🚀FastAPI Complete Notes (Beginner to Advanced)

🌐1. Introduction to FastAPI

- FastAPI is a modern, fast (high-performance), web framework for building APIs with Python 3.7+.
- Based on **Starlette** for web handling and **Pydantic** for data validation.

🎠Key Features:

- Asynchronous & synchronous support
- Auto-generated Swagger & Redoc docs
- Built-in data validation with `pydantic`
- Fast performance (comparable to Node.js, Go)

```
pip install fastapi uvicorn
```

🛩️2. Basic Hello World App

```
# main.py
from fastapi import FastAPI

app = FastAPI()

@app.get("/")
def read_root():
    return {"message": "Hello, FastAPI!"}
```

```
uvicorn main:app --reload
```

Visit: `http://127.0.0.1:8000`

Docs:

- Swagger UI: `/docs`
- ReDoc: `/redoc`

🛺3. Path & Query Parameters

```
@app.get("/items/{item_id}")
def read_item(item_id: int, q: str = None):
    return {"item_id": item_id, "q": q}
```

🚣4. Request Body with Pydantic Models

```
from pydantic import BaseModel

class Item(BaseModel):
    name: str
    price: float
    is_offer: bool = False

@app.post("/items/")
def create_item(item: Item):
    return {"item": item}
```

🏍️5. Validation & Type Conversion

```
from typing import Optional
from fastapi import Query

@app.get("/validate/")
def validate_query(q: Optional[str] = Query(None, min_length=3,
max_length=50)):
    return {"q": q}
```

🔊 6. Path, Query, Header, and Cookie Parameters

```
from fastapi import Header, Cookie

@app.get("/params/")
def read_params(user_agent: str = Header(...), session_id: str =
Cookie(None)):
    return {"user_agent": user_agent, "session_id": session_id}
```

---

7. Dependency Injection

```python
from fastapi import Depends

def get_token(token: str = Query(...)):
    return token

@app.get("/secure/")
def secure_route(token: str = Depends(get_token)):
    return {"token": token}
```

---

8. Middleware & Event Handlers

```python
@app.middleware("http")
async def log_requests(request, call_next):
    response = await call_next(request)
    return response

@app.on_event("startup")
def startup_event():
    print("App started!")
```

🚇9. Response Models & Status Codes

```python
from fastapi import status
from fastapi.responses import JSONResponse

@app.post("/items/", status_code=status.HTTP_201_CREATED)
def create(item: Item):
    return JSONResponse(content={"item": item.dict()})
```

---

10. Exception Handling

```python
from fastapi import HTTPException

@app.get("/error/")
def error():
    raise HTTPException(status_code=404, detail="Item not found")
```

🏎️11. File Uploads & Form Data

```python
from fastapi import File, UploadFile

@app.post("/upload/")
def upload(file: UploadFile = File(...)):
    return {"filename": file.filename}
```

---

12. Authentication & Authorization (OAuth2 + JWT)

```python
from fastapi.security import OAuth2PasswordBearer
oauth2_scheme = OAuth2PasswordBearer(tokenUrl="token")

@app.get("/secure/")
def secure(token: str = Depends(oauth2_scheme)):
    return {"token": token}
```

🛖13. Asynchronous Programming

```python
@app.get("/async/")
async def async_example():
    await some_async_func()
    return {"message": "Done"}
```

---

14. Database Integration

```
pip install sqlalchemy databases asyncpg
```

```python
from databases import Database

database = Database("postgresql://user:pass@localhost/db")

@app.on_event("startup")
async def startup():
    await database.connect()

@app.on_event("shutdown")
async def shutdown():
    await database.disconnect()
```

🛕 15. Testing FastAPI Apps

```
pip install pytest httpx
```

```
from fastapi.testclient import TestClient
from main import app

client = TestClient(app)

def test_read_main():
    response = client.get("/")
    assert response.status_code == 200
    assert response.json() == {"message": "Hello, FastAPI!"}
```

🚚16. Security & Rate Limiting

```
pip install slowapi
```

Use `passlib`, `pyjwt` for hashing and token generation.

🚲17. Modular Project Structure (Best Practice)

```
app/
├── main.py
├── api/
│   ├── routes.py
├── core/
│   ├── config.py
├── models/
│   ├── user.py
├── db/
│   ├── database.py
├── schemas/
│   ├── user_schema.py
├── services/
│   ├── auth_service.py
```

18. FastAPI with Docker

Dockerfile

```
FROM python:3.10

WORKDIR /app
COPY . .
RUN pip install -r requirements.txt

CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "8000"]
```

docker-compose.yml

```
version: '3'
services:
  api:
    build: .
    ports:
      - "8000:8000"
```

🧭19. Async Background Tasks

```
from fastapi import BackgroundTasks

def write_log(message: str):
    with open("log.txt", "a") as f:
        f.write(message)

@app.post("/log/")
def log(background_tasks: BackgroundTasks):
    background_tasks.add_task(write_log, "User hit the endpoint\n")
    return {"message": "Logged"}
```

🏫20. FastAPI + WebSocket

```
from fastapi import WebSocket

@app.websocket("/ws")
async def websocket_endpoint(websocket: WebSocket):
    await websocket.accept()
    while True:
```

```
        data = await websocket.receive_text()
        await websocket.send_text(f"You said: {data}")
```

Summary: Why Use FastAPI?

- Blazing fast (thanks to ASGI and Starlette)
- Built-in Swagger docs
- First-class async support
- Excellent for microservices & REST APIs

 Want More?

Let me know if you want:

- Realtime WebSocket + Redis PubSub chat
- FastAPI + Celery + RabbitMQ background tasks
- FastAPI + MongoDB (Motor)
- FastAPI + React frontend integration
- FastAPI + GraphQL (Strawberry or Ariadne)