FastAPI From ZERO 🧠😰

(For someone who knows Spring Boot, but nothing about FastAPI)

This canvas is a **complete mental-model + hands-on guide**. By the end, FastAPI will feel as clear as Spring Boot.

We will move in this exact order:

1. How Spring Boot Dependency Injection works (mental model)
2. Mapping that DI model to FastAPI
3. What FastAPI actually is (internals & behind-the-scenes)
4. Request → Response lifecycle
5. Middleware (Logger middleware in depth)
6. In-memory Todo REST API
7. PostgreSQL REST API using **RAW SQL (no ORM)**

🤩 Spring Boot Dependency Injection (What you already know)

In Spring Boot

```java
@Service
public class TodoService {
    public List<String> getTodos() { ... }
}

@RestController
public class TodoController {

    @Autowired
    private TodoService todoService;
}
```

What is ACTUALLY happening

- Spring creates **objects (beans)** at startup
- Stores them inside **ApplicationContext**
- When controller starts → Spring **injects** the dependency

💡 You **never create objects manually** using `new`

🤪 FastAPI Dependency Injection (Same idea, different style)

FastAPI DI is **function-based**, not annotation-based.

FastAPI version of the same idea

```
def get_todo_service():
    return TodoService()

@app.get("/todos")
def get_todos(service=Depends(get_todo_service)):
    return service.get_todos()
```

Mental Mapping

Spring Boot	FastAPI
@Service	provider function
@Autowired	Depends()
ApplicationContext	Dependency graph

🔑 **FastAPI creates objects PER REQUEST (by default)** (not singleton like Spring)

🤫 What FastAPI REALLY is (Behind the scenes)

FastAPI is built on top of:

- **Starlette** → HTTP, routing, middleware
- **Pydantic** → validation & serialization
- **ASGI** → async server interface

Request Flow

```
Client → ASGI Server (uvicorn)
       → Middleware chain
       → Router
       → Dependency Injection
       → Path operation function
       → Response
```

Spring Boot equivalent:

```
Tomcat → Filters → DispatcherServlet → Controller → Response
```

🤬 Request → Response lifecycle (VERY IMPORTANT)

1. Request hits ASGI server

2. Middleware executes (before request)
3. Dependencies are resolved
4. Endpoint function executes
5. Middleware executes (after response)

💡 Middleware wraps **everything** 💡 Dependencies wrap **only endpoints**

🤭 Middleware in FastAPI (Logger Example)

Why middleware exists

- Logging
- Authentication
- Rate limiting
- Request modification

Logger Middleware

```python
import time
from fastapi import Request

async def logger_middleware(request: Request, call_next):
    start = time.time()
    response = await call_next(request)
    duration = time.time() - start
    print(f"{request.method} {request.url} - {duration:.2f}s")
    return response
```

Mounting middleware

```python
app.middleware("http")(logger_middleware)
```

Spring Boot mapping:

Spring	FastAPI
Filter	Middleware
OncePerRequestFilter	ASGI middleware

🤮 In-Memory Todo REST API (No DB) 🧠

Now we write **FULL WORKING CODE**, not snippets. This section alone is enough to understand FastAPI deeply.

Project structure (minimal)

```
app/
 ├── main.py
 └── todo/
      ├── models.py
      ├── store.py
      └── routes.py
```

---

todo/models.py (Request vs Response models)

IMPORTANT CONCEPT 👂

In FastAPI (like Spring DTOs): - **Request model** → what client sends - **Response model** → what server returns

Never mix them blindly.

```
from pydantic import BaseModel

# Request DTO (like CreateTodoRequest in Spring)
class TodoCreateRequest(BaseModel):
    title: str

# Request DTO for update
class TodoUpdateRequest(BaseModel):
    title: str
    completed: bool

# Response DTO
class TodoResponse(BaseModel):
    id: int
    title: str
    completed: bool
```

Spring Boot mapping: | Spring Boot | FastAPI | |------------|---------| | @RequestBody DTO | Pydantic Request Model | | ResponseEntity<DTO> | response_model |

---

todo/store.py (In-memory DB)

```
from typing import List
from .models import TodoResponse

# This acts like an in-memory database
todos: List[TodoResponse] = []
```

```python
current_id = 1


def get_all():
    return todos


def get_by_id(todo_id: int):
    for todo in todos:
        if todo.id == todo_id:
            return todo
    return None


def create(title: str):
    global current_id
    todo = TodoResponse(id=current_id, title=title, completed=False)
    todos.append(todo)
    current_id += 1
    return todo


def update(todo_id: int, title: str, completed: bool):
    todo = get_by_id(todo_id)
    if todo:
        todo.title = title
        todo.completed = completed
    return todo


def delete(todo_id: int):
    global todos
    todos = [t for t in todos if t.id != todo_id]
```

---

todo/routes.py (FULL CRUD)

```python
from fastapi import APIRouter, HTTPException
from typing import List
from .models import (
    TodoCreateRequest,
    TodoUpdateRequest,
    TodoResponse
)
from .store import get_all, get_by_id, create, update, delete

router = APIRouter(prefix="/todos", tags=["Todos"])

@router.get("/", response_model=List[TodoResponse])
def list_todos():
    return get_all()

@router.get("/{todo_id}", response_model=TodoResponse)
def get_todo(todo_id: int):
    todo = get_by_id(todo_id)
    if not todo:
        raise HTTPException(status_code=404, detail="Todo not found")
    return todo

@router.post("/", response_model=TodoResponse, status_code=201)
def create_todo(request: TodoCreateRequest):
    return create(request.title)

@router.put("/{todo_id}", response_model=TodoResponse)
def update_todo(todo_id: int, request: TodoUpdateRequest):
    todo = update(todo_id, request.title, request.completed)
    if not todo:
        raise HTTPException(status_code=404, detail="Todo not found")
    return todo

@router.delete("/{todo_id}", status_code=204)
def delete_todo(todo_id: int):
    delete(todo_id)
```

---

main.py (Application Entry Point)

```
from fastapi import FastAPI
from todo.routes import router as todo_router

app = FastAPI(title="FastAPI Todo App")

app.include_router(todo_router)

@app.get("/")
def root():
    return {"message": "FastAPI is running"}
```

Run the app

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

Open: - http://127.0.0.1:8000/docs

👙 You get **Swagger UI automatically**.

VERY IMPORTANT: Why Request & Response models matter

If you used ONE model: - client could send `id` - security issues - bad API design

FastAPI forces **clean API contracts** (better than Spring defaults).

🤯 PostgreSQL REST API (RAW SQL, No ORM) 👿

🤯 PostgreSQL REST API (RAW SQL, No ORM) 👿

Why RAW SQL?

- You understand DB fundamentals
- No magic
- Similar to JDBC template

Database Connection

```
import psycopg2

conn = psycopg2.connect(
    dbname="todo",
    user="postgres",
    password="password",
    host="localhost"
)
```

Table

```
CREATE TABLE todos (
  id SERIAL PRIMARY KEY,
  title TEXT,
  completed BOOLEAN
);
```

CRUD Endpoints (RAW SQL)

GET /todos

```
cursor.execute("SELECT * FROM todos")
```

GET /todos/{id}

```
cursor.execute("SELECT * FROM todos WHERE id=%s", (id,))
```

POST /todos

```
cursor.execute(
  "INSERT INTO todos (title, completed) VALUES (%s, %s)",
  (title, False)
)
```

UPDATE /todos/{id}

```
cursor.execute(
  "UPDATE todos SET title=%s, completed=%s WHERE id=%s",
  (title, completed, id)
)
```

DELETE /todos/{id}

```
cursor.execute("DELETE FROM todos WHERE id=%s", (id,))
```

Spring mapping:

Spring Boot	FastAPI
JdbcTemplate	psycopg2
@Repository	DB utility function

🥰 Why FastAPI feels "too easy" 🤯

Because: - No XML - No annotations - No reflection - Pure Python + type hints

FastAPI uses **type hints** to: - Validate input - Generate OpenAPI docs - Serialize responses

Swagger UI comes **for free**.

🥱 Final Mental Model (Lock this in 😊)

- FastAPI is **function-first**
- DI is **explicit**
- Middleware wraps everything

- Dependencies wrap endpoints
- Pydantic = DTOs
- Starlette = web engine

If you master this canvas → You can build **ANY FastAPI system**.

What we can do next 👇

- Convert this into clean architecture
- Add JWT auth
- Async DB version
- Compare FastAPI vs Spring Boot performance
- Build production-grade project

Just tell me 📀