FastAPI Testing with Pytest - Complete Guide

1. Introduction

This document provides a comprehensive guide to testing FastAPI applications using Pytest. It covers API development, unit testing, fixtures, parameterized tests, and mocking.

2. FastAPI Application (main.py)

The API provides endpoints to fetch and create users. It uses Pydantic models for validation.

```
@app.get("/users/{user_id}")
def get_user(user_id: int):
    if user_id in users_db:
        return users_db[user_id]
    raise HTTPException(status_code=404, detail="User not found")
```

3. Unit Testing with Pytest

Unit tests help verify API functionality. Pytest provides a framework for writing and running tests efficiently.

3.1 FastAPI TestClient

TestClient is used to send requests to FastAPI endpoints without running a real server.

```
client = TestClient(app)
```

3.2 Pytest Fixtures

Fixtures allow setting up reusable test environments. Example:

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

3.3 Parameterized Tests

Parameterized tests allow running the same test with different inputs.

```
@pytest.mark.parametrize("user_data, expected_status", [
    ({"name": "David", "email": "david@example.com"}, 201),
    ({}, 422)
])
def test_create_user_various_cases(test_client, user_data, expected_status):
    response = test_client.post("/users", json=user_data)
    assert response.status_code == expected_status
```

3.4 Mocking External Dependencies

Mocking replaces real database/API calls with fake data for testing.

```
@patch("main.users_db", {1: {"name": "Mocked User", "email": "mock@example.com"}})
def test_mocked_get_user(test_client):
    response = test_client.get("/users/1")
    assert response.status_code == 200
    assert response.json()["name"] == "Mocked User"
```

4. Steps to Run Tests

1. Install dependencies: `pip install fastapi uvicorn pytest httpx`

2. Run tests using: `pytest -v`

3. Verify the output. All tests should pass.