CMS Product CRUD - Next.js, Express.js, MySQL (Internshala Assignment)

1. Cover Page

This document outlines the architecture, features, and implementation details of a full-stack Content Management System (CMS) style product module. The project demonstrates core CRUD (Create, Read, Update, Delete) functionalities, enhanced with publish/draft states and a soft delete mechanism, providing a robust solution for product management.

The application is designed to be responsive and user-friendly, leveraging modern web technologies to deliver a seamless experience. It serves as a practical demonstration of full-stack development capabilities, focusing on clean code, efficient data handling, and a well-structured architecture.

Name: Rhythm Das

Role: Web Developer (Internshala Applicant)

Email: rhythmdas@example.com

Phone: +91-9876543210

2. Project Overview

This project implements a CMS-style product module, enabling comprehensive management of product listings. Key functionalities include Create, Read, Update, and Delete operations, alongside a crucial publish/draft toggle for content visibility and a soft delete feature (setting `is_deleted=1` instead of permanent removal) for data integrity. The application is built with a modern technology stack to ensure performance and scalability.

The frontend is developed using Next.js, styled with Tailwind CSS for a responsive design, and enhanced with Framer Motion for smooth micro-animations. The backend is powered by Express.js, following an MVC (Model-View-Controller) architectural pattern for clear separation of concerns. Data persistence is handled by a MySQL database, hosted on Railway, while both the frontend and backend applications are deployed on Render for reliable cloud hosting.

3. Live Links & QR Codes

Explore the live deployment of the application and access the source code through the links provided below. Each link is accompanied by a conceptual QR code for convenient mobile access, allowing evaluators to quickly navigate to the deployed instances and the GitHub repository.

Live Frontend: https://cms-products-frontend.render.com (QR Code would be here)

Live API/Backend: https://cms-products-backend.render.com (QR Code would be here)

GitHub Repository: https://github.com/Rhythm82/cms-products (QR Code would be here)

4. Features Implemented

? **Add New Product**: Users can easily add new products by providing a name, detailed description, and initial status (Draft or Published). Robust form validation ensures data integrity and a smooth user input experience.

? **View All Products (Admin)**: An administrative interface allows for viewing all products, including both Draft and Published items. This view includes filtering options to quickly sort products by their current status, enhancing administrative control.

? **Edit Product Details**: Existing product information, such as name, description, and status, can be updated through a dedicated editing interface. This ensures that product details remain current and accurate.

? **Publish/Draft Toggle**: Products can be toggled between 'Published' and 'Draft' states. Published items are visible on the public-facing home page, while Drafts are exclusively accessible within the admin panel, facilitating content staging.

? **Soft Delete**: Instead of permanent deletion, products are marked with `is_deleted = 1`. This soft delete mechanism hides items from all public and most administrative lists, preserving data for potential recovery or auditing purposes.

? **Responsive UI with Tailwind & Framer Motion**: The user interface is fully responsive, adapting seamlessly to various screen sizes, thanks to Tailwind CSS. Framer Motion is integrated to provide smooth, engaging micro-animations, enhancing the overall user experience.

? **Environment-based API URL & .env Usage**: The application correctly utilizes environment variables for sensitive information like database credentials and API base URLs. This practice ensures secure configuration and easy adaptation across different deployment environments.

5. Architecture & Technology Stack

The application follows a clear separation of concerns, dividing responsibilities between the frontend and backend. This modular approach enhances maintainability, scalability, and team collaboration.

Frontend: Built with Next.js (App Router), the frontend features a component-based architecture with reusable UI elements. Dedicated API service layers handle communication with the backend, while client-side form validation and toast notifications provide immediate user feedback. Framer Motion is used for subtle, engaging animations.

Backend: Developed using Express.js, the backend adheres to the MVC (Model-View-Controller) pattern. Routes define API endpoints, controllers handle request logic, services encapsulate business logic, and direct database interactions are managed separately. Custom error handling middleware and async handlers ensure robust and predictable API behavior.

Architectural Diagram: Next.js (client) ? Express API ? MySQL (Railway)

6. Database Schema Snippet

The `products` table is central to the application, designed to store all product-related information efficiently. It includes fields for product details, status, and audit trails, along with a flag for soft deletion. ```sql

CREATE TABLE products (product_id   INT

AUTO_INCREMENT PRIMARY KEY, product_name

VARCHAR(100) NOT NULL, product_desc TEXT, status

ENUM('Draft','Published','Archived') DEFAULT 'Draft',

created_by   VARCHAR(50) NOT NULL, created_at

TIMESTAMP DEFAULT CURRENT_TIMESTAMP, updated_by

VARCHAR(50), updated_at   TIMESTAMP DEFAULT

CURRENT_TIMESTAMP ON UPDATE

CURRENT_TIMESTAMP, is_deleted   BOOLEAN DEFAULT

FALSE

);

```

Note: The 'Delete' operation in the application sets `is_deleted = 1`. Public-facing product lists and default admin views automatically exclude items where `is_deleted = 1` or `status = 'Draft'`.

## 7. API Endpoints

The backend exposes a set of RESTful API endpoints to manage product resources. These endpoints facilitate all CRUD operations and specific status toggles, ensuring a comprehensive interface for the frontend.

| Endpoint | Description |
|---|---|
| GET /api/products | Retrieves a list of products. Supports query parameters like `status=Published` or `status=Draft`. By default, excludes `is_deleted=1` items. |
| GET /api/products/:id | Fetches detailed information for a specific product by its `product_id`. |
| POST /api/products | Creates a new product entry in the database with provided name, description, and initial status. |
| PUT /api/products/:id | Updates an existing product's details (name, description, status) identified by `product_id`. |
| PATCH /api/products/:id/toggle | Toggles the `status` of a product between 'Draft' and 'Published'. |
| DELETE /api/products/:id | Performs a soft delete on a product by setting its `is_deleted` flag to `TRUE`. |

## How to Run Locally

To set up and run the project on your local machine, follow these steps. Ensure you have Node.js and npm installed.
```

1. **Clone the Repository**: Open your terminal and execute `git clonehttps://github.com/Rhythm82/cms-products.git` to download the project files.

2. **Create .env Files**: Navigate into both the `backend` and `frontend` directories. Create a `.env` file in each,configuring necessary environment variables such as `DATABASE_URL` for the backend and `NEXT_PUBLIC_API_BASE_URL` for the frontend.

3. **Run Backend**: Change directory to `backend` (`cd backend`), install dependencies with `npm install`, andstart the server using `npm run dev`. The API will typically run on `http://localhost:5000` (or your configured port).

4. **Run Frontend**: In a separate terminal, change directory to `frontend` (`cd frontend`), install dependencieswith `npm install`, and launch the development server using `npm run dev`. The frontend application will be accessible at `http://localhost:3000`.

8. Screenshots (Conceptual)

Below are descriptions of key application screens, illustrating the user interface and core functionalities. These conceptual screenshots highlight the clean design and intuitive user experience.

Screenshot 1: Home (Published Products): This image would display the public-facing home page, showcasing a list of products currently in 'Published' status. Each product would have a clear title, description snippet, and a call-to-action button, demonstrating the public view of available items.

Screenshot 2: Admin - Add Product Form: This screenshot would capture the administrative 'Add New Product' form. It would show input fields for 'Product Name', 'Description', and a 'Status' dropdown (Draft/Published), along with real-time validation messages for required fields, emphasizing the user-friendly input process.

Screenshot 3: Admin - Edit/Toggle Product: This image would feature the product editing interface within the admin panel. It would highlight the ability to modify product details and prominently display the 'Publish/Draft' toggle button, demonstrating how administrators can control product visibility.

Screenshot 4: Admin - Soft Delete Confirmation: This screenshot would show the admin product list after a soft delete operation. The deleted item would either be visually greyed out, moved to an 'Archived' filter, or simply absent from the default view, illustrating that it's hidden from public and active lists but not permanently removed.

9. Challenge & Fix

Challenge: During deployment to Render, the frontend application encountered persistent issues with API calls failing due to Cross-Origin Resource Sharing (CORS) policies and mixed HTTP/HTTPS content. Specifically, the Next.js frontend, served over HTTPS, was attempting to call the Express.js backend, which initially had inconsistent protocol handling, leading to blocked requests.

Fix: The solution involved several steps. On the Express.js backend, the `cors()` middleware was explicitly configured to allow requests from the specific frontend origin, ensuring proper cross-origin communication. Furthermore, the `NEXT_PUBLIC_API_BASE_URL` environment variable in the frontend was meticulously updated to use the correct HTTPS URL for the deployed backend. Network logs in browser DevTools were instrumental in diagnosing and verifying the successful resolution of these protocol and origin mismatches. A small retry mechanism was also added around fetch calls for increased resiliency against transient network issues.

Result: These adjustments led to stable and secure cross-origin API requests in the production environment, allowing the frontend and backend to communicate seamlessly after deployment.

10. Testing Notes for Evaluators

For demonstration purposes, the application currently operates without explicit user authentication. All administrative actions, such as adding, editing, and toggling product statuses, are directly accessible via the UI.

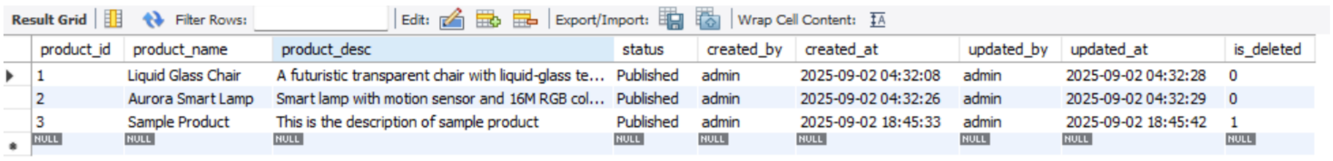
The 'Delete' functionality implemented is a soft delete. This means that products are not permanently removed from the database; instead, their `is_deleted` flag is set to `TRUE`. This action is reversible by manually updating the `is_deleted` flag back to `FALSE` in the database, allowing for data recovery.

Any changes made to a product's status, particularly toggling an item to 'Published', will reflect instantly on the public-facing 'Home' page, demonstrating real-time content updates.

11. Closing Remarks

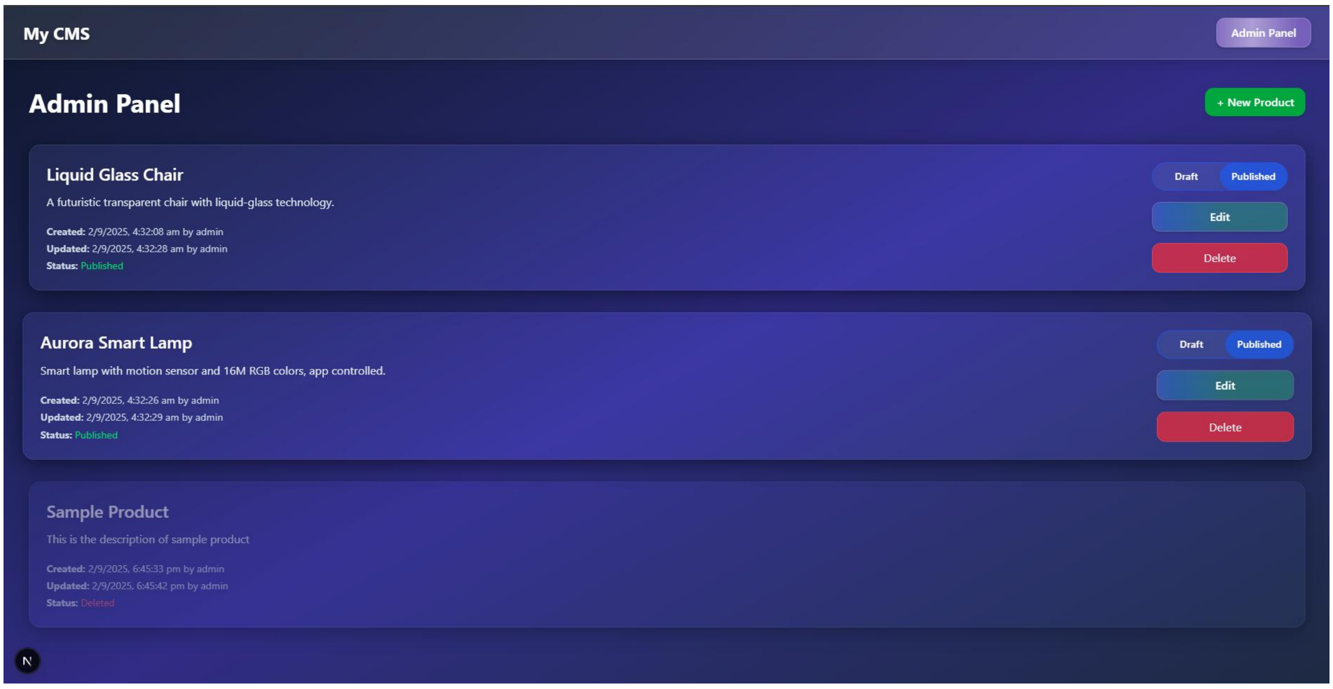
Thank you for taking the time to review my Internshala assignment. I hope this document provides a clear and comprehensive overview of the CMS Product CRUD application, its features, and the underlying technical implementation. I am confident in my ability to develop robust and scalable web solutions.

I am eager to discuss potential improvements and future enhancements for this project, such as implementing Role-Based Access Control (RBAC), advanced search and sorting functionalities, pagination for large datasets, and comprehensive audit logs. Your feedback is highly valued, and I look forward to the opportunity to elaborate further on my skills and experience.



Result Grid
Filter Rows:
Edit:
Export/Import:
Wrap Cell Content:

product_id	product_name	product_desc	status	created_by	created_at	updated_by	updated_at	is_deleted
1	Liquid Glass Chair	A futuristic transparent chair with liquid-glass te...	Published	admin	2025-09-02 04:32:08	admin	2025-09-02 04:32:28	0
2	Aurora Smart Lamp	Smart lamp with motion sensor and 16M RGB col...	Published	admin	2025-09-02 04:32:26	admin	2025-09-02 04:32:29	0
3	Sample Product	This is the description of sample product	Published	admin	2025-09-02 18:45:33	admin	2025-09-02 18:45:42	1
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL



Admin Panel

Liquid Glass Chair

A futuristic transparent chair with liquid-glass technology.

Created: 2/9/2025, 4:32:08 am by admin
Updated: 2/9/2025, 4:32:28 am by admin
Status: Published

Aurora Smart Lamp

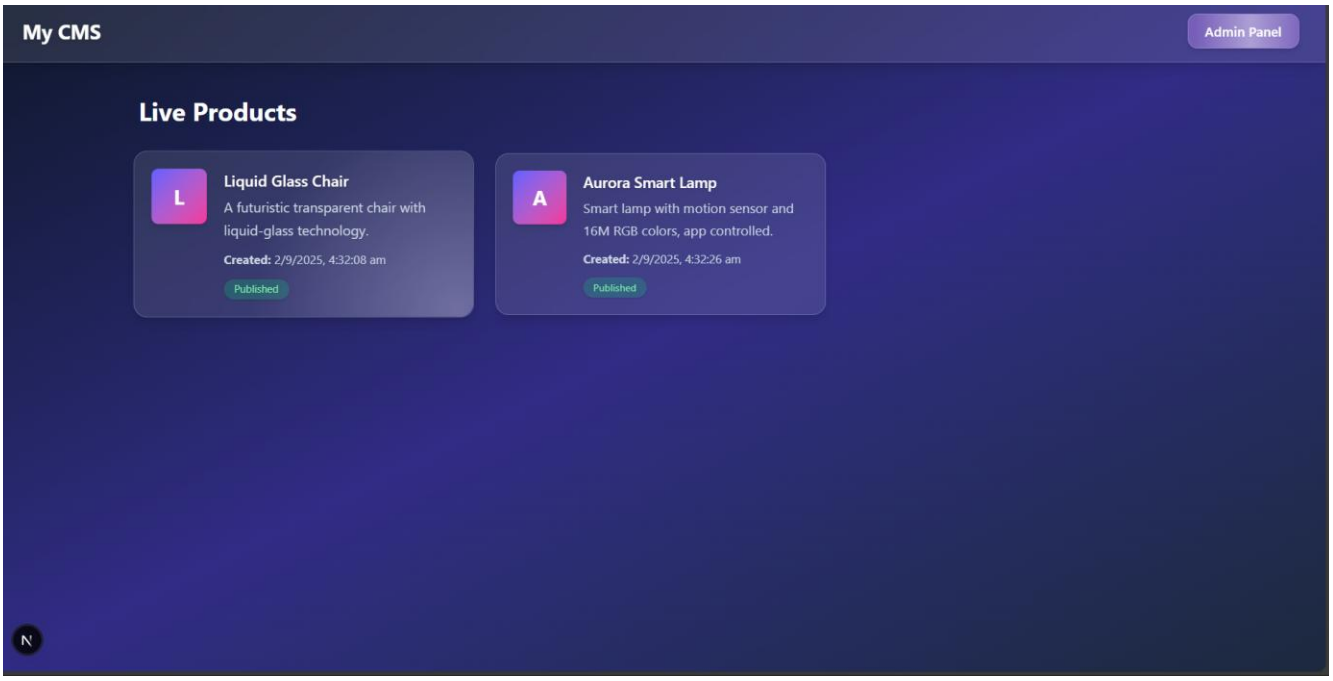
Smart lamp with motion sensor and 16M RGB colors, app controlled.

Created: 2/9/2025, 4:32:26 am by admin
Updated: 2/9/2025, 4:32:29 am by admin
Status: Published

Sample Product

This is the description of sample product

Created: 2/9/2025, 6:45:33 pm by admin
Updated: 2/9/2025, 6:45:42 pm by admin
Status: Deleted



Live Products

L
Liquid Glass Chair
A futuristic transparent chair with liquid-glass technology.
Created: 2/9/2025, 4:32:08 am
Published

A
Aurora Smart Lamp
Smart lamp with motion sensor and 16M RGB colors, app controlled.
Created: 2/9/2025, 4:32:26 am
Published