Task: Permit Request Service Portal

1. Overview and Goal

The goal of this task is to build a simplified Permit Request Service Portal that allows citizens to submit and view permit applications.

2. Required Technology Stack

You must use the latest stable versions available:

Component	**Technology**	**Version**	**Role**
Frontend/Server	Nuxt	4.x (Latest)	Full-stack framework, UI, and API Gateway
Backend/CMS	Choose ONE: NestJS OR Strapi	11.x (Latest) OR 5.x (Latest)	Business logic and data management
Database	PostgreSQL	Latest	Persistent data storage
Design System	Saudi National Design System	Platforms Code	UI/UX implementation

3. Project Scenario: Permit Application Service

A. Data Model (Required for both NestJS and Strapi)

You must implement a data model for a "Permit Application" with the following fields:

Field Name	Type	Constraints	Default
applicant_name	Text	Required	-
applicant_email	Email	Required	-
permit_type	Text	Required	-
application_status	Enumeration	"Pending", "Approved", "Rejected"	"Pending"
submitted_at	Datetime	Auto-generated	Current timestamp

B. Functional Requirements (Nuxt Frontend)

1. Home Page (/): Display a list of all submitted permit applications.
2. Application Form Page (/apply): A form to submit a new permit application.
3. Status Visualization: The application status ("Pending", "Approved", "Rejected") must be clearly and visually distinguished on the Home Page.

C. Architectural Requirements (Nuxt Server Routes)

CRITICAL: All API calls from the Nuxt frontend must be proxied through Nuxt Server Routes (e.g., ~/server/api/permits.ts).

- The Nuxt Server Routes will act as the API Gateway, handling the communication with your chosen backend (NestJS or Strapi).
- This ensures that sensitive logic and direct backend communication are contained on the server side.

4. Technology-Specific Requirements

Design System

- The entire user interface must be built using components and guidelines from the Saudi National Design System (Platforms Code).
- You must demonstrate Right-to-Left (RTL) support and responsiveness.
- Reference the official Figma community profile for components.

Nuxt 4 (Frontend/Server)

- Use the new app/ directory structure.
- Use useAsyncData or useFetch for all data fetching on the client side, calling your Nuxt Server Routes.
- Implement the Nuxt Server Routes to handle the business logic and communication with your chosen backend.

Backend (NestJS or Strapi)

- PostgreSQL: Must be used as the database for your chosen backend.
- If NestJS is chosen: Implement a full REST API with controllers, services, and DTOs to manage the Permit Application data model.
- If Strapi is chosen: Configure the CMS, create the "Permit Application" collection type, and manage the application status through the Strapi Admin Panel.

5. Deliverables

1. A link to a single Git repository containing the source code for all components (Nuxt, Backend/CMS, and any configuration files).
2. A README.md file in the root of the repository with clear, step-by-step instructions on how to set up, configure, and run the application stack.
3. Be prepared to present your work in a 5–10-minute presentation during the interview, explaining your architectural decisions and code.

Resources

- Saudi National Design System: https://design.dga.gov.sa/
- DGA Figma Profile: https://www.figma.com/@sdga
- Nuxt 4 Documentation: https://nuxt.com/docs/4.x/getting-started/introduction
- NestJS Documentation: https://docs.nestjs.com/
- Strapi 5 Documentation: https://docs.strapi.io/