# Project Name: Product Catalog Viewer

---

## Objective

Build a simple full-stack application that allows users to **view and add products** to a catalog. Use **React** for the frontend, **Spring Boot (Java)** for the backend, and **either PostgreSQL or SQL Server** for storage. Containerize the solution using **Docker**. The app should demonstrate clean architecture, RESTful API design, and attention to code quality and detail.

A **sample product JSON** will be provided as a guide to help with understanding the expected structure and fields.

💡 **Bonus:** If you're experienced with Ext JS, feel free to implement the frontend using **Ext JS instead of React** — this is considered a **plus**, but not required.

---

## Functional Requirements

**Frontend (React/ExtJS)**

- Build a UI that allows users to:
  - View a list of products with the following fields:
    - *product_name*
    - *brand*
    - *price*
    - *model*
  - View full product details on click (showing *product_description*, etc.)
  - Add a new product using a form with basic validation

**Backend (Spring Boot + Hibernate + PostgreSQL/SQL Server)**

- Create a *Product* entity with at least the following fields:
  - *productKey* (Long, unique)
  - *retailer* (String)
  - *brand* (String)
  - *model* (String)
  - *productName* (String)
  - *productDescription* (Text)
  - *price* (Decimal)
- Implement the following API endpoints:
  - *GET /products* — List all products
  - *POST /products* — Add a new product
  - *GET /products/{productKey}* — Get full product details

---

**SQL/Query Requirement**

Add one custom query-based endpoint:

- *GET /products/brand-summary*
  Returns a summary count of products grouped by brand, implemented using **JPQL or native SQL**.

**Example response:**

```
[
  { "brand": "Stupell Industries", "count": 5 },
  { "brand": "Fortress Building Products", "count": 2 }
]
```

---

**Testing Expectations**

- Backend: At least 1 unit test (e.g., for service or repository layer)
- Frontend: At least 1 unit test (e.g., for rendering or data fetch logic)

---

**Containerization**

- Provide:
  - Dockerfile for backend
  - (Optional) Dockerfile for frontend
  - *docker-compose.yml* to run:
    - Backend
    - PostgreSQL/SQL Server
    - (Optional) Frontend

Command:

```
docker-compose up --build
```

---

**Resources Provided**

- A sample product file, **products.json**, is included in the assignment package under the **/data** folder.
- Use this file as a **reference** for:
  - Structuring your *Product* model
  - Populating sample data into your database (manually or at startup)
- You are not required to handle JSON parsing unless you choose to.

## Deliverables

- Code (GitHub repo or zip file)
- README including:
  - Setup instructions
  - API overview
  - Sample request/response if helpful
  - Any assumptions or notes
  - What you'd improve with more time

## Documentation & Walkthrough (Required)

- **Record a short (< 5 min) screenshare video**:
  - Walk through your solution
  - Show how to run it and explain key parts

  - Include the video file or a link in the root directory

In addition to the video, feel free to document your implementation in whatever format works best for you (e.g., markdown files, comments, diagrams).

## Scope Guidance

This assignment is designed to take **no more than 2–3 hours**.

Focus on:

- Clear structure and working functionality
- Thoughtful code and design choices

You can:

- Use libraries and scaffolding tools
- Skip full styling and deep test coverage
- Leave TODOs or notes for improvements