Final Project: MERN Stack "Mini App"

Project Goal

Build a small but complete **full stack MERN application** that:

- Uses a **MongoDB database** with at least one **complex data model** (relations, embedded docs, etc.).
- Implements **user authentication** (register/login/logout).
- Provides a **React front-end** with **Redux state management**.
- Calls your **own API endpoints** from the frontend.
- Handles **errors gracefully** and includes at least one **test case**.
- Is **deployed to the cloud**.

You should upload the project to **GitHub** and send you the repository link with a **README file** that explains:

- Setup instructions
- Cloud deployment link

Suggested App Ideas (Pick One)

Choose whichever feels easiest to complete:

1. **Task Manager** – Users can sign up, log in, and manage a list of tasks (CRUD operations). Tasks can have categories, priorities, or due dates (complex data model). You already know about this project – you can reuse the code from earlier versions of this application.
2. **Simple Blog** – Users can register and post blog entries. Posts can have comments and tags.
3. **Book Tracker** – Users can add books they've read, with fields like author, genre, rating, and notes.

Requirements Breakdown

1. Database & Models

- Create at least **two Mongoose models** with a relation (e.g., User ↔ Task, User ↔ Post).
- Example:
- const UserSchema = new mongoose.Schema({
- username: String,
- email: String,
- password: String,
- });
-
- const TaskSchema = new mongoose.Schema({
- title: String,
- completed: Boolean,
- user: { type: mongoose.Schema.Types.ObjectId, ref: "User" },
- });

2. API & Redux

- Backend: Build at least **3 CRUD routes** (GET, POST, PUT/PATCH, DELETE).
- Frontend: Use **Redux Toolkit** to store the user's login status and at least one piece of app data.
- Example: Tasks fetched from the backend stored in Redux.

3. User Authentication

- Implement **JWT-based authentication** (jsonwebtoken + bcrypt).
- Allow login, register, logout.
- Protect at least one route so only logged-in users can access it.

4. Error Handling & Testing

- Handle errors in backend routes with try/catch.

- Send meaningful error responses (e.g., 400, 401, 500).
- Add at least **1 test file** (Jest or Mocha/Chai). Example: test that API returns 200 when fetching tasks.

5. Deployment

Deploy to any **free tier cloud service (Just suggestions)**:

- **Frontend**: Vercel or Netlify
- **Backend**: Render, Railway, or Heroku
- **Database**: MongoDB Atlas

Provide the deployment link in the README.

Tips & Shortcuts

- Don't overcomplicate – keep UI minimal (a form and a list is fine).
- Reuse code from earlier class projects.
- For auth, use a starter template (many examples exist).
- For Redux, only manage **one piece of state** beyond auth.
- Testing: One simple API test is enough.

Deliverables

1. GitHub repo with:
   - /client → React + Redux frontend
   - /server → Express backend
   - README.md with instructions + cloud link
2. Working deployed version online

You can use other tools, cloud services, authentication methods, etc. You are not restricted to what I suggested here. Just create a good but simple full-stack application that helps you with your goals...