Full-Stack Assignment – Extensible Task-Management Platform

1. Objective

Build the initial version of a **Task-Management Platform** that cleanly separates:

1. **General workflow rules** (apply to **every** task, present and future).
2. **Task-specific rules** (apply only to a given task type).

We define here Procurement and Development tasks with their custom fields and lifecycle validation logic, but they are merely **the first two examples**. Your architecture **must** be extensible and ready to host many additional task types, each with its own fields and validation logic, without structural rewrites.

---

2. Core Workflow Rules (General Logic)

#	Rule
1	A task is assigned to **exactly one user** at any moment.
2	A task is either **Open** or **Closed**. Closed tasks are immutable.
3	Each task's progress is tracked by an **ascending numeric status**: 1, 2, 3 …
4	**Forward moves must be sequential** – jumping from status 1 directly to 3 (or any other skip) is forbidden.
5	**Backward moves** are always allowed (e.g., to fix mistakes).
6	A task may be **closed only** when it reaches its **final status**.
7	Every status change must: a. satisfy type-specific data requirements (see 3) b. record the **next assigned user**.

Enforcement of these rules is mandatory for every current and future task type.

---

3. Current Defined Task Types

3.1 Procurement Task

Status	Meaning	Data required before entering
1	Created	—
2	Supplier offers received	2 price-quote strings
3	Purchase completed	Receipt string
Closed	—	May close **only from 3**

Validation flow: 1 → 2 needs two offers; 2 → 3 needs receipt; close only from 3.

3.2 Development Task

Status	Meaning	Data required before entering
1	Created	—
2	Specification completed	Specification text
3	Development completed	Branch name
4	Distribution completed	Version number
Closed	—	May close **only from 4**

Validation flow: 1 → 2 needs specification; 2 → 3 needs branch; 3 → 4 needs version; close only from 4.

---

4. Required Operations (Your actual work):

Operation	Purpose
Create New Task	Accept task type (Procurement or Development) + initial user. The user should come from Users table. No need to manage users in this assignment, just add few users to the table.
Change Stage	Support both task types. Move forward/backward; enforce all general & type-specific rules; assign next user. Use simple strings for things like price quotes, receipts, specifications, etc., and save these custom fields in the DB.

Operation	Purpose
Close Task	Allowed only from final status after validations pass.
Get User Tasks	The user can see the status of all the tasks that are currently assigned to him

---

5. Server-Side Implementation Requirements

- Use **Entity Framework Core** against **SQL Server**, migrations or manual scripts are acceptable. Think how to organize the tables in extensible way, so the system remains as generic as possible.
- **REST API controllers** for the React client (follow REST guidelines) – see client-side requirements
- **MVC controllers + Razor views** for server-rendered scenarios – see client-side requirements.
- Minimize code duplication as much as possible by leveraging the tools available in the .NET Core environment.
- Organize the project into appropriate layers based on standard best practices in .NET Core development.

---

6. Client-Side Requirements

- Provide two implementations of a minimal, reasonable user interface through which the implemented server-side actions can be performed – in React and in MVC. **No need to invest in styling or design:**
    - Create a new task
    - Manage a procurement task lifecycle (advance, reverse, close)
    - Mange development task lifecycle (advance, reverse, close)
    - Show tasks assigned to the current user (hard-coded ID acceptable)
    - Minimize code duplication

---

7. Deliverables

1. **Source code** (solution and React project).
2. **README** with setup steps
3. SQL scripts or migrations, including inserting of few demo users.

Good Luck!