

# FastAPI Python Concepts Study Notes

## Table of Contents

1. Introduction to FastAPI
2. Setting Up FastAPI
  - Installation
  - First FastAPI Application
3. Understanding Request and Response
  - Request Examples
  - Response Models
4. Path Parameters and Query Parameters
  - Path Parameters
  - Query Parameters
5. Request Body and Data Validation
  - Pydantic Models
  - Validating Input Data
6. Middleware and Dependency Injection
  - Understanding Middleware
  - Dependency Injection Concepts
7. Asynchronous Programming in FastAPI
  - Async and Await
  - Background Tasks
8. Security in FastAPI
  - Authentication
  - Authorization
9. FastAPI and Database Integration
  - Connecting to a Database
  - Using SQLAlchemy with FastAPI

## 10. Quick Testing and Documentation

- Testing FastAPI Applications
- Auto-generated API Documentation

## 11. Summary

---

# 1. Introduction to FastAPI

- **FastAPI**: A modern, fast (high-performance), web framework for building APIs with Python 3.6+ based on standard Python type hints.
- **Key Features**:
  - Fast: One of the fastest Python frameworks.
  - Easy to use: Designed with developer productivity in mind.
  - Automatic Documentation: Interactive docs via Swagger UI and ReDoc.

# 2. Setting Up FastAPI

## ### Installation

- Install FastAPI and an ASGI server (e.g., Uvicorn):

```
pip install fastapi uvicorn
```

## ### First FastAPI Application

- Code Example:

```
from fastapi import FastAPI
```

```
app = FastAPI()
```

```
@app.get("/")
```

```
def read_root():
```

```
    return {"Hello": "World"}
```

Run using: `uvicorn filename:app --reload`

### 3. Understanding Request and Response

#### ### Request Examples

- **HTTP Methods**: GET, POST, PUT, DELETE, etc.
- **FastAPI Routes**: Use decorators to define routes.

```
@app.get("/items/{item_id}")
```

```
def read_item(item_id: int):
```

```
    return {"item_id": item_id}
```

#### ### Response Models

- Use Pydantic models to define response schema:

```
from pydantic import BaseModel
```

```
class Item(BaseModel):
```

```
    name: str
```

```
    price: float
```

```
    is_offer: bool = None
```

```
@app.post("/items/")
```

```
def create_item(item: Item):
```

```
    return item
```

### 4. Path Parameters and Query Parameters

#### ### Path Parameters

- Dynamic parameters in the URL.
- Example:

```
@app.get("/users/{user_id}")
```

```
def read_user(user_id: int):
```

```
    return {"user_id": user_id}
```

### ### Query Parameters

- Optional parameters in the URL after the question mark.
- Example:

```
@app.get("/items/")
```

```
def read_items(skip: int = 0, limit: int = 10):
```

```
    return {"skip": skip, "limit": limit}
```

## 5. Request Body and Data Validation

### ### Pydantic Models

- Define request body using Pydantic:

```
class User(BaseModel):
```

```
    username: str
```

```
    email: str
```

```
    full_name: str = None
```

### ### Validating Input Data

- FastAPI uses Pydantic to validate data automatically. If validations fail, FastAPI returns a 422 Unprocessable Entity error.

## 6. Middleware and Dependency Injection

### ### Understanding Middleware

- Middleware processes requests before they reach the endpoint and responses before they are sent.
- Example:

```
from starlette.middleware.cors import CORSMiddleware
```

```
app.add_middleware(
```

```
    CORSMiddleware,
```

```
    allow_origins=["*"],
```

```
allow_methods=["*"],
allow_headers=["*"],
)
```

### Dependency Injection Concepts

- FastAPI allows defining dependencies for use in routes.
- Example:

```
def query_param(q: str = None):
    return q
```

```
@app.get("/items/")
```

```
def read_items(q: str = Depends(query_param)):
    return {"q": q}
```

## 7. Asynchronous Programming in FastAPI

### Async and Await

- Use asynchronous endpoints for better performance.
- Example:

```
import asyncio
```

```
@app.get("/items/")
```

```
async def read_items():
```

```
    await asyncio.sleep(1)
```

```
    return [{"item": "Item 1"}, {"item": "Item 2"}]
```

### Background Tasks

- Run long tasks in the background without blocking the main event loop:

```
from fastapi import BackgroundTasks
```

```
def write_log(message: str):
```

```
    with open("log.txt", mode="a") as log:
```

```
log.write(f"{message}\n")
```

```
@app.post("/send-notification/")
```

```
async def send_notification(background_tasks: BackgroundTasks, email: str):
```

```
background_tasks.add_task(write_log, f"Email sent to {email}")
```

```
return {"message": "Notification sent"}
```

## 8. Security in FastAPI

### ### Authentication

- FastAPI supports various authentication techniques, including OAuth2.
- Example:

```
from fastapi.security import OAuth2PasswordBearer
```

```
oauth2_scheme = OAuth2PasswordBearer(tokenUrl="token")
```

### ### Authorization

- Define permissions and roles using dependencies.
- Example:

```
from fastapi import Security, HTTPException
```

```
@app.get("/users/me")
```

```
async def read_users_me(token: str = Security(oauth2_scheme)):
```

```
user = get_current_user(token) # Custom logic to validate token
```

```
return user
```

## 9. FastAPI and Database Integration

### ### Connecting to a Database

- Use ORM like SQLAlchemy for database interaction.

### ### Using SQLAlchemy with FastAPI

- Quick setup example:

```

from sqlalchemy import create_engine, Column, Integer, String
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import sessionmaker

engine = create_engine("sqlite:///./test.db")
SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)

Base = declarative_base()

class User(Base):
    __tablename__ = "users"

    id = Column(Integer, primary_key=True, index=True)
    name = Column(String)

Base.metadata.create_all(bind=engine)

```

## 10. Quick Testing and Documentation

### ### Testing FastAPI Applications

- Use standard Python testing frameworks (e.g., pytest) for testing.
- Example:

```

from fastapi.testclient import TestClient

client = TestClient(app)

def test_read_root():
    response = client.get("/")
    assert response.status_code == 200
    assert response.json() == {"Hello": "World"}

```

### ### Auto-generated API Documentation

- FastAPI automatically generates interactive API documentation (Swagger UI and ReDoc):
- Swagger UI at: `http://127.0.0.1:8000/docs``

- ReDoc at: ``http://127.0.0.1:8000/redoc``

## 11. Summary

- FastAPI is a powerful framework for building APIs quickly and efficiently, leveraging Python's type hints.
- Key concepts include request/response handling, data validation, asynchronous programming, security features, and integrations with databases.
- Testing and auto-generated documentation enhance development productivity.

---

### End of Study Notes

Keep revising each section to solidify your understanding, and practice building small applications to apply what you've learned. Happy coding with FastAPI!