FastAPI Section 16: Testing

1. Testing Introduction

Why Testing is Important

- **Catch bugs early** in development
- **Prevent regressions** when adding new features
- **Improve code quality** and maintainability
- **Build confidence** in your application
- **Documentation** through test cases

Types of Testing

- **Unit Tests**: Test individual functions/methods
- **Integration Tests**: Test component interactions
- **End-to-End Tests**: Test complete user workflows
- **API Tests**: Test HTTP endpoints

2. pytest Setup

Installation

bash

```bash
pip install pytest
pip install pytest-asyncio  # For async tests
```

Basic Test Structure

python

```python
# test_basic.py
def test_addition():
    assert 2 + 2 == 4

def test_string_operations():
    name = "FastAPI"
    assert name.lower() == "fastapi"
    assert len(name) == 7
```

Running Tests

bash

```
# Run all tests
pytest

# Run specific test file
pytest test_basic.py

# Run specific test function
pytest test_basic.py::test_addition
```

3. pytest Flags and Options

Common Flags

bash

```
# Verbose output (-v)
pytest -v

# Show print statements (-s)
pytest -s

# Stop on first failure (-x)
pytest -x

# Run last failed tests (--lf)
pytest --lf

# Show coverage
pytest --cov=app

# Run tests in parallel
pytest -n auto  # requires pytest-xdist
```

Configuration File (pytest.ini)

ini

```ini
[tool:pytest]
testpaths = tests
python_files = test_*.py
python_classes = Test*
python_functions = test_*
addopts = -v --strict-markers
markers =
    slow: marks tests as slow
    integration: marks tests as integration tests
```

4. Testing Functions

Simple Function Testing

python

```python
# app/calculations.py
def add(x: int, y: int) -> int:
    return x + y

def divide(x: int, y: int) -> float:
    if y == 0:
        raise ValueError("Cannot divide by zero")
    return x / y

# tests/test_calculations.py
import pytest
from app.calculations import add, divide

def test_add():
    assert add(5, 3) == 8
    assert add(-1, 1) == 0

def test_divide():
    assert divide(10, 2) == 5.0
    assert divide(7, 2) == 3.5
```

5. Parametrize

Basic Parametrization

```python
import pytest

@pytest.mark.parametrize("x,y,expected", [
    (5, 3, 8),
    (2, 7, 9),
    (-1, 1, 0),
    (0, 0, 0),
])
def test_add_parametrized(x, y, expected):
    assert add(x, y) == expected

@pytest.mark.parametrize("x,y", [
    (10, 2),
    (15, 3),
    (100, 4),
])
def test_divide_parametrized(x, y):
    result = divide(x, y)
    assert result == x / y
```

Advanced Parametrization

```python
@pytest.mark.parametrize("username,password,expected_status", [
    ("valid_user", "correct_password", 200),
    ("invalid_user", "any_password", 401),
    ("valid_user", "wrong_password", 401),
    ("", "password", 422),
])
def test_login_scenarios(username, password, expected_status):
    # Test implementation
    pass
```

6. Testing Classes

Class-Based Tests

```python
class TestBankAccount:
    def test_initial_balance(self):
        account = BankAccount(100)
        assert account.balance == 100

    def test_deposit(self):
        account = BankAccount(100)
        account.deposit(50)
        assert account.balance == 150

    def test_withdraw(self):
        account = BankAccount(100)
        account.withdraw(30)
        assert account.balance == 70

    def test_withdraw_insufficient_funds(self):
        account = BankAccount(50)
        with pytest.raises(ValueError):
            account.withdraw(100)
```

7. Fixtures

Basic Fixtures

python

```python
import pytest
from app.database import get_db
from app.models import User

@pytest.fixture
def test_user():
    return {
        "email": "test@example.com",
        "password": "testpassword"
    }

@pytest.fixture
def sample_posts():
    return [
        {"title": "First Post", "content": "First content"},
        {"title": "Second Post", "content": "Second content"},
    ]

def test_user_creation(test_user):
    assert test_user["email"] == "test@example.com"
    assert test_user["password"] == "testpassword"
```

Database Fixtures

python

```python
@pytest.fixture
def session():
    # Create test database session
    engine = create_engine("sqlite:///test.db")
    TestingSessionLocal = sessionmaker(bind=engine)
    Base.metadata.create_all(bind=engine)

    db = TestingSessionLocal()
    try:
        yield db
    finally:
        db.close()
        Base.metadata.drop_all(bind=engine)
```

8. Fixture Scope

Scope Options

```python
@pytest.fixture(scope="function")  # Default - runs for each test
def function_fixture():
    return "function scope"

@pytest.fixture(scope="class")  # Runs once per test class
def class_fixture():
    return "class scope"

@pytest.fixture(scope="module")  # Runs once per test module
def module_fixture():
    return "module scope"

@pytest.fixture(scope="session")  # Runs once per test session
def session_fixture():
    return "session scope"
```

Practical Example

```python
@pytest.fixture(scope="module")
def database_connection():
    # Expensive setup - only do once per module
    connection = create_test_database()
    yield connection
    connection.close()

@pytest.fixture
def clean_database(database_connection):
    # Clean database before each test
    database_connection.execute("DELETE FROM users")
    database_connection.execute("DELETE FROM posts")
    database_connection.commit()
    yield database_connection
```

9. Testing Exceptions

Exception Testing

python

```python
def test_divide_by_zero():
    with pytest.raises(ValueError):
        divide(10, 0)

def test_divide_by_zero_with_message():
    with pytest.raises(ValueError, match="Cannot divide by zero"):
        divide(10, 0)

def test_exception_info():
    with pytest.raises(ValueError) as exc_info:
        divide(10, 0)

    assert "Cannot divide by zero" in str(exc_info.value)
```

10. FastAPI TestClient

Basic Setup

python

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

client = TestClient(app)

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

Testing with Fixtures

python

```python
@pytest.fixture
def client():
    return TestClient(app)

def test_get_posts(client):
    response = client.get("/posts")
    assert response.status_code == 200
    assert isinstance(response.json(), list)
```

11. Testing Database Operations

Test Database Setup

python

```python
import pytest
from sqlalchemy import create_engine
from sqlalchemy.orm import sessionmaker
from app.database import get_db, Base
from app.main import app

# Test database URL
SQLALCHEMY_DATABASE_URL = "sqlite:///./test.db"

engine = create_engine(SQLALCHEMY_DATABASE_URL, connect_args={"check_same_thread": False})
TestingSessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)

@pytest.fixture
def session():
    Base.metadata.drop_all(bind=engine)
    Base.metadata.create_all(bind=engine)
    db = TestingSessionLocal()
    try:
        yield db
    finally:
        db.close()

@pytest.fixture
def client(session):
    def override_get_db():
        try:
            yield session
        finally:
            session.close()

    app.dependency_overrides[get_db] = override_get_db
    yield TestClient(app)
    app.dependency_overrides.clear()
```

12. User Authentication Tests

User Registration Test

python

```python
def test_create_user(client):
    response = client.post(
        "/users/",
        json={"email": "test@example.com", "password": "testpassword"}
    )
    assert response.status_code == 201
    data = response.json()
    assert data["email"] == "test@example.com"
    assert "id" in data
    assert "password" not in data  # Password should not be returned

def test_create_user_duplicate_email(client):
    # Create first user
    client.post(
        "/users/",
        json={"email": "test@example.com", "password": "password1"}
    )

    # Try to create user with same email
    response = client.post(
        "/users/",
        json={"email": "test@example.com", "password": "password2"}
    )
    assert response.status_code == 400
```

Login Tests

python

```python
def test_login_success(client, test_user):
    # Create user first
    client.post("/users/", json=test_user)

    # Login
    response = client.post(
        "/login",
        data={"username": test_user["email"], "password": test_user["password"]}
    )
    assert response.status_code == 200
    data = response.json()
    assert "access_token" in data
    assert data["token_type"] == "bearer"

def test_login_invalid_credentials(client):
    response = client.post(
        "/login",
        data={"username": "wrong@example.com", "password": "wrongpassword"}
    )
    assert response.status_code == 401
```

13. Testing with Authentication

Authenticated User Fixture

python

```python
@pytest.fixture
def test_user(client):
    user_data = {"email": "test@example.com", "password": "testpassword"}
    response = client.post("/users/", json=user_data)
    assert response.status_code == 201
    return response.json()

@pytest.fixture
def token(client, test_user):
    response = client.post(
        "/login",
        data={"username": test_user["email"], "password": "testpassword"}
    )
    return response.json()["access_token"]

@pytest.fixture
def authorized_client(client, token):
    client.headers = {
        **client.headers,
        "Authorization": f"Bearer {token}"
    }
    return client
```

14. Posts Testing

Create Posts Fixture

python

```python
@pytest.fixture
def test_posts(test_user, session):
    posts_data = [
        {"title": "First Post", "content": "First content", "owner_id": test_user["id"]},
        {"title": "Second Post", "content": "Second content", "owner_id": test_user["id"]},
        {"title": "Third Post", "content": "Third content", "owner_id": test_user["id"]},
    ]

    posts = []
    for post_data in posts_data:
        post = Post(**post_data)
        session.add(post)
        posts.append(post)

    session.commit()
    return posts
```

Post CRUD Tests

python

```python
def test_get_all_posts(authorized_client, test_posts):
    response = authorized_client.get("/posts")
    assert response.status_code == 200
    assert len(response.json()) == len(test_posts)

def test_unauthorized_get_all_posts(client):
    response = client.get("/posts")
    assert response.status_code == 401

def test_get_one_post(authorized_client, test_posts):
    response = authorized_client.get(f"/posts/{test_posts[0].id}")
    assert response.status_code == 200
    post = response.json()
    assert post["Post"]["id"] == test_posts[0].id
    assert post["Post"]["title"] == test_posts[0].title

def test_get_nonexistent_post(authorized_client):
    response = authorized_client.get("/posts/99999")
    assert response.status_code == 404

def test_create_post(authorized_client, test_user):
    post_data = {"title": "New Post", "content": "New content"}
    response = authorized_client.post("/posts", json=post_data)
    assert response.status_code == 201
    created_post = response.json()
    assert created_post["title"] == post_data["title"]
    assert created_post["content"] == post_data["content"]
    assert created_post["owner_id"] == test_user["id"]

def test_delete_post(authorized_client, test_posts):
    response = authorized_client.delete(f"/posts/{test_posts[0].id}")
    assert response.status_code == 204

def test_delete_nonexistent_post(authorized_client):
    response = authorized_client.delete("/posts/99999")
    assert response.status_code == 404

def test_update_post(authorized_client, test_posts):
    updated_data = {"title": "Updated Title", "content": "Updated content"}
    response = authorized_client.put(f"/posts/{test_posts[0].id}", json=updated_data)
    assert response.status_code == 200
    updated_post = response.json()
```

```
    assert updated_post["title"] == updated_data["title"]
    assert updated_post["content"] == updated_data["content"]
```

15. Voting System Tests

Vote Tests

python

```python
def test_vote_on_post(authorized_client, test_posts):
    response = authorized_client.post(
        "/vote",
        json={"post_id": test_posts[0].id, "dir": 1}
    )
    assert response.status_code == 201
    assert response.json()["message"] == "Successfully added vote"

def test_vote_twice_on_same_post(authorized_client, test_posts):
    # First vote
    authorized_client.post(
        "/vote",
        json={"post_id": test_posts[0].id, "dir": 1}
    )

    # Second vote (should fail)
    response = authorized_client.post(
        "/vote",
        json={"post_id": test_posts[0].id, "dir": 1}
    )
    assert response.status_code == 409

def test_delete_vote(authorized_client, test_posts):
    # Add vote first
    authorized_client.post(
        "/vote",
        json={"post_id": test_posts[0].id, "dir": 1}
    )

    # Remove vote
    response = authorized_client.post(
        "/vote",
        json={"post_id": test_posts[0].id, "dir": 0}
    )
    assert response.status_code == 201
    assert response.json()["message"] == "Successfully removed vote"

def test_vote_nonexistent_post(authorized_client):
    response = authorized_client.post(
        "/vote",
        json={"post_id": 99999, "dir": 1}
```

```
    )
    assert response.status_code == 404
```

16. conftest.py

Centralized Configuration

python

```python
# tests/conftest.py
import pytest
from fastapi.testclient import TestClient
from sqlalchemy import create_engine
from sqlalchemy.orm import sessionmaker

from app.main import app
from app.database import get_db, Base
from app.models import User, Post

SQLALCHEMY_DATABASE_URL = "sqlite:///./test.db"

engine = create_engine(
    SQLALCHEMY_DATABASE_URL,
    connect_args={"check_same_thread": False}
)
TestingSessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)

@pytest.fixture
def session():
    Base.metadata.drop_all(bind=engine)
    Base.metadata.create_all(bind=engine)
    db = TestingSessionLocal()
    try:
        yield db
    finally:
        db.close()

@pytest.fixture
def client(session):
    def override_get_db():
        try:
            yield session
        finally:
            session.close()

    app.dependency_overrides[get_db] = override_get_db
    yield TestClient(app)
    app.dependency_overrides.clear()

@pytest.fixture
def test_user_data():
    return {"email": "test@example.com", "password": "testpassword"}
```

```python
@pytest.fixture
def test_user(client, test_user_data):
    response = client.post("/users/", json=test_user_data)
    assert response.status_code == 201
    return response.json()

@pytest.fixture
def token(client, test_user_data):
    response = client.post(
        "/login",
        data={"username": test_user_data["email"], "password": test_user_data["password"]}
    )
    return response.json()["access_token"]

@pytest.fixture
def authorized_client(client, token):
    client.headers = {
        **client.headers,
        "Authorization": f"Bearer {token}"
    }
    return client
```

Best Practices

Test Organization

```
tests/
├── conftest.py
├── test_auth.py
├── test_posts.py
├── test_users.py
├── test_votes.py
└── test_calculations.py
```

Naming Conventions

- Test files: `test_*.py`
- Test functions: `test_*`
- Test classes: `Test*`
- Fixtures: descriptive names without `test_` prefix

Test Writing Tips

1. **AAA Pattern**: Arrange, Act, Assert
2. **One assertion per test** (when possible)
3. **Descriptive test names** that explain what is being tested
4. **Test edge cases** and error conditions
5. **Use fixtures** to avoid code duplication
6. **Keep tests independent** - each test should be able to run alone

Coverage Goals

- Aim for **80%+ code coverage**
- Focus on **critical business logic**
- Test **error handling** and **edge cases**
- Don't sacrifice **test quality** for coverage percentage

Running Tests in CI/CD

yaml

```yaml
# .github/workflows/test.yml
name: Tests
on: [push, pull_request]
jobs:
  test:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v2
      - name: Set up Python
        uses: actions/setup-python@v2
        with:
          python-version: 3.9
      - name: Install dependencies
        run: |
          pip install -r requirements.txt
      - name: Run tests
        run: |
          pytest --cov=app --cov-report=html
```