Task Manager App

Overview

This is a full-stack task manager application built using **Flask** for the backend, **PostgreSQL** for the database, and **React** for the frontend. The app allows users to create, update, delete, and mark tasks as completed or incomplete. It also includes user registration, authentication using JWT, and protected routes for task management.

Features

- User registration and authentication using JWT.
- Task management with the ability to add, edit, delete, and mark tasks as complete.
- Secure password handling using bcrypt.
- Protected routes for authenticated users.
- API endpoints for task management.
- Admin panel for managing users and tasks using Flask Admin.

Tech Stack

- **Backend**: Flask, SQLAlchemy, Flask-JWT-Extended, PostgreSQL
- **Frontend**: React, Axios, Tailwind CSS
- **Database**: PostgreSQL

Setup Instructions

Prerequisites

- **Node.js** (for the frontend)
- **Python 3.8+** (for the backend)
- **PostgreSQL** (for the database)

Backend Setup

1. Clone the repository.
2. Navigate to the backend folder:

CMD/VSCode Terminal:

```
cd backend
```

3. Create a virtual environment and activate it:

   CMD/VSCode Terminal:

   ```
   python -m venv venv
   source venv\Scripts\activate
   ```

4. Install the dependencies:

   CMD/VSCode Terminal:

   ```
   pip install -r requirements.txt
   ```

5. Set up your environment variables:

   CMD/VSCode Terminal:

   ```
   export FLASK_APP=app.py

   export DATABASE_URL=your_postgres_database_url

   export FLASK_ENV=development
   ```

6. Initialize the database:

   CMD/VSCode Terminal:

   ```
   flask db init

   flask db migrate

   flask db upgrade
   ```

7. Run the Flask server:

   CMD/VSCode Terminal:

   ```
   flask run
   ```

Frontend Setup

1. Navigate to the frontend folder:

CMD/VSCode Terminal:

```
cd frontend
```

2. Install the dependencies:

CMD/VSCode Terminal:

```
npm install
```

3. Start the React development server:

CMD/VSCode Terminal:

```
npm start
```

Note: The frontend will be available at `http://localhost:3000`, and the backend will be available at

`http://localhost:5000`.

API Endpoints

Authentication & User Management

Endpoint	Method	Description	Authentication Required
/register	POST	Register a new user	No
/login	POST	Log in and receive a JWT token	No
/users	GET	Get all registered users	Yes

Task Management

Endpoint	Method	Description	Authentication Required
/tasks	GET	Get a list of all tasks	Yes
/tasks	POST	Create a new task	Yes

/tasks/:id	GET	Get a specific task by ID	Yes
/tasks/:id	PUT	Update a task	Yes
/tasks/:id	DELETE	Delete a task	Yes

Request & Response Examples

1. Register a User

- Endpoint: `/register`
- Method: POST
- Request Body:
  - JSON:

```
{
        "username": "testuser",
        "password": "password123"
}
```

- Response:
  - JSON:

```
{
  "message": "User created successfully"
}
```

2. Login a User

- Endpoint: `/login`
- Method: POST
- Request Body:
  - JSON:

```
{
  "username": "testuser",
  "password": "password123"
}
```

- Response:
    - JSON:

```
{
  "token": "jwt_token_here"
}
```

3. Get All Tasks

- Endpoint: `/tasks`
- Method: GET
- Response:
    - JSON:

```
[
  {
    "id": 1,
    "title": "Complete Project",
    "description": "Finish the task manager project",
    "completed": false
  },
  {
    "id": 2,
    "title": "Prepare Documentation",

    "description": "Write the README.md file",

    "completed": true

  }

]
```

4. Create a New Task

- Endpoint: `/tasks`
- Method: POST
- Request Body:
  - JSON:

```
{

  "title": "New Task",

  "description": "Description of the new task"

}
```

- Response:
  - JSON:

```
{

  "message": "Task created.",

  "task": {

    "id": 3,

    "title": "New Task",

    "description": "Description of the new task",

    "completed": false
```

```
 }
}
```

5. Update a Task

- Endpoint: `/tasks/:id`
- Method: PUT
- Request Body:
    - JSON:

```
{
  "title": "Updated Task",
  "description": "Updated task description",
  "completed": true
}
```

- Response:
    - JSON:

```
{
  "message": "Task updated!"
}
```

6. Delete a Task

- Endpoint: `/tasks/:id`
- Method: DELETE
- Response:
    - JSON:

```
{
  "message": "Task deleted."
}
```

Deployment Instructions

The app is deployed on Render. The deployment steps are:

1. **Backend Deployment**:
   - Set up the backend by creating a **Render Web Service** for the Flask app.
   - Add environment variables (e.g., `DATABASE_URL`, `JWT_SECRET_KEY`).
   - Push your code to the **Render Git** repository.
2. **Frontend Deployment**:
   - Create a **Render Static Site** for the React app.
   - Set the build and start commands:
     - **Build command**: `npm install && npm run build`
     - **Start command**: `serve -s build`
   - Push your code to the **Render Git** repository.
3. **Database Deployment:**
   - Create a **Render PostgreSQL** DB.
   - Set name of database and get **Hostname, Username, Password and URL** from Render.(Use an internal url for connection with render.)
   - Download **PGadmin** and connect the server using **external url** from render.