2025 Assignment 2 - Recording API

Project Information

Repository URL: https://gitlab.com/f.saponara/2025_assignment2_python_pipeline_group32

Application: FastAPI User and Todo Management System with CRUD Operations

Programming Language: Python 3.13

Team Members:

- Francesco Saponara
- Cristina M. Stanculea
- Mattia Chittoni

Project Overview

This project implements a complete CI/CD pipeline for a **FastAPI-based User and Todo Management API** that provides full CRUD (Create, Read, Update, Delete) operations for two entities. The application uses SQLAlchemy ORM with SQLite database and is fully containerized using Docker.

The focus of this assignment is on the **GitLab CI/CD pipeline**, which automates the entire development workflow from code compilation to deployment.

Application Description

Features

- RESTful API built with FastAPI framework
- **User CRUD operations** (Create, Read, Update, Delete)
- **Todo CRUD operations** (Create, Read, Update, Delete, Toggle completion)
- SQLite database with SQLAlchemy ORM
- Pydantic models for data validation
- Comprehensive unit and integration tests
- Docker containerization
- Automated CI/CD pipeline

API Endpoints

User Endpoints

- `GET /api/v1/users` - List all users
- `POST /api/v1/users` - Create a new user
- `GET /api/v1/users/{id}` - Get a specific user
- `PUT /api/v1/users/{id}` - Update a user
- `DELETE /api/v1/users/{id}` - Delete a user

Todo Endpoints

- `GET /api/v1/todos` - List all todos (with optional completed filter)
- `POST /api/v1/todos` - Create a new todo
- `GET /api/v1/todos/{id}` - Get a specific todo
- `PUT /api/v1/todos/{id}` - Update a todo
- `DELETE /api/v1/todos/{id}` - Delete a todo
- `PATCH /api/v1/todos/{id}/toggle` - Toggle todo completion status

Technology Stack

- **Framework:** FastAPI 0.116.1
- **Database:** SQLite with SQLAlchemy 2.0.42
- **Web Server:** Uvicorn 0.35.0
- **Testing:** pytest 8.4.1
- **Containerization:** Docker
- **CI/CD:** GitLab CI/CD

Pipeline Stages

The CI/CD pipeline consists of **6 stages** as required by the assignment:

Stage 1: BUILD 🔨

Purpose: Resolve dependencies and prepare the build environment.

Implementation:

- Uses Python 3.13 Docker image
- Installs pip and upgrades to the latest version
- Installs all project dependencies from `requirements.txt`
- Creates a virtual environment (`.venv/`)
- Caches pip packages to speed up subsequent builds

Key Commands:

```
pip install --upgrade pip
pip install -r requirements.txt
python -m venv .venv
```

Artifacts: Virtual environment stored for subsequent stages

Duration: ~2-3 minutes

Stage 2: VERIFY 🔍

Purpose: Run static and dynamic analysis to ensure code quality and security.

IMPORTANT: This stage consists of **two jobs running in parallel** as required:

Job 2.1: Static Analysis (verify:static)

Tool: Prospector (aggregates pylint, pep8, pyflakes, mccabe, pyroma)

What it checks:

- Code style compliance (PEP 8)
- Code complexity metrics (McCabe)
- Potential bugs and code smells
- Best practices violations
- Documentation quality

Command:

```
prospector app/ --output-format grouped
```

Configuration: Uses default Prospector configuration

Job 2.2: Dynamic/Security Analysis (verify:security)

Tool: Bandit

What it checks:

- Common security vulnerabilities
- Hardcoded passwords and secrets
- Insecure function usage
- SQL injection vulnerabilities
- Use of unsafe functions

Command:

```
bandit -r app/ -f json -o bandit-report.json -ll
```

Artifacts: Security report in JSON format (`bandit-report.json`)

Both jobs run simultaneously to reduce pipeline execution time.

Duration: ~3-4 minutes (parallel execution)

Stage 3: TEST 🧪

Purpose: Run unit and integration tests with code coverage analysis.

Implementation:

- Uses pytest as the testing framework
- Runs all tests in the `tests/` directory
- Generates code coverage reports in multiple formats (terminal, HTML, XML)
- Uses in-memory SQLite database for testing
- Includes FastAPI TestClient for endpoint testing

Command:

```
pytest tests/ -v --cov=app --cov-report=term --cov-report=html --cov-report=xml
```

Test Coverage:

- API endpoint tests (create, read, update, delete operations)
- Service layer tests
- Database integration tests
- Error handling tests

Artifacts:

- HTML coverage report (`htmlcov/`)
- XML coverage report (`coverage.xml`)
- Cobertura format for GitLab integration

Duration: ~1-2 minutes

Stage 4: PACKAGE 📦

Purpose: Create distributable Python packages ready for release.

Implementation:

- Uses Python's `build` module with setuptools and wheel
- Creates both source distribution and wheel distribution
- Follows PEP 517/518 standards
- Uses configuration from `pyproject.toml`

Command:

```
python -m build
```

Outputs:

- **Source Archive:** `recording-0.1.0.tar.gz`
- **Built Distribution (Wheel):** `recording-0.1.0-py3-none-any.whl`

Artifacts: Distribution packages stored in `dist/` directory

Duration: ~1 minute

Stage 5: RELEASE 🚀

Purpose: Build Docker image and publish to GitLab Container Registry.

Implementation:

- Builds Docker image from the project's Dockerfile
- Tags image with commit reference and 'latest' tag
- Pushes to GitLab Container Registry
- Uses Docker-in-Docker (dind) service

Commands:

```
docker build -t $CI_REGISTRY_IMAGE:$CI_COMMIT_REF_SLUG .
docker push $CI_REGISTRY_IMAGE:$CI_COMMIT_REF_SLUG
docker push $CI_REGISTRY_IMAGE:latest
```

Docker Image Configuration:

- Base image: Python 3.13-slim-bookworm
- Uses `pip` for fast dependency installation

- Exposes port 80
- Runs with Uvicorn ASGI server

Registry Location: `registry.gitlab.com/f.saponara/2025_assignment2_python_pipeline_group32`

Trigger Conditions: Only runs on `main` branch and tags

Duration: ~3-5 minutes

Stage 6: DOCS 📚

Purpose: Generate and publish project documentation to GitLab Pages.

Implementation:

- Uses MkDocs with Material theme
- Converts Markdown files to static HTML website
- Includes API documentation, installation guide, and CI/CD details
- Publishes to GitLab Pages

Command:

```
mkdocs build --strict --verbose
mv site public
```

Documentation Structure:

```
docs/
├── index.md                 # Home page
├── getting-started/
│   ├── installation.md      # Installation guide
│   └── quickstart.md        # Quick start tutorial
├── api/
│   ├── user-endpoints.md    # User endpoint docs
└──   └── todo-endpoints.md    # Todo endpoint docs
```

Features:

- Material Design theme
- Search functionality
- Dark/light mode toggle
- Code syntax highlighting

- Responsive design

Published URL: `https://f.saponara.gitlab.io/2025_assignment2_python_pipeline_group32`

Trigger Conditions: Only runs on `main` branch

Duration: ~1-2 minutes

Commands:

```
ssh $DEPLOY_USER@$DEPLOY_HOST << 'EOF'
  docker pull $DOCKER_IMAGE_NAME
  docker stop recording-app || true
  docker rm recording-app || true
  docker run -d --name recording-app -p 8000:80 $DOCKER_IMAGE_NAME
  docker image prune -f
EOF
```

Required GitLab CI/CD Variables:

- `SSH_PRIVATE_KEY`: Private SSH key for authentication
- `DEPLOY_HOST`: Production server hostname/IP
- `STAGING_HOST`: Staging server hostname/IP
- `DEPLOY_USER`: SSH username

Security: SSH key-based authentication, no password storage

Duration: ~2-3 minutes

Pipeline Configuration

Triggers and Branches

The pipeline behavior varies by branch:

- `main` **branch:** Runs all 7 stages (full pipeline)
- `develop` **branch:** Runs stages 1-6 (excludes production deploy)
- **Merge Requests:** Runs stages 1-4 (build, verify, test, package)
- **Tags:** Runs stages 1-5 (includes release)

Manual Jobs

The following jobs require manual approval:

- Production deployment (`deploy:production`)
- Staging deployment (`deploy:staging`)

Caching Strategy

To optimize pipeline performance, the following are cached:

- Pip packages (`.cache/pip/`)
- Virtual environment (`.venv/`)

Cache duration: Until next pipeline run

Artifacts

Each stage produces artifacts that are passed to subsequent stages:

Stage	Artifact	Retention
Build	Virtual environment	1 hour
Verify:Security	Security report (JSON)	1 week
Test	Coverage reports (HTML, XML)	1 week
Package	Distribution packages	1 week
Docs	Static website	1 week

Required GitLab CI/CD Variables

Configure these variables in **Settings > CI/CD > Variables**:

Variable	Description	Protected	Masked
`SSH_PRIVATE_KEY`	SSH private key for deployment	Yes	Yes
`DEPLOY_HOST`	Production server IP/hostname	Yes	No

Variable	Description	Protected	Masked
`STAGING_HOST`	Staging server IP/hostname	No	No
`DEPLOY_USER`	SSH username for deployment	No	No

Note: `CI_REGISTRY_USER` and `CI_REGISTRY_PASSWORD` are automatically provided by GitLab.

Local Development

Prerequisites

- Python 3.13+
- pip or uv package manager
- Docker (optional)

Setup

```
# Clone repository
git clone https://gitlab.com/f.saponara/2025_assignment2_python_pipeline_group32.git
cd 2025_assignment2_python_pipeline_group32

# Install dependencies
pip install -r requirements.txt

# Run the application
uvicorn app.main:app --reload

# Run tests
pytest tests/ -v

# Build documentation
mkdocs serve
```

Docker Development

```
# Build image
docker build -t recording-api .

# Run container
docker run -p 8000:80 recording-api

# Or use docker-compose
docker-compose up
```

Testing the Application

Manual Testing with cURL

User Endpoints

```
# Create a user
curl -X POST "http://localhost:8000/api/v1/users" \
    -H "Content-Type: application/json" \
    -d '{"name": "Ada Lovelace"}'

# Get all users
curl -X GET "http://localhost:8000/api/v1/users"

# Update user
curl -X PUT "http://localhost:8000/api/v1/users/1" \
    -H "Content-Type: application/json" \
    -d '{"name": "Grace Hopper"}'

# Delete user
curl -X DELETE "http://localhost:8000/api/v1/users/1"
```

Todo Endpoints

```
# Create a todo
curl -X POST "http://localhost:8000/api/v1/todos" \
    -H "Content-Type: application/json" \
    -d '{"title": "Buy groceries", "completed": false}'

# Get all todos
curl -X GET "http://localhost:8000/api/v1/todos"

# Filter completed todos
curl -X GET "http://localhost:8000/api/v1/todos?completed=true"

# Toggle completion
curl -X PATCH "http://localhost:8000/api/v1/todos/1/toggle"

# Delete todo
curl -X DELETE "http://localhost:8000/api/v1/todos/1"
```

Interactive API Documentation

Once the server is running, visit:

- Swagger UI: http://localhost:8000/docs
- ReDoc: http://localhost:8000/redoc

Project Structure

```
2025_assignment2_python_pipeline_group32/
├── .gitlab-ci.yml          # CI/CD pipeline configuration
├── Dockerfile              # Docker image definition
├── docker-compose.yaml         # Docker Compose configuration
├── pyproject.toml           # Project metadata and dependencies
├── requirements.txt          # Python dependencies
├── mkdocs.yml               # Documentation configuration
├── README.md                  # This file
├── app/                    # Application source code
│   ├── __init__.py
│   ├── main.py               # FastAPI application entry point
│   ├── api/                  # API routes
│   │   └── v1/
│   │       ├── user.py         # User endpoints
│   │       └── todo.py         # Todo endpoints
│   ├── core/                 # Core configuration
│   │   ├── config.py           # Application settings
│   │   └── logging.py          # Logging configuration
│   ├── db/                   # Database layer
│   │   └── schema.py           # SQLAlchemy models
│   ├── models/               # Pydantic models
│   │   ├── user.py             # User data models
│   │   └── todo.py             # Todo data models
│   └── services/             # Business logic
│       ├── user_service.py     # User service layer
│       └── todo_service.py     # Todo service layer
├── tests/                  # Test suite
│   ├── __init__.py
│   ├── test_db.py            # Test database setup
│   └── api/
│       └── v1/
│           ├── test_user.py    # User endpoint tests
│           └── test_todo.py    # Todo endpoint tests
└── docs/                   # Documentation source
    ├── index.md
    ├── getting-started/
    └── api/
```

Pipeline Execution Timeline

Typical execution times for a complete pipeline run:

```
Stage 1: Build          [██████████] ~2-3 min
Stage 2: Verify (||)    [██████████] ~3-4 min (parallel)
Stage 3: Test           [█████]      ~1-2 min
Stage 4: Package          [███]       ~1 min
Stage 5: Release         [████████]  ~3-5 min
Stage 6: Docs           [████]       ~1-2 min

Total Duration: ~13-20 minutes
```

Lessons Learned

Technical Challenges

1. **Parallel Verification Jobs:** Implementing truly parallel static and security analysis required careful job configuration
2. **Docker Image Optimization:** Balancing image size with build speed
3. **Test Database Isolation:** Ensuring tests use in-memory database to avoid conflicts

Best Practices Applied

1. **Comprehensive Comments:** Every stage in `.gitlab-ci.yml` is thoroughly documented
2. **Artifact Management:** Strategic use of artifacts to pass data between stages
3. **Caching:** Implemented caching to reduce pipeline execution time
4. **Security:** Used GitLab CI/CD variables for sensitive information

Future Improvements

- Add performance testing stage
- Implement database migrations
- Add integration with notification services
- Implement blue-green deployment strategy
- Add automated rollback mechanism

Compliance with Assignment Requirements

✅ **6 Pipeline Stages:** All 6 stages implemented (build, verify, test, package, release, docs)

✅ **Parallel Verification:** Static analysis (Prospector) and dynamic analysis (Bandit) run in parallel

✅ **CRUD Operations:** Application implements Create, Read, Update, and Delete operations for **User and Todo entities**

✅ **Comprehensive Comments:** All pipeline stages are thoroughly documented with explanatory comments

✅ **Python-Specific Tools:**

- Build: pip
- Verify: Prospector + Bandit
- Test: pytest
- Package: setuptools + wheel
- Release: Docker + GitLab Container Registry
- Docs: MkDocs

✅ **Documentation:** Complete documentation with MkDocs published to GitLab Pages

References

- FastAPI Documentation
- GitLab CI/CD Documentation
- Docker Documentation
- MkDocs Documentation
- pytest Documentation

Contact

For questions or issues related to this project, please contact:

- **Francesco Saponara 886465 (f.saponara@campus.unimib.it)**
- **Cristina M. Stanculea 945221**
- **Mattia Chittoni 886462**

Course: Software Development Process
Academic Year: 2025-2026
Submission Date: November 2025

This project demonstrates the implementation of a complete CI/CD pipeline for a Python web application, fulfilling all requirements of Assignment 2.