# Week 8: Testing + Jenkins CI

## CSI403 Full Stack Development | Lab 7 (8%)

Semester 1/2569

# Agenda

1. Why Testing?
2. pytest Basics
3. Writing Tests
4. Running Tests
5. CI/CD Introduction
6. Jenkins Setup
7. Jenkinsfile
8. Lab 7 Assignment

# Why Write Tests?

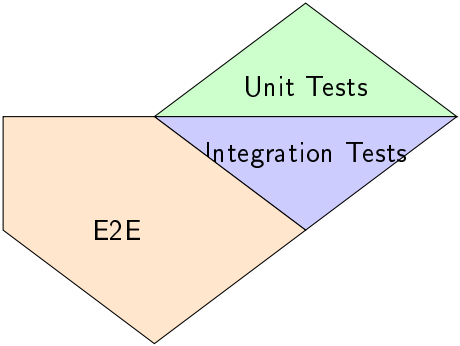
**Benefits:**

- Catch bugs early
- Confidence to refactor
- Documentation of behavior
- Prevent regressions

**Required for:**

- CI/CD pipelines
- Code reviews
- Team collaboration
- Production deployment

# Types of Tests




- **Unit Tests:** Single function/method (fast, many)
- **Integration Tests:** Multiple components together
- **E2E Tests:** Full user flow (slow, few)

# pytest Framework

**Most popular Python testing framework**

- Simple syntax
- Powerful fixtures
- Rich plugin ecosystem
- Good error messages
- Works with FastAPI TestClient

# Install pytest

```
# Install pytest and related packages
pip install pytest pytest-cov httpx

# Add to requirements.txt
pytest==7.4.4
pytest-cov==4.1.0
httpx==0.26.0
```

# Test File Structure

```
taskflow/
  app/
    main.py
    routes/
      tasks.py
  tests/
    __init__.py
    conftest.py           # Shared fixtures
    test_tasks.py         # Task API tests
    test_users.py         # User API tests
    test_categories.py    # Category tests
```

# conftest.py - Test Database

```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 Base, get_db

# Use in-memory SQLite for tests
TEST_DATABASE_URL = "sqlite:///:memory:"
engine = create_engine(TEST_DATABASE_URL)
TestingSessionLocal = sessionmaker(bind=engine)
```

# conftest.py - Client Fixture

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

@pytest.fixture(scope="function")
def client(db):
    def override_get_db():
        yield db
    app.dependency_overrides[get_db] = override_get_db
    with TestClient(app) as c:
        yield c
    app.dependency_overrides.clear()
```

# Simple Test

```python
# tests/test_tasks.py

def test_root(client):
    """Test root endpoint"""
    response = client.get("/")
    assert response.status_code == 200
    assert "message" in response.json()

def test_health(client):
    """Test health check"""
    response = client.get("/health")
    assert response.status_code == 200
    assert response.json()["status"] == "healthy"
```

# Test Create Task

```python
def test_create_task(client):
    """Test creating a new task"""
    task_data = {
        "title": "Test Task",
        "priority": "high"
    }

    response = client.post("/api/tasks", json=task_data)

    assert response.status_code == 201
    data = response.json()
    assert data["title"] == "Test Task"
    assert data["priority"] == "high"
    assert data["status"] == "pending"
    assert "id" in data
```

# Test Get Task

```python
def test_get_task(client):
    """Test getting a single task"""
    # First create a task
    create_response = client.post("/api/tasks",
        json={"title": "Test"})
    task_id = create_response.json()["id"]

    # Then get it
    response = client.get(f"/api/tasks/{task_id}")

    assert response.status_code == 200
    assert response.json()["title"] == "Test"

def test_get_task_not_found(client):
    """Test getting non-existent task"""
    response = client.get("/api/tasks/9999")
    assert response.status_code == 404
```

# Test Update Task

```python
def test_update_task(client):
    """Test updating a task"""
    # Create task
    create_resp = client.post("/api/tasks",
        json={"title": "Original"})
    task_id = create_resp.json()["id"]

    # Update task
    update_data = {"title": "Updated", "status": "done"}
    response = client.put(f"/api/tasks/{task_id}",
        json=update_data)

    assert response.status_code == 200
    assert response.json()["title"] == "Updated"
    assert response.json()["status"] == "done"
```

# Test Delete Task

```python
def test_delete_task(client):
    """Test deleting a task"""
    # Create task
    create_resp = client.post("/api/tasks",
        json={"title": "To␣Delete"})
    task_id = create_resp.json()["id"]

    # Delete task
    response = client.delete(f"/api/tasks/{task_id}")
    assert response.status_code == 204

    # Verify deleted
    get_response = client.get(f"/api/tasks/{task_id}")
    assert get_response.status_code == 404
```

# Test Validation Error

```python
def test_create_task_validation_error(client):
    """Test validation error for empty title"""
    response = client.post("/api/tasks",
        json={"title": ""})

    assert response.status_code == 422

def test_create_task_missing_title(client):
    """Test validation error for missing title"""
    response = client.post("/api/tasks",
        json={"priority": "high"})

    assert response.status_code == 422
```

# Running Tests

```
# Run all tests
pytest

# Run with verbose output
pytest -v

# Run specific file
pytest tests/test_tasks.py

# Run specific test
pytest tests/test_tasks.py::test_create_task

# Run tests matching pattern
pytest -k "create"
```

# Code Coverage

```
# Run with coverage report
pytest --cov=app

# Generate HTML report
pytest --cov=app --cov-report=html

# Show missing lines
pytest --cov=app --cov-report=term-missing

# Generate JUnit XML (for Jenkins)
pytest --junitxml=test-results.xml
```

# What is CI/CD?


- **CI (Continuous Integration):** Auto build and test on every push
- **CD (Continuous Deployment):** Auto deploy to production

# Why CI/CD?

- Catch bugs before they reach production
- Automated testing on every change
- Faster feedback for developers
- Consistent build process
- Reliable deployments
- Industry standard practice

# What is Jenkins?

**Open-source automation server**

- Build, test, and deploy code
- Pipeline as code (Jenkinsfile)
- Hundreds of plugins
- Web-based UI
- Free and widely used

# Jenkins Setup Steps

1. Start Jenkins container (from docker-compose)
2. Access http://localhost:8080
3. Get initial admin password
4. Install suggested plugins
5. Create admin user
6. Create Pipeline job
7. Connect to GitHub repository

# Get Jenkins Password

```
# Get initial admin password
docker exec taskflow-jenkins \
    cat /var/jenkins_home/secrets/initialAdminPassword

# Or view in logs
docker logs taskflow-jenkins 2>&1 | grep -A 5 "initialAdminPassword"
```

# Jenkinsfile Structure

```
pipeline {
    agent any

    stages {
        stage('Stage Name') {
            steps {
                // Commands here
            }
        }
    }

    post {
        always { /* cleanup */ }
        success { /* notify */ }
        failure { /* alert */ }
    }
}
```

# Jenkinsfile: Checkout Stage

```
pipeline {
    agent any

    stages {
        stage('Checkout') {
            steps {
                checkout scm
            }
        }
    }
}
```

# Jenkinsfile: Install Stage

```
        stage('Install Dependencies') {
            steps {
                sh '''
                    python3 -m venv venv
                    . venv/bin/activate
                    pip install -r requirements.txt
                    pip install pytest pytest-cov
                '''
            }
        }
```

# Jenkinsfile: Test Stage

```
        stage('Run Tests') {
            steps {
                sh '''
. venv/bin/activate
pytest --junitxml=test-results.xml \
--cov=app \
--cov-report=xml \
-v
'''
            }
            post {
                always {
                    junit 'test-results.xml'
                }
            }
        }
```

# Lab 7: Testing + Jenkins CI (8%)

**Requirements:**

1. Create tests/conftest.py with fixtures
2. Write at least 8 test cases
3. Achieve 60%+ code coverage
4. Create Jenkinsfile with CI stages
5. Pipeline runs successfully in Jenkins

# Lab 7 Grading Rubric

| Criteria | Points |
|---|---|
| conftest.py with fixtures | 1.5% |
| Test cases (8+ tests) | 2.5% |
| Code coverage 60%+ | 1% |
| Jenkinsfile (CI stages) | 2% |
| Pipeline runs successfully | 1% |
| **Total** | **8%** |

# Questions?

Write tests for your API!

Next Week: Jenkins CD (Last Lab!)