SCT211-0024/2023 NEVEAN ADHIAMBO OMONDI

**Project Documentation: Django-React Task Manager Application**

This document provides comprehensive information about the Django-React Task Manager project. It includes a description of the project's distinctiveness and complexity, design approach, file structure, setup instructions, and additional details.

**Distinctiveness and Complexity**

The Task Manager application is a unique combination of Django's robust backend framework and React's dynamic front-end capabilities. The following features highlight its distinctiveness and complexity:

1. **Integration of Django and React**:
   * The project integrates Django REST Framework (DRF) for creating APIs and React for a modern, responsive user interface.
   * Achieving smooth communication between the backend and frontend required careful API design and implementation.
2. **Full CRUD Functionality**:
   * Users can create, read, update, and delete tasks.
3. **JWT Authentication**:
   * The app uses JSON Web Tokens (JWT) for secure user authentication.
   * Only authenticated users can access or manipulate their tasks.
4. **Frontend Interactivity**:
   * React's reusable components provide a dynamic and smooth user experience.
   * Features like real-time task updates and error feedback add complexity.
5. **Backend Logic**:
   * Django handles database management, task prioritization, and API endpoint security.
   * The backend was optimized for performance and scalability.

This combination of technologies and features demonstrates a unique and challenging implementation.

**Design Approach**

**1. Frontend Design**

* The React frontend was built with reusable and modular components for easy maintenance and scalability.
* State management is handled using React's useState and useEffect hooks, ensuring real-time updates when interacting with the backend.
* CSS was used for styling, with a focus on creating a clean, minimal, and responsive UI.

**2. Backend Design**

* Django REST Framework (DRF) was chosen for its simplicity in creating RESTful APIs.
* Models were designed to include fields for task title, description, priority, due date, and user association.
* API endpoints were built to support CRUD operations with proper permissions and validation.

**3. Authentication**

* Django’s built-in authentication system was extended with JWT for secure user authentication and token management.
* A middleware was implemented to ensure only authorized users can perform specific actions.

**File Structure and Descriptions**

Here’s an overview of the key files and their roles in the project:

**Backend (Django)**

1. manage.py: Entry point for running the Django application.
2. settings.py: Contains project settings, including database configuration and REST framework settings.
3. urls.py: Defines the URL patterns for API endpoints.
4. models.py:
   * Defines the Task model with fields like title, description, priority, due date, and user association.
5. serializers.py:
   * Transforms model data into JSON format and validates incoming data.
6. views.py:
   * Contains logic for handling API requests (GET, POST, PUT, DELETE).
7. permissions.py:
   * Implements custom permissions to ensure users can only access their own tasks.

**Frontend (React)**

1. index.js: Entry point for the React application.
2. App.js:
   * Main application component managing routing and global state.
3. components/:
   * TaskList.js: Displays a list of tasks fetched from the backend.
   * TaskForm.js: Handles creating and updating tasks.
   * Login.js and Register.js: Handles user authentication.
4. services/api.js:
   * Contains helper functions for making API calls to the backend.
5. styles/: CSS files for styling components.

**How to Run the Application**

**Prerequisites**

* Python 3.8+
* Node.js and npm
* Django and React dependencies

**Setup Instructions**

1. **Clone the Repository**:

bash

CopyEdit

git clone <repository\_url>

cd project\_directory

1. **Backend Setup**:
   * Navigate to the backend folder:

bash

CopyEdit

cd backend

* + Install Python dependencies:

bash

CopyEdit

pip install -r requirements.txt

* + Run database migrations:

bash

CopyEdit

python manage.py migrate

* + Start the Django development server:

bash

CopyEdit

python manage.py runserver

1. **Frontend Setup**:
   * Navigate to the frontend folder:

bash

CopyEdit

cd frontend

* + Install Node.js dependencies:

bash

CopyEdit

npm install

* + Start the React development server:

bash

CopyEdit

npm start

1. **Access the Application**:  
   Open your browser and navigate to http://localhost:3000 for the frontend or http://localhost:8000 for the backend API.

**Additional Information**

* **Database**: SQLite was used for development; it can be replaced with PostgreSQL or another production-ready database.
* **Error Handling**: The application includes both backend and frontend validation for a seamless user experience.
* **Deployment**: The app can be deployed using tools like Docker, Heroku, or AWS.
* **Scalability**: The architecture supports easy extension, such as adding new features like task reminders or notifications.