Jessa Orobia 5/22/2025

BSIT-3R5

TO-DO LIST APP DOCUMENTATION

Project Title: Todo List App using FastAPI and React

This project is a full-stack To-Do List application built using FastAPI for the backend and React (Vite) for the frontend. It allows users to:

- Add, edit, delete tasks
- Mark tasks as completed
- Filter tasks by status (all, completed, pending)
- Toggle dark mode
- Persist data using a backend API and database

The backend exposes a RESTful API and is connected to a database using SQLAlchemy with SQLite. SQLite was used in this project because the free PostgreSQL instance on Render had already been used for a previous activity, and SQLite provided a simple and lightweight alternative suitable for development and small-scale deployment. The frontend interacts with this API using Axios and is styled for responsiveness and theme toggling.

FastAPI vs Django REST Framework (DRF)

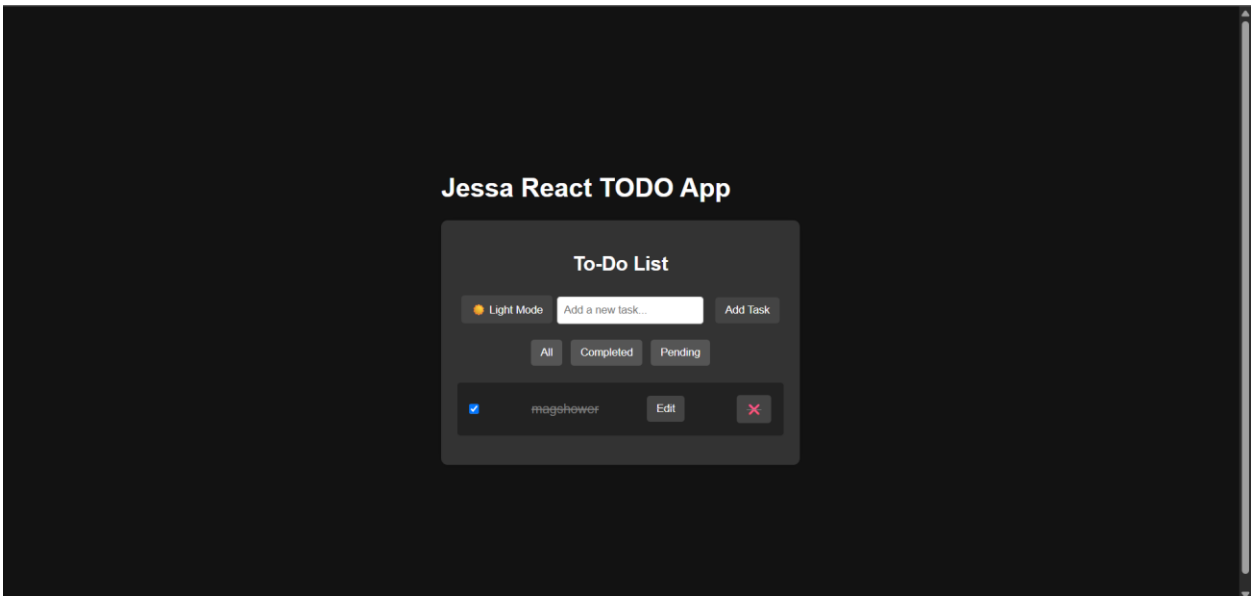
Feature	FastAPI	Django REST Framework (DRF)
Performance	Very fast (async supported)	Slower, sync by default
Simplicity	Lightweight and easy to set up	More boilerplate, heavier
Learning Curve	Requires understanding async concepts	Easier for beginners in APIs, if familiar with Django
Flexibility	Highly flexible, modular	Tied closely to Django's ecosystem
Best For	High-performance APIs, microservices	Full-featured web apps, rapid prototyping

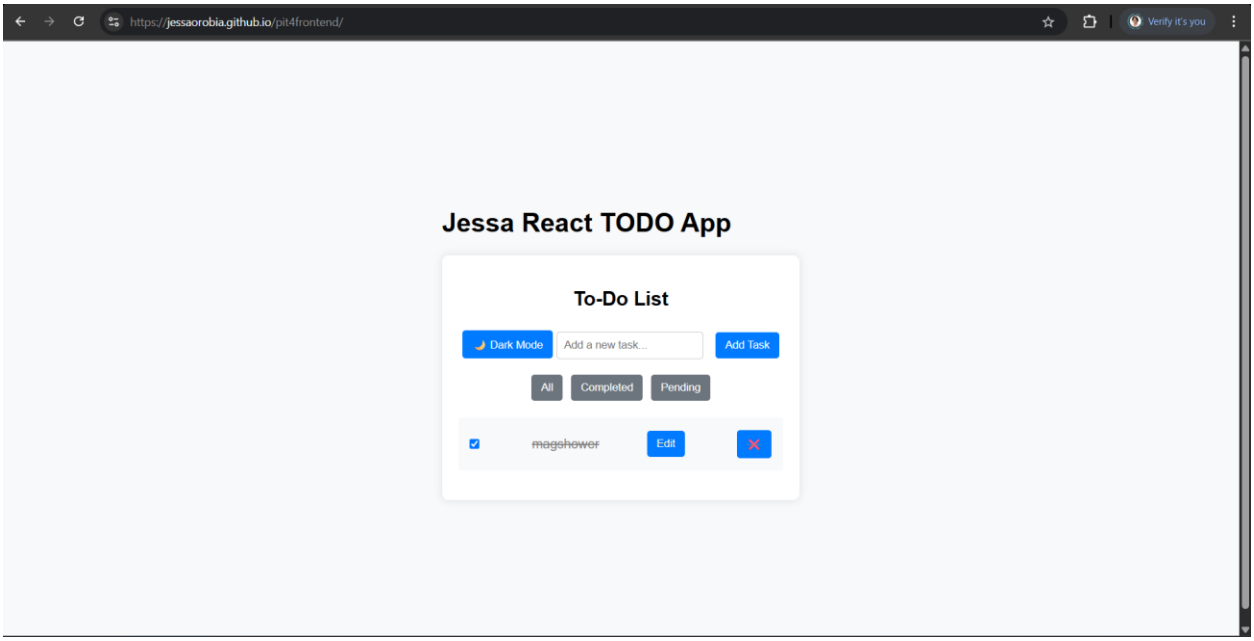
Technologies Used:

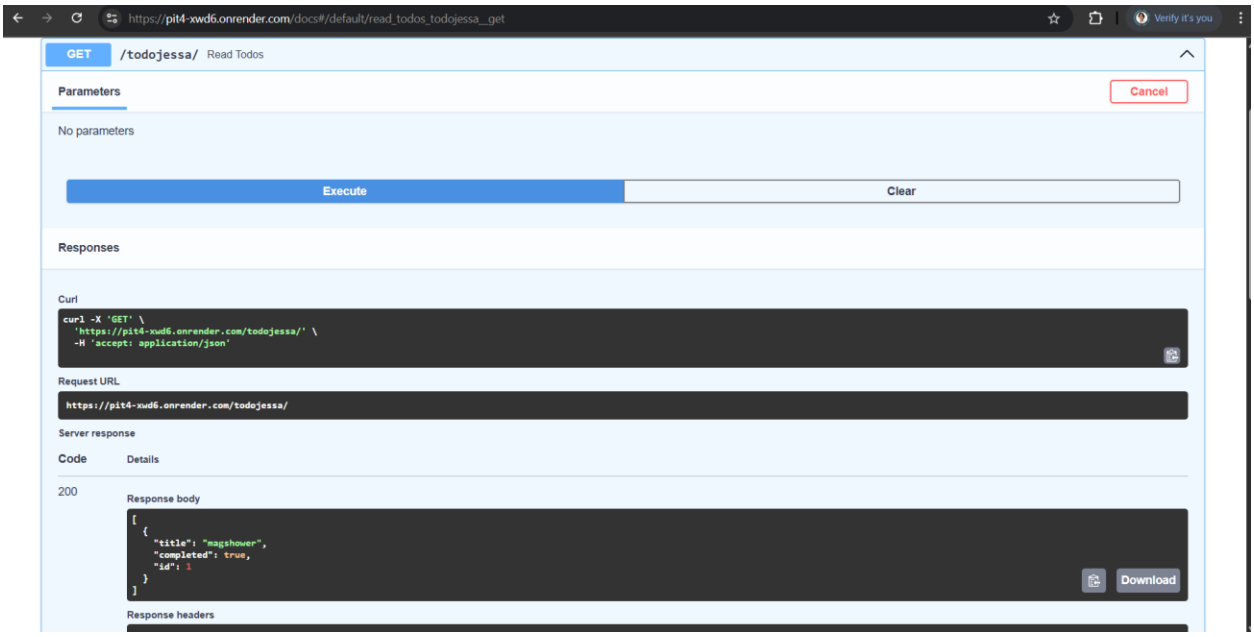
- Frontend: React + Vite
- Backend: FastAPI + SQLAlchemy
- Database: SQLite (Free Postgres instance on render had already been used for a previous activity)

- **Deployment:** Backend on Render, Frontend on GitHub Pages

Screenshots:







Live Links:

Frontend: https://jessaorobia.github.io/pit4frontend/
Backend: https://pit4-xwd6.onrender.com/docs

Challenges Encountered:

- **Django REST Framework (DRF):**
  One of the main challenges was deploying the project to Render. I had trouble pushing the repository to GitHub initially, which slowed down the deployment process. Additionally, configuring environment variables and setting up the database on Render took longer than expected. The tight coupling with Django also meant I had to manage settings across multiple files, which could get confusing. Debugging during deployment was another hurdle, as error messages weren't always clear.

- **FastAPI:**
  Although I had prior experience with DRF, working with FastAPI introduced a few new learning curves. Understanding how to properly structure the project and handle asynchronous code required some extra research. There were minor issues with dependency management and integrating third-party packages. Deploying with Render also had its own small challenges, particularly in setting up the Uvicorn server and ensuring the requirements.txt included all necessary modules. Despite these, the overall setup process was smoother due to FastAPI's simplicity.