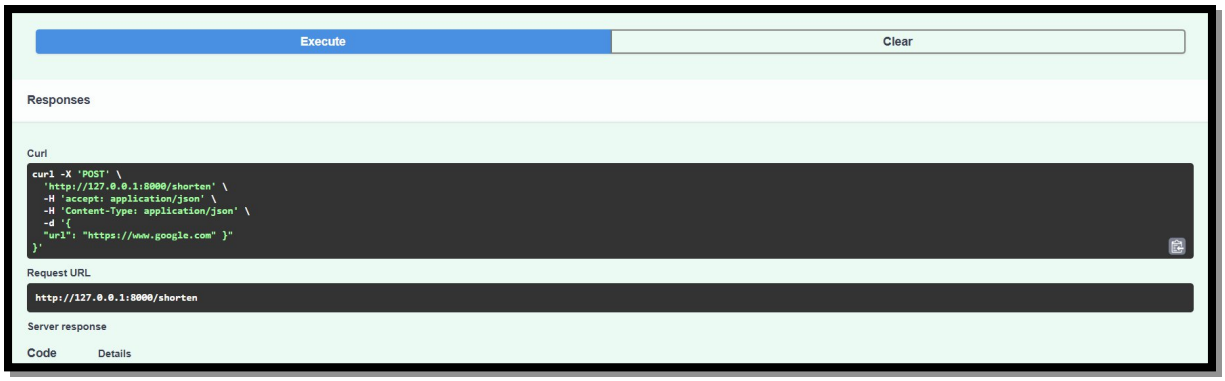
main.py > ...

```python

from fastapi import FastAPI, HTTPException, Request, Depends
from fastapi.responses import RedirectResponse
from pydantic import BaseModel, HttpUrl
from sqlalchemy.ext.asyncio import AsyncSession, create_async_engine
from sqlalchemy.orm import sessionmaker, declarative_base
from sqlalchemy import Column, Integer, String, select
import string, random, asyncio

DATABASE_URL = "sqlite+aiosqlite:///./urls.db"
engine = create_async_engine(DATABASE_URL, echo=True)
SessionLocal = sessionmaker(engine, expire_on_commit=False, class_=AsyncSession)
Base = declarative_base()

class URLMap(Base):
    __tablename__ = "urls"
    id = Column(Integer, primary_key=True, index=True)
    short_code = Column(String(10), unique=True, index=True)
    long_url = Column(String, nullable=False)

```

```python
class URLMap(Base):
    __tablename__ = "urls"
    id = Column(Integer, primary_key=True, index=True)
    short_code = Column(String(10), unique=True, index=True)
    long_url = Column(String, nullable=False)

app = FastAPI()

class URLRequest(BaseModel):
    url: HttpUrl

def generate_short_code(length=6):
    return ''.join(random.choices(string.ascii_letters + string.digits,
    k=length))

async def get_db():
    async with SessionLocal() as session:
        yield session
```

```python
@app.on_event("startup")
async def on_startup():
    async with engine.begin() as conn:
        await conn.run_sync(Base.metadata.create_all)

@app.post("/shorten")
async def shorten_url(request: URLRequest, db: AsyncSession = Depends(get_db)):
    # Generate unique short code
    for _ in range(10):
        short_code = generate_short_code()
        result = await db.execute(select(URLMap).where(URLMap.short_code ==
        short_code))
        if not result.scalar():
            break
    else:
        raise HTTPException(status_code=500, detail="Could not generate unique
        short code")
    url_map = URLMap(short_code=short_code, long_url=str(request.url))
    db.add(url_map)
    await db.commit()
    return {"short_url": f"/{short_code}"}
```

```python
@app.get("/{short_code}")
async def redirect_url(short_code: str, db: AsyncSession = Depends(get_db)):
    result = await db.execute(select(URLMap).where(URLMap.short_code ==
    short_code))
    url_map = result.scalar()
    if not url_map:
        raise HTTPException(status_code=404, detail="URL not found")
    return RedirectResponse(url_map.long_url)
```



Data Validation:

- Pydantic models ensure only valid URLs are accepted.

Persistence:

- URLs and their short codes are stored in a relational database for durability and scalability.

Extensibility:

- The codebase is designed in a modular format, allowing for easy addition of features such as analytics, authentication, or rate limiting.

Data Modelling:

- Table: URLs
- `id` (Integer, Primary Key)
- `short_code` (String, Unique, Indexed)
- `long_url` (String, Not Null)

Technology Selections & Justifications:

- ***FastAPI:*** Modern, async, type-safe, and auto-generates documentation.
- ***SQLAlchemy:*** Abstracts database logic, making migrations and upgrades easier.
- ***SQLite:*** Lightweight, file-based DB for demo/development; can be swapped for PostgreSQL/MySQL in production.
- ***Uvicorn:*** ASGI server for running FastAPI apps.

Assumptions:

- The service is initially deployed for demo or small-scale use; for production, a more robust DB and distributed cache may be used.
- No authentication or rate limiting is implemented, but the design allows for easy integration.
- Short code collisions are rare due to randomness and are checked before insertion.

Scalability & Maintainability:

- The use of async endpoints and database sessions supports high concurrency.
- The modular codebase and ORM abstraction allow for easy scaling, refactoring, and feature addition.
- The system can be containerized and deployed in cloud environments.

Future Enhancements:

- Analytics can be added via click tracking.
- User authentications can be enabled with an additional rate limiting feature (based on the frequency of user visits in order to shorten URLs).
- Support for custom short codes can be included.
- Migration to distributed databases and caching can be implemented as an for large-scale deployments.