Module 3: Intermediate FastAPI

🚀 Overview

- Request and response models
- Response status codes
- Error handling
- Modular architecture and routers
- Environment variables and config management
- Advanced logging

Pydantic Models

- Pydantic models are used for:
  - Validating input data (requests)
  - Structuring output data (responses)

Response Models

- Use `response_model` parameter:

```
class Item_Response(BaseModel):
 id: int
 name: str
```

```
@app.get("/{id}", response_model=Item_Response)
def get_item(id: int):
  for item in items:
      if item.id == id:
          return item
  raise HTTPException(status_code=404, detail="Item not found")
```

Custom Response Status Codes

- Define expected status codes:

```
@app.post("/items/", status_code=201)
```

```
@app.delete("/items/{id}", status_code=status.HTTP_204_NO_CONTENT)
```

Error Handling

- Use `HTTPException` for custom errors:

```
from fastapi import HTTPException

@app.get("/items/{id}")
def get_item(id: int):
    if id not in items:
        raise HTTPException(status_code=404, detail="Item not found")
    return items[id]
```

- FastAPI automatically handles validation errors and returns 422 status code with details.

Modular Architecture

- Organize code into modules:
  - `routers/` for route definitions
  - `models/` for Pydantic models
  - `services/` for business logic
- Example structure:

```
project/
    ├── main.py
    ├── routers/
        ├── users.py
        └── items.py
```

Modular Architecture: main.py

```python
from fastapi import FastAPI
from routers import users, items

app = FastAPI()

app.include_router(users.router, prefix="/users", tags=["Users"])
app.include_router(items.router, prefix="/items", tags=["Items"])

@app.get("/")
def read_root():
    return {"message": "Welcome to the FastAPI backend!"}
```

Modular Architecture: routers/users.py

```python
from fastapi import APIRouter

router = APIRouter()

@router.get("/")
def get_users():
    return {"users": ["Alice", "Bob", "Charlie"]}
```

Modular Architecture: routers/items.py

```python
from fastapi import APIRouter

router = APIRouter()

@router.get("/")
def get_items():
    return {"items": ["Item1", "Item2", "Item3"]}
```

Environment Variables

- Store secrets and configs in `.env` file
- Use `python-dotenv` or `pydantic-settings`
- `.env` example:

```
DATABASE_URL=postgresql://user:pass@localhost/db
SECRET_KEY=supersecret
DB_USER=postgres
DB_PASS=postgres
```

Loading config from `.env`

- Install `python-dotenv`

```
pip install python-dotenv
```

- Use environment variables

```
import os
from dotenv import load_dotenv

load_dotenv()

DB_USER = os.getenv("DB_USER", "postgres")
DB_PASS = os.getenv("DB_PASS", "postgres")
```

Middleware in FastAPI

- Middleware is a function that runs *before* and *after* each **request** in FastAPI.
- The `@app.middleware("http")` decorator registers log_requests as middleware for all HTTP requests.

Logging Middleware (Request/Response)

```python
from fastapi import Request
import time

@app.middleware("http")
async def log_requests(request: Request, call_next):
    start_time = time.time()

    response = await call_next(request)

    process_time = (time.time() - start_time) * 1000
    logger.info(f"{request.method} {request.url.path} - {response.status_code} - {process_time:.2f}ms")

    return response
```

uvicorn Logging

- FastAPI usually runs with Uvicorn, which has its own logging config:

```
uvicorn app:app --log-level info --log-config logging.yaml
```

- **logging.yaml** can define loggers, handlers, and formatters.

Homework

Link to homework

Section: **Practical Exercises: FastAPI Modular App**

🎯 Remember

- The structure of a FastAPI app
- Modular architecture and routes
- Request and response models using pydantic
- Using status code in response
- Error handling using HTTPException
- Environment variables for configuration