FastAPI Course Notes - Sections 2, 3 & 4

Section 2: Setup & Installation

Python Installation

Mac Users:

- Download Python from official website (python.org)
- Use installer package (.pkg file)
- Verify installation: `python3 --version`

Windows Users:

- Download from python.org
- Check "Add Python to PATH" during installation
- Verify: `python --version`

VS Code Setup

Mac & Windows:

- Download VS Code from official website
- Install Python extension
- Configure Python interpreter
- Set up integrated terminal

Virtual Environments

Purpose: Isolate project dependencies to avoid conflicts

Windows:

```bash
# Create virtual environment
python -m venv project_name
# Activate
project_name\Scripts\activate
# Deactivate
deactivate
```

Mac:

```bash
# Create virtual environment
python3 -m venv project_name
# Activate
source project_name/bin/activate
# Deactivate
deactivate
```

Section 3: FastAPI

Getting Started

Install Dependencies:

```bash
pip install fastapi uvicorn[standard]
```

Basic FastAPI App:

```python
from fastapi import FastAPI

app = FastAPI()

@app.get("/")
def read_root():
    return {"Hello": "World"}
```

Starting the Server:

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

Path Operations

Basic Operations:

```python
@app.get("/")        # GET request
@app.post("/posts")  # POST request
@app.put("/posts/{id}") # PUT request
@app.delete("/posts/{id}") # DELETE request
```

Path Parameters:

```python
@app.get("/posts/{id}")
def get_post(id: int):
    return {"post_id": id}
```

⚠️ **Path Order Matters:**

- More specific paths should come before general ones
- `/posts/latest` should be before `/posts/{id}`

HTTP Requests & Postman

Postman Setup:

- Download and install Postman
- Create collections to organize requests
- Save requests for reuse
- Test different HTTP methods

POST Request Example:

```python
from pydantic import BaseModel

class Post(BaseModel):
    title: str
    content: str

@app.post("/posts")
def create_post(post: Post):
    return {"data": post}
```

Schema Validation with Pydantic

Benefits:

- Automatic data validation
- Type conversion
- Error handling
- Documentation generation

Example Model:

```python
class Post(BaseModel):
    title: str
    content: str
    published: bool = True
    rating: Optional[int] = None
```

CRUD Operations

In-Memory Storage (Array):

```python
my_posts = [
    {"title": "Post 1", "content": "Content 1", "id": 1},
    {"title": "Post 2", "content": "Content 2", "id": 2}
]
```

Create Post:

```python
@app.post("/posts")
def create_post(post: Post):
    post_dict = post.dict()
    post_dict['id'] = len(my_posts) + 1
    my_posts.append(post_dict)
    return {"data": post_dict}
```

Get All Posts:

python

```python
@app.get("/posts")
def get_posts():
    return {"data": my_posts}
```

Get Single Post:

python

```python
@app.get("/posts/{id}")
def get_post(id: int):
    post = find_post(id)
    if not post:
        raise HTTPException(status_code=404, detail="Post not found")
    return {"data": post}
```

Update Post:

python

```python
@app.put("/posts/{id}")
def update_post(id: int, post: Post):
    index = find_index_post(id)
    if index == None:
        raise HTTPException(status_code=404, detail="Post not found")

    post_dict = post.dict()
    post_dict['id'] = id
    my_posts[index] = post_dict
    return {"data": post_dict}
```

Delete Post:

```python
@app.delete("/posts/{id}")
def delete_post(id: int):
    index = find_index_post(id)
    if index == None:
        raise HTTPException(status_code=404, detail="Post not found")

    my_posts.pop(index)
    return Response(status_code=status.HTTP_204_NO_CONTENT)
```

Status Codes

```python
from fastapi import status

@app.post("/posts", status_code=status.HTTP_201_CREATED)
@app.delete("/posts/{id}", status_code=status.HTTP_204_NO_CONTENT)
```

Automatic Documentation

- Swagger UI: `http://localhost:8000/docs`
- ReDoc: `http://localhost:8000/redoc`
- Automatically generated from your code

Section 4: Databases

Database Fundamentals

What is a Database?

- Organized collection of data
- Provides efficient storage, retrieval, and management
- Ensures data integrity and consistency

PostgreSQL:

- Open-source relational database
- ACID compliance
- Excellent performance and reliability

PostgreSQL Installation

Windows:

- Download from postgresql.org
- Use installer wizard
- Remember the password for postgres user

Mac:

- Use Homebrew: `brew install postgresql`
- Or download from postgresql.org
- Start service: `brew services start postgresql`

Database Schema & Tables

Creating a Database:

```sql
CREATE DATABASE fastapi;
```

Creating Tables:

```sql
CREATE TABLE posts (
    id SERIAL PRIMARY KEY,
    title VARCHAR NOT NULL,
    content VARCHAR NOT NULL,
    published BOOLEAN NOT NULL DEFAULT TRUE,
    created_at TIMESTAMP WITH TIME ZONE NOT NULL DEFAULT NOW()
);
```

PgAdmin GUI

- Web-based PostgreSQL administration tool
- Visual interface for database management
- Query editor and result viewer
- Object browser for tables, functions, etc.

SQL Queries

Basic SELECT:

sql

```sql
SELECT * FROM posts;
SELECT title, content FROM posts;
```

Filtering with WHERE:

sql

```sql
SELECT * FROM posts WHERE published = true;
SELECT * FROM posts WHERE id = 1;
```

SQL Operators:

sql

```sql
-- Comparison
SELECT * FROM posts WHERE id > 5;
SELECT * FROM posts WHERE id >= 5;
SELECT * FROM posts WHERE id <> 1; -- Not equal

-- Logical
SELECT * FROM posts WHERE published = true AND id > 1;
SELECT * FROM posts WHERE published = true OR id = 1;
```

IN Keyword:

sql

```sql
SELECT * FROM posts WHERE id IN (1, 3, 5);
```

Pattern Matching with LIKE:

sql

```sql
SELECT * FROM posts WHERE title LIKE 'My%';      -- Starts with 'My'
SELECT * FROM posts WHERE title LIKE '%post%';   -- Contains 'post'
SELECT * FROM posts WHERE title LIKE '_ost';     -- Single character wildcard
```

Ordering Results:

sql

```sql
SELECT * FROM posts ORDER BY created_at DESC;
SELECT * FROM posts ORDER BY title ASC;
SELECT * FROM posts ORDER BY id DESC, title ASC;  -- Multiple columns
```

LIMIT & OFFSET (Pagination):

sql

```sql
SELECT * FROM posts LIMIT 5;                -- First 5 records
SELECT * FROM posts LIMIT 5 OFFSET 2;       -- Skip 2, take 5
SELECT * FROM posts ORDER BY id LIMIT 3;    -- Top 3 by id
```

Inserting Data:

sql

```sql
INSERT INTO posts (title, content, published)
VALUES ('My Post', 'This is my content', true);

INSERT INTO posts (title, content)
VALUES ('Another Post', 'More content');  -- published defaults to true

-- Multiple inserts
INSERT INTO posts (title, content, published)
VALUES
    ('Post 1', 'Content 1', true),
    ('Post 2', 'Content 2', false),
    ('Post 3', 'Content 3', true);

-- Return inserted data
INSERT INTO posts (title, content)
VALUES ('New Post', 'New content')
RETURNING *;
```

Updating Data:

sql

```sql
UPDATE posts SET title = 'Updated Title' WHERE id = 1;

UPDATE posts SET
    title = 'New Title',
    content = 'New content',
    published = false
WHERE id = 2;

-- Update multiple records
UPDATE posts SET published = true WHERE published = false;

-- Return updated data
UPDATE posts SET title = 'Updated' WHERE id = 1 RETURNING *;
```

Deleting Data:

sql

```sql
DELETE FROM posts WHERE id = 1;
DELETE FROM posts WHERE published = false;

-- Return deleted data
DELETE FROM posts WHERE id = 1 RETURNING *;

-- Delete all (be careful!)
DELETE FROM posts;
```

Key Concepts

Primary Key:

- Unique identifier for each row
- Cannot be NULL
- Often uses SERIAL (auto-incrementing integer)

Data Types:

- `VARCHAR`: Variable character string
- `INTEGER`: Whole numbers
- `BOOLEAN`: True/false

- `TIMESTAMP`: Date and time
- `SERIAL`: Auto-incrementing integer

Constraints:

- `NOT NULL`: Field cannot be empty
- `PRIMARY KEY`: Unique identifier
- `DEFAULT`: Default value if none provided
- `UNIQUE`: Value must be unique across table

Best Practices

1. **Always use WHERE clauses** with UPDATE and DELETE
2. **Use RETURNING clause** to see affected data
3. **Be careful with DELETE and UPDATE** operations
4. **Use meaningful column names**
5. **Plan your schema** before creating tables
6. **Use appropriate data types**
7. **Consider indexing** for frequently queried columns

Common Mistakes to Avoid

1. Forgetting WHERE clause in UPDATE/DELETE
2. Not handling NULL values properly
3. Using wrong data types
4. Not planning for data growth
5. Ignoring case sensitivity in string comparisons
6. Not backing up data before major operations