🌐 Service Design & RESTful APIs

Generative AI Bootcamp – Week 1, Day 3, Session 1

November 19, 2025

Learning Objectives

- Understand the principles of RESTful API design
- Recognize the role of APIs in AI system integration
- Explore HTTP methods, status codes, and data formats
- Introduce FastAPI concepts for service implementation

Why APIs Matter in AI Systems

- Connect **models**, **data stores**, and **UI layers**
- Enable **microservice** architectures
- Allow scalable deployment of LLM-based endpoints
- Support interoperability between tools (LangChain, Vertex, Watson X)

Why REST Is Relevant (and what it is)

- Software architechtural style created for the development of the WWW
- Stands for *Representational State Transfer*
- Defines a set of constraints for how the architecture of a distributed hypermedia system should behave
- Emphasizes uniform interfaces, independent deployment of components, scalability of interactions, and a layered architecture (promote caching, enforce security, and encapsulate legacy systems).

REST constraints

- **Client/Server** – Clients are separated from servers by a well-defined interface
- **Stateless** – A specific client does not consume server storage when the client is "at rest"
- **Cache** – Responses indicate their own cacheability
- **Uniform interface**
- **Layered system** – A client cannot ordinarily tell whether it is connected directly to the end server, or to an intermediary along the way
- **Code on demand** (optional) – Servers are able to temporarily extend the functionality of a client by transferring logic to the client

RESTful API Design Principles

1. **Statelessness** — each request is independent
2. **Resource orientation** — use nouns for endpoints (`/models`, `/predictions`)
3. **HTTP verbs** — `GET`, `POST`, `PUT`, `DELETE`
4. **Consistent status codes** — `200 OK`, `400 Bad Request`, `500 Server Error`
5. **JSON everywhere** — simple, human-readable data exchange

Example: Inference API Workflow

```
Client → FastAPI Service → Model Inference Layer → Response
```

Component	Responsibility
Client	Sends request with input text
Service	Validates request, routes to model
Model Layer	Processes data, generates output
Response	Returns JSON-formatted prediction

Example FastAPI Endpoint

```python
from fastapi import FastAPI
from pydantic import BaseModel

app = FastAPI()

class Input(BaseModel):
    text: str

@app.post("/predict")
def predict(input: Input):
    return {"output": input.text[::-1]}
```

API Design Best Practices

- Validate inputs with **Pydantic models**
- Use clear and consistent routes
- Log all requests/responses (structured logging)
- Version your API: `/v1/predict`
- Add health checks (`/ping`)

Takeaways

- REST is the foundation for scalable AI systems
- **FastAPI** simplifies endpoint design with validation and docs
- Clean service design enables easier integration and testing