Project Goal: Build a simplified Task Management application.

Technologies:

- **Backend:** .NET 9 Web API
- **Frontend:** Your choice of React (with TypeScript) or Angular (latest stable).
- **Database:** Your choice (below are some examples)
  - In-memory database (e.g., SQLite in-memory, or a simple collection in .NET) or SQL Server LocalDB or a Dockerised SQL Server/PostgreSQL instance.

General Requirements & Best Practices

1. **Code Quality:** Your code should be clean, readable, well-organised, and adhere to standard coding conventions.
2. **Project Structure:** Maintain a logical and scalable project structure
3. **Error Handling:** Implement user-friendly error messages for API failures.
4. **Version Control:** Submit your solution as a GitHub repository
5. **`README.md`:**
   - Clear instructions on how to set up and run both the backend and frontend components.
   - A brief explanation of your architectural decisions, assumptions, or trade-offs made.

Core Requirements:

Your solution should implement the following functionalities:

Backend (.NET 9 Web API)

1. **Data Model:** Create a `Task` entity with at least the following properties:
   - `Id` (int, primary key)
   - `Title` (string, required, max length 100)
   - `Description` (string, optional)
   - `IsCompleted` (bool, default `false`)
   - `DueDate` (DateTime, optional)
2. **RESTful API Endpoints:** Implement standard CRUD operations for tasks
3. **Data Persistence:** Use Entity Framework Core with your chosen database.
4. **Business Logic & Architecture:**
   - **Service Layer:** Clearly separate business logic from controllers
   - **Dependency Injection:** use of .NET's built-in DI.
5. **Error Handling:** Implement robust server-side error handling
6. **Validation:** Implement server-side validation for the `Task` model

7. **Project Structure:** Organise your backend code logically
8. **Unit Tests:** Write one or two meaningful unit tests for your backend business logic or service layer using xUnit/NUnit and a mocking framework (e.g., Moq, nUnit, xUnit).

Frontend (Styling Not Assessed)

Core Task: Create a Single-Page Application (SPA) that consumes your .NET Web API.

1. **User Interface:**
    - **Task List View:** Display all tasks, showing title, completion status, and due date.
    - **Task Detail/Edit View:** A dedicated view/form to see details of a task and allow editing.
    - **Add New Task:** A form to create a new task.
2. **Functionality:**
    - Display, add, edit, and delete tasks by interacting with your backend API.
    - Provide a way to toggle a task's `IsCompleted` status directly from the task list.
3. **Form Handling & Validation:**
    - Implement client-side form validation (e.g., `Title` is required). Display user-friendly error messages.
4. **API Integration:**
    - Use `HttpClient` (Angular) or `fetch`/`axios` (React) to communicate with the backend.
5. **Routing:**
    - Implement basic client-side routing to navigate between different views (e.g., task list, add/edit task).
6. **State Management:**
    - Demonstrate a clear understanding of state management within your chosen framework.
    - (Optional) Implement a robust state management solution like Redux Toolkit/Ngrx, or a well-structured Context API/service-based approach.
7. **Client-Side Filtering, Sorting, Pagination:**
    - Implement UI controls for the enhanced task list features that interact with the backend API's filtering, sorting, and pagination parameters.
8. **Frontend Tests:** Write a few unit tests for key components or services.
    - Please include one or two basic **end-to-end (E2E) tests** using Playwright to demonstrate a full user flow.

Bonus Section (Optional): Advanced Considerations

If you're eager to showcase a deeper level of expertise and have additional time beyond the core requirements, consider addressing some of the following points.

These are not mandatory for a successful submission but will provide a stronger indicator of senior-level design and thought processes.

1. **Data Model:** Add `CreatedDate`, `LastModifiedDate`, `CreatedBy` (User ID), `AssignedTo` (User ID).
2. **`README.md`:**
    - (Optional) Also include a brief discussion on how you might approach scalability, security considerations, and potential deployment strategies for this application in a production environment.)
3. **Backend**
    - **Filtering, Sorting, Pagination:**
        i. Enhance the `GET /api/tasks` endpoint to support querying tasks by completion status, due date range, sorting options (e.g., by title, due date), and basic pagination.
4. **Enhanced Code Quality & Maintainability:**
    - Demonstrate exceptional code clarity, consistency, and adherence to advanced style guides.
    - Show sensible use of design patterns (e.g., Mediator, Strategy, Decorator where applicable) to enhance modularity and maintainability.
    - Focus on clean architecture principles, emphasizing loose coupling and high cohesion throughout the application.
5. **Advanced Documentation:**
    - Beyond basic setup, discuss your architectural decisions in more detail within your `README.md` or a separate `Decisions.md` file.
    - Integrate basic API Documentation (e.g., Swagger/OpenAPI) for the backend.
    - Ensure effective code commenting where necessary to explain complex logic, while always prioritizing self-documenting code.
6. **Deeper Version Control Insights:**
    - Showcase a well-structured Git history with atomic, clear, and concise commit messages that tell a story of your development process.
7. **Scalability & Performance Discussion:**
    - In your README.md or a separate Decisions.md file, discuss potential performance bottlenecks you identified (or anticipate) and how you addressed them within your solution or would address them in a production environment.
8. **Security Discussion:**
    - Elaborate on any security measures implemented (e.g., detailed JWT handling, HTTPS considerations, robust input validation, CORS policies).
    - Mention other critical security considerations for a real-world application, such as secure password hashing, SQL injection prevention, XSS prevention, rate limiting, etc.