# **PART 2: Backend Engineering**

## **Backend Architecture**

### **Technologies**

* Framework: FastAPI for high-performance asynchronous API
* Authentication: OAuth2 with JWT
* Validation: Pydantic for data validation
* API Documentation: Automatic Swagger/OpenAPI documentation
* Background Tasks: Celery or FastAPI background tasks

### **API Design**

1. RESTful Endpoints

/api/v1/conversations # Collection endpoints

/api/v1/conversations/{id} # Resource endpoints

/api/v1/conversations/{id}/messages # Sub-resource endpoints

1. GraphQL API (Optional alternative)

type Conversation {

id: ID!

title: String

messages: [Message!]!

createdAt: DateTime!

}

type Query {

conversation(id: ID!): Conversation

conversations: [Conversation!]!

}

### **Code Organization**

backend/

├── app/

│ ├── api/

│ │ ├── endpoints/

│ │ │ ├── auth.py

│ │ │ ├── conversations.py

│ │ │ ├── llm.py

│ │ │ └── users.py

│ │ ├── dependencies.py

│ │ └── router.py

│ ├── core/

│ │ ├── config.py

│ │ ├── security.py

│ │ └── logging.py

│ ├── db/

│ │ ├── models.py

│ │ └── repositories.py

│ ├── services/

│ │ ├── llm\_service.py

│ │ ├── embedding\_service.py

│ │ └── user\_service.py

│ ├── schemas/

│ │ ├── conversation.py

│ │ ├── message.py

│ │ └── user.py

│ └── main.py

├── tests/

│ ├── api/

│ ├── services/

│ └── conftest.py

├── alembic/

│ └── versions/

├── Dockerfile

└── requirements.txt

### **Best Practices**

1. API Design:
   * Follow RESTful principles for resource management
   * Use versioned endpoints (e.g., /api/v1/resource)
   * Implement comprehensive error handling with meaningful status codes
   * Structure responses consistently with standardized formats
2. Dependency Injection:

# dependencies.py

from fastapi import Depends

from sqlalchemy.orm import Session

from app.db.session import get\_db

from app.services.llm\_service import LLMService

def get\_llm\_service(db: Session = Depends(get\_db)) -> LLMService:

return LLMService(db)

* + Use FastAPI's dependency injection system
  + Create reusable dependencies for common functionality
  + Implement scoped dependencies for request-level resources

1. Environment Configuration:

# config.py

from pydantic import BaseSettings, PostgresDsn, SecretStr

class Settings(BaseSettings):

API\_V1\_STR: str = "/api/v1"

PROJECT\_NAME: str = "LLM Application"

POSTGRES\_SERVER: str

POSTGRES\_USER: str

POSTGRES\_PASSWORD: SecretStr

POSTGRES\_DB: str

SQLALCHEMY\_DATABASE\_URI: PostgresDsn = None

# LLM Config

LLM\_API\_KEY: SecretStr

LLM\_MODEL\_NAME: str = "gpt-4"

LLM\_MAX\_TOKENS: int = 2048

class Config:

env\_file = ".env"

settings = Settings()

* + Use environment variables for configuration
  + Never hardcode sensitive information
  + Implement configuration validation with Pydantic

1. Logging and Monitoring:

# logging.py

import logging

import json

from datetime import datetime

class JSONFormatter(logging.Formatter):

def format(self, record):

log\_record = {

"timestamp": datetime.utcnow().isoformat(),

"level": record.levelname,

"message": record.getMessage(),

"module": record.module,

"function": record.funcName,

"line": record.lineno

}

if hasattr(record, "correlation\_id"):

log\_record["correlation\_id"] = record.correlation\_id

return json.dumps(log\_record)

def setup\_logging():

handler = logging.StreamHandler()

handler.setFormatter(JSONFormatter())

logging.basicConfig(

handlers=[handler],

level=logging.INFO

)

* + Implement structured logging
  + Use correlation IDs to track requests across services
  + Log appropriate information for debugging and auditing

1. Error Handling:

# errors.py

from fastapi import HTTPException, Request

from fastapi.responses import JSONResponse

class LLMServiceError(Exception):

def \_\_init\_\_(self, message: str, code: str = "llm\_error"):

self.message = message

self.code = code

super().\_\_init\_\_(self.message)

async def llm\_exception\_handler(request: Request, exc: LLMServiceError):

return JSONResponse(

status\_code=500,

content={

"error": {

"code": exc.code,

"message": exc.message,

"request\_id": request.state.request\_id

}

}

)

# In main.py

app.add\_exception\_handler(LLMServiceError, llm\_exception\_handler)

* + Create custom exception classes
  + Implement global exception handlers
  + Return consistent error response formats
  + Include request IDs in error responses

## 

## **Backend Infrastructure**

### **AWS Infrastructure**

1. Compute Options
   * ECS Fargate for containerized services
   * EC2 instances with Auto Scaling Groups
   * Elastic Beanstalk for simplified deployment
   * Lambda for serverless microservices
2. Networking
   * VPC with public and private subnets
   * Application Load Balancer for traffic distribution
   * API Gateway for API management
   * WAF for security and rate limiting

### **GCP Infrastructure**

1. Compute Options
   * Cloud Run for containerized services
   * Compute Engine with managed instance groups
   * App Engine for simplified deployment
   * Cloud Functions for serverless components
2. Networking
   * VPC network configuration
   * Cloud Load Balancing
   * API Gateway for API management
   * Cloud Armor for security

### **Best Practices**

Infrastructure as Code (IaC):

# Example Terraform configuration for FastAPI service on ECS

resource "aws\_ecs\_task\_definition" "fastapi\_task" {

family = "fastapi-service"

network\_mode = "awsvpc"

requires\_compatibilities = ["FARGATE"]

cpu = 256

memory = 512

execution\_role\_arn = aws\_iam\_role.ecs\_execution\_role.arn

container\_definitions = jsonencode([{

name = "fastapi-container"

image = "${aws\_ecr\_repository.fastapi\_repo.repository\_url}:latest"

essential = true

portMappings = [{

containerPort = 8000

hostPort = 8000

protocol = "tcp"

}]

environment = [

{ name = "POSTGRES\_SERVER", value = aws\_rds\_cluster.postgres.endpoint },

{ name = "POSTGRES\_DB", value = "app" },

{ name = "POSTGRES\_USER", value = "app\_user" }

]

secrets = [

{ name = "POSTGRES\_PASSWORD", valueFrom = aws\_secretsmanager\_secret.db\_password.arn },

{ name = "LLM\_API\_KEY", valueFrom = aws\_secretsmanager\_secret.llm\_api\_key.arn }

]

logConfiguration = {

logDriver = "awslogs"

options = {

"awslogs-group" = aws\_cloudwatch\_log\_group.fastapi\_logs.name

"awslogs-region" = var.aws\_region

"awslogs-stream-prefix" = "fastapi"

}

}

}])

}

* + Define infrastructure as code with Terraform, AWS CDK, or Pulumi
  + Version control infrastructure definitions
  + Use modules for reusable components
  + Implement variable parameterization