Minimal To-Do Application – SDLC & DevOps Report

Author: Omar El Hajj Chehade

Course: DevOps Pipeline Design – Individual Assignment 1

Date: October 2025

1. Introduction & Objective

The goal of this project is to design and implement a minimal To-Do List Manager application that can later be integrated into a DevOps pipeline. This application allows users to create, view, update, and delete tasks, providing persistent storage through SQLite and a simple HTML/JavaScript interface. The project demonstrates practical application of the Software Development Life Cycle (SDLC) and prepares the groundwork for future DevOps pipeline automation.

2. Chosen SDLC Model & Justification

The Agile SDLC model was selected for this project due to its flexibility and iterative development style. Agile allows short cycles of implementation, testing, and feedback. Each iteration added features: first CRUD operations, then persistence, frontend, and finally Docker + CI integration. This approach was efficient for a small-scale project, allowing continuous improvement and adaptive planning.

3. SDLC Phases

Planning: Defined the problem, scope, and tools (FastAPI, SQLite, Docker, GitHub). Requirements: Specified CRUD functionality, database persistence, and RESTful design. Design: Created architecture and UML diagrams. Designed modular structure. Implementation: Developed the backend, database, and frontend. Testing: Verified endpoints via Swagger UI and manual frontend checks. Deployment: Added Dockerfile and GitHub Actions workflow. Maintenance: Structured for future pipeline automation and monitoring.

4. Requirements

Functional: 1. Users can create new tasks. 2. Users can view all tasks. 3. Users can edit or mark tasks complete. 4. Users can delete tasks. 5. Data persists in SQLite.
Non-Functional: - Quick response time (<1s per request) - Clean, modular, readable code - Platform-independent execution via Docker

5. User Stories & Acceptance Criteria

User Story 1: As a user, I can create tasks with a title. Acceptance: Task appears immediately after submission. User Story 2: As a user, I can view all tasks. Acceptance: Task list loads automatically on page load. User Story 3: As a user, I can toggle completion. Acceptance: Checkbox updates database state. User Story 4: As a user, I can delete tasks. Acceptance: Task removed permanently from both UI and DB.

6. Architecture Overview

The architecture follows a five-layer modular design: 1. Frontend: index.html manages UI and user interactions. 2. Router: FastAPI routes handle HTTP requests. 3. CRUD Logic: Business logic for task operations. 4. ORM: SQLAlchemy manages data mapping. 5. SQLite Database: Stores persistent data. This separation simplifies development, debugging, and future scalability.

7. UML/Class Diagram

Class: Task Attributes: - id (Integer, Primary Key) - title (String) - completed (Boolean) - created_at (Datetime) - updated_at (Datetime) This single model is central to the CRUD system, representing the To-Do entity managed through FastAPI.

8. Implementation Details

Technologies Used: - Python 3.11 + FastAPI Framework - SQLite + SQLAlchemy ORM - HTML + JavaScript frontend - Docker & GitHub Actions for DevOps Key Endpoints: - GET /api/tasks → Retrieve all tasks - POST /api/tasks → Create a task - PUT /api/tasks/{id} → Update task - DELETE /api/tasks/{id} → Delete task - GET /api/health → Server health check

9. Version Control

Version control was implemented using Git and GitHub with at least three meaningful commits: 1. feat: add CRUD backend and minimal frontend 2. docs: add UML diagrams and SDLC report 3. chore: add Dockerfile and CI workflow This commit history demonstrates structured and progressive development.

10. Reflection: DevOps Adaptation

The To-Do app is designed for easy DevOps integration. It includes a Dockerfile for consistent deployment and a GitHub Actions workflow for CI testing. The modular FastAPI codebase supports continuous integration, containerization, and future scalability to cloud platforms. Potential enhancements include automated testing (pytest), environment-based configuration, and deployment through container orchestration tools.

11. Setup Instructions

Local Setup: 1. python -m venv .venv 2. source .venv/bin/activate (Windows: .venv\Scripts\Activate.ps1) 3. pip install -r requirements.txt 4. uvicorn app.main:app --reload Access at: http://127.0.0.1:8000/ Docker Setup: 1. docker build -t todo-fastapi . 2. docker run -p 8000:8000 todo-fastapi

12. Conclusion

This project effectively applies the SDLC process and Agile principles to develop a small but complete web application. It provides a practical foundation for future DevOps pipeline integration, CI/CD, and cloud deployment. The result is a maintainable, documented, and ready-to-scale micro-application demonstrating modern development standards.

13. References

- FastAPI Documentation: https://fastapi.tiangolo.com/ - SQLAlchemy ORM: https://www.sqlalchemy.org/ - Python 3.11: https://docs.python.org/3/ - Docker Docs: https://docs.docker.com/ - GitHub Actions: https://docs.github.com/actions