
# TaskTrail Web Application

Supervisor's name: Saif Mujahid

Student's name: Jackson Aulds

Group: RIM-140930

# Contents

# Introduction

## Project Goals

TaskTrail is a web-based project management system designed to help individuals and teams track tasks and deadlines. The goal is to create a responsive and feature-rich platform that allows task tracking, status updates, and deadline management.

## Business Use Case

In small to medium teams, project visibility and accountability tend to be lacking due to disorganized tools or non-collaborative systems. TaskTrail fills this gap by offering a lightweight and user-friendly task manager with user-based access and live status updates, which is a viable alternative to heavier platforms like Jira or Asana.  Especially easier to use for those who are not tech savvy.

## Problem Statement

Existing project management tools are either too complex or too limited for lean teams and individual users. TaskTrail aims to provide the right balance of simplicity, usability, and power to help users stay productive without a steep learning curve.

# System Design

## Functional Architecture

- **Frontend:** Django Templates (HTML/CSS/JS)
- **Backend:** Django (Python)
- **Database:** SQLite (development), PostgreSQL (production)
- **Cache:** Memcached
- **Server/Runtime:** Gunicorn + Nginx (for production)

## Key Use Cases

- User registration and login
- Creating and updating tasks
- Changing task status (Not Started → In Progress → Completed)
- Viewing task dashboards
- Caching frequent views with Django Caching

## Key Features

- Authentication with Django's built-in auth system
- Task creation and update forms
- Task status change buttons
- Django Caching caching for frequently accessed views
- Unit testing using Django’s testing framework

# Implementation

## Technology Choices

- **Backend:** Django (Python)
- **Frontend:** HTML5, Bootstrap, JavaScript
- **Database:** SQLite (for development), PostgreSQL (for production)
- **Caching:** Django Caching via cache_page
- **Deployment Tools:** Gunicorn, Nginx, Docker

## Key Implementation Notes

- The update_task_status view updates task status using a form and the task's primary key.
- Access to task-modifying views is protected using Django's @login_required decorator.
- Django Caching is used to store results of expensive queries such as frequently accessed homepage data.

# Testing & Deployment

## Testing Summary

- test_taskstatus_update: Validates that task status updates correctly using the update view.
- test_update_task_requires_login: Ensures unauthorized users are redirected when accessing protected views.

## Docker Setup

### Dockerfile

```
FROM python:3.13.2

ENV VIRTUAL_ENV=/opt/venv
RUN python -m venv $VIRTUAL_ENV
ENV PATH="/$VIRTUAL_ENV/bin:$PATH"

ENV APP_HOME=/app

# create the directory/folder
RUN mkdir $APP_HOME
# create the folder for static files
RUN mkdir $APP_HOME/static
WORKDIR $APP_HOME

EXPOSE 8000
COPY requirement.txt $APP_HOME/requirement.txt
RUN pip install -r requirement.txt
COPY . $APP_HOME/
```

## docker-compose.yml

```
services:
  db:
    image: postgres:15.2
    restart: always
    env_file:
      - .env
    ports:
      - "127.0.0.1:${DATABASE_PORT}:${DATABASE_PORT}"
    environment:
      - PGPORT=${DATABASE_PORT}
      - POSTGRES_DB=${DATABASE_NAME}
      - POSTGRES_USER=${DATABASE_USER}
      - POSTGRES_PASSWORD=${DATABASE_PASS}
    volumes:
      - postgres_volumn:/var/lib/postgresql/data

  api:
    build: .
    command: >
      sh -c "python manage.py migrate &&
        python manage.py collectstatic --noinput &&
        gunicorn base.wsgi:application --bind 0.0.0.0:8000"
    ports:
      - 8000:8000
    volumes :
      - ./:/app
    depends_on:
      - db

  nginx:
    build: ./nginx
    volumes :
      - ./static:/app/static
      - ./media:/app/media
    ports:
      - 81:80
    depends_on:
      - api

volumes:
  postgres_volumn:
```

# Conclusion & Literature

## Lessons Learned

- Security is a complex issue and there are many ways to break through
- Writing unit tests ensures long-term code reliability and helps security
- Django's admin interface simplifies project and user management

## Future Work

- Add user roles and permissions for multi-tier access
- Expose REST APIs for mobile and third-party integrations
- Add analytics and reporting features for projects

## References


- Django Documentation
- Docker Documentation
- Our Course Website and Projects