Final Project Requirements Document

Overview

For your final project, you will build a fully functional, server-side rendered web application with strong emphasis on backend development. The application must follow industry best practices, be modular, and use ESM (ECMAScript Modules). You will be graded on functionality, code quality, database design, and adherence to best practices.

While you may choose the theme of your website, it must have key structural components outlined in this document. You are expected to **complete at least 70% of the requirements**.

Technology Stack

Your project must use the following technologies:

- **Node.js** with **Express.js** as the backend framework
- **EJS** (Embedded JavaScript) for server-side rendering
- **ESM** (ECMAScript Modules) - No CommonJS (require is not allowed)
- **PostgreSQL** for the database
- **Deployed on Render** with a connected PostgreSQL database

---

Core Features

Your website must include the following features:

1. User Accounts & Authentication

- Users can create an account, log in, and log out.
- Users should have different roles:
    - **Admin** (full access to manage core site content and users)

    - **Other roles** depending on your website's purpose (e.g., employees, customers, moderators, vendors, etc.)
    - **Standard User** (basic account functions)
- Session-based authentication using **Express-session** (No JWT).

2. Database Design & Structure

- Your database must be properly normalized and use **appropriate data types**.
- It must include **multiple related tables** to structure your data efficiently.
- Relationships between tables must be correctly implemented using **foreign keys**.
- You must design your tables thoughtfully instead of dumping all data into one.

3. Dynamic Content Management

- Admins (or equivalent role) must have control over **key site content** that other users do not.
- Users should be able to view and interact with dynamically generated content based on their roles.
- Your database should support structured data that allows users to interact with the website.

4. Page Views and Server-Side Rendering

- All pages must be server-side rendered using **EJS**.
- Implement dynamic routing for displaying specific content (e.g., /product/:id, /game/:id, /listing/:id).
- Use layouts and **partials** to keep code **DRY** (Don't Repeat Yourself):
    - DRY means breaking down repeated sections (headers, footers, sidebars, forms) into reusable components.

    - Avoid redundancy by making functions, database queries, and templates modular.

5. User Interactions & Engagement

- Implement an **interaction system** relevant to your site (e.g., reviews, ratings, comments, upvotes, etc.).
- User-generated content should be stored in the database and linked to users.
- Implement validation to prevent inappropriate or empty submissions.

6. Contact Form (Saved to Database)

- The site must have a **contact form** that allows users to submit messages.
- Messages must be saved to the database (NOT emailed).
- A designated role (e.g., admin, moderator, employee) must be able to **view and manage** these messages.

7. Internal Task Management or Workflow System

- Your site must include some form of **task tracking or workflow management** that aligns with its purpose.
- Examples include:
    - A **support ticket system** (users submit tickets, staff handle them).
    - A **content approval system** (admins review user-submitted content before publishing).
    - A **request system** (users request something, and staff respond or approve them).
- Users should be able to see the history/status of their interactions.

8. Admin/Management Dashboard

- Your application must have **administrative controls** for managing key site functions.
- The dashboard should allow management of some or all of the following features:
    - **Users & Roles** (assign permissions, manage accounts)
    - **Site Content** (adding, editing, deleting core data related to your website)
    - **Moderation Tools** (approving, flagging, or handling user-generated content)
    - **Operational Data** (viewing logs, tracking system activity)
- The level of control should be appropriate to your website's needs.

9. Code Quality & Best Practices

- Code should be **modular** and follow **industry standards**.
- Use **ESM syntax** (no require() statements).
- Implement **proper error handling** (try/catch, middleware for error handling).
- Maintain **clean folder structure**:
    - **MVC Pattern**: Separate concerns into **Models, Views, and Controllers**.
    - Go beyond classroom examples by **organizing files efficiently**, using **modular imports/exports**, and **structuring your app for scalability**.

10. Deployment & Final Review

- The project must be **deployed on Render**.
- The PostgreSQL database must be properly connected in production.

Grading Breakdown

Category	Points
Functionality (Meets Core Requirements)	30
Quality & Industry Standards	20
Proper Database Design & Relationships	20
Session-Based Authentication & Security	10
UI & UX (User-Friendly, Responsive)	10
Total	**100**

Minimum to Pass: 70 Points

- Completing **at least 70% of the requirements** is the absolute minimum to pass. You should aim to implement as much as possible to demonstrate your backend development skills.
- Completing **90% or more** will be considered an **excellent** submission.

Submission Requirements

- Your project's source code must be hosted on GitHub with a well-structured repository.
- You must submit a working live deployment on Render with a properly connected PostgreSQL database.
- Your database design and structure will be reviewed for normalization, relationships, and appropriate data types.