BHARAT COKING COAL LIMITED (BCCL)
A subsidiary of Coal India Limited
Dhanbad, Jharkhand

भारत कोकिंग कोल लिमिटेड
Bharat Coking Coal Limited

REPORT ON

DECOUPLED, MULTI-ROLE CONTENT MANAGEMENT SYSTEM FOR BCCL (FULLSTACK)

Devdeep Saha
COMPUTER SCIENCE AND ENGINEERING
B.TECH
(2022-2026)

ABACUS INSTITUTE OF ENGINEERING AND MANAGEMENT
MAGRA,HOOGHLY

DECLARATION

We hereby declare that the work, which is being presented in the project, titled

'Decoupled, Multi-Role Content Management System (CMS) for the Bharat Coking Coal Limited (BCCL) Website'

submitted by me (*Devdeep Saha*) to BCCL, Dhanbad and *Abacus Institute of Engineering and Management (Magra, Hooghly)*, is in fulfilment of the requirement for the award of Degree of B.Tech in the Department of Computer Science and Engineering

This is a record of the Internship Project carried out under the guidance of Mr. Rajeev Kumar, System Department, BCCL, Dhanbad. The information has been collected from genuine and authentic sources.

NAME: DEVDEEP SAHA
BRANCH COMPUTER SCIENCE AND ENGINEERING
REGISTRATION NO: 222400110013

भारत कोकिंग कोल लिमिटेड
Bharat Coking Coal Limited

BHARAT COKING COAL LIMITED

CERTIFICATE

This is to certify that this project report entitled "*Decoupled, Multi-Role Content Management System (CMS) for the Bharat Coking Coal Limited (BCCL) Website*" submitted by *Devdeep Saha* student of Computer Science And Engineering (2022-2026) from *Abacus Institute of Engineering and Management (Magra, Hooghly)* has undertaken an internship project for four weeks starting from 2nd September 2025 under our observation and guidance, with full dedication.

Here is a bonafide record of the project carried out by him under our supervision and guidance.

Rajeev Kumar
Deputy Manager System
Koyla Bhawan, BCCL
Dhanbad, Jharkhand

भारत कोकिंग कोल लिमिटेड
Bharat Coking Coal Limited

BHARAT COKING COAL LIMITED

CERTIFICATE

This is to certify that this project report entitled "*Decoupled, Multi-Role Content Management System (CMS) for the Bharat Coking Coal Limited (BCCL) Website*" submitted by *Devdeep Saha* student of Computer Science And Engineering (2022-2026) from *Abacus Institute of Engineering and Management (Magra, Hooghly)* has undertaken an internship project for four weeks starting from 2nd September 2025 under our observation and guidance, with full dedication.

Here is a bonafide record of the project carried out by him under our supervision and guidance.

Durga Prasad Mishra
GM (System)
Koyla Bhawan, BCCL
Dhanbad, Jharkhand

ACKNOWLEDGEMENT

It was my profound privilege to undertake the present project work at Bharat Coking Coal Limited and submit completion report here. I owe my deep gratitude to the System Department, BCCL Dhanbad for improving this project.

I am extremely grateful to my project guide Mr. RAJEEV KUMAR, Deputy Manager, System Department, BCCL, Dhanbad for assigning such a challenging project and for his consistent guidance and assistance which I had during the formulation and implementation of this project. The work we completed till now would certainly not have been possible without his valuable help and support.

I would also like to thank our college *Abacus Institute of Engineering and Management (Magra, Hooghly)* providing me an opportunity to work on this project.

Last but not the least, I would like to thank my parents, college faculties and all the staff members of the System Department for their help and their guidance.

NAME: DEVDEEP SAHA
BRANCH COMPUTER SCIENCE AND ENGINEERING
REGISTRATION NO: 222400110013

INDEX

ABSTRACT

This report details the successful architecture, development, and implementation of a comprehensive, full-stack web application for Bharat Coking Coal Limited (BCCL).

The project was built using a modern decoupled architecture, with a **React** frontend and a **Laravel** backend API.

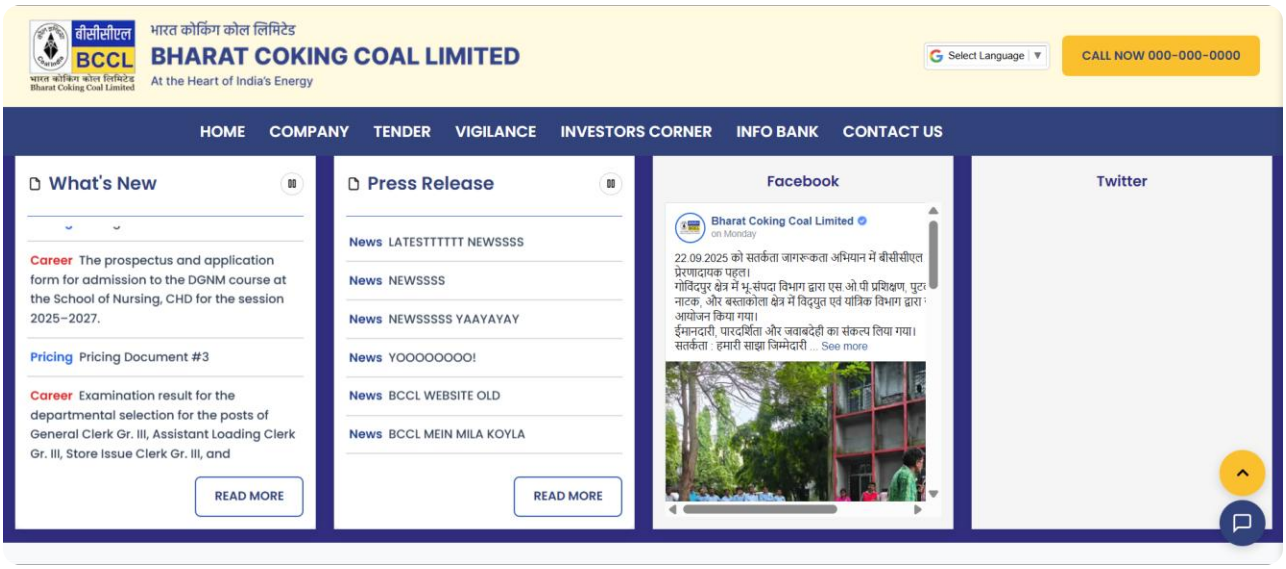
A key innovation is the integration of a standalone **Python** server hosting a hybrid AI chatbot, demonstrating a microservice-based approach.

The primary outcome is a dynamic, high-performance public website coupled with a secure, multi-role Content Management System (CMS). The CMS features a sophisticated "Departmental CMS" that empowers non-technical departmental administrators to manage their own specific sections of the website. The project successfully integrates advanced features including a hybrid AI chatbot, dynamic data tickers, and professional-grade admin interfaces, demonstrating a complete end-to-end development cycle from database design to final user interface polishing.

OBJECTIVE AND PURPOSE

The primary objective of this project was to modernize the digital presence of BCCL by replacing a static website with a dynamic, scalable, and easily manageable platform.

The key purposes were:
To Empower Non-Technical Users: To create a secure and intuitive admin panel where employees with different roles (Super Admin, Admin, Departmental Admin) could manage website content without needing any programming knowledge.
To Improve Information Dissemination: To build dynamic sections for News, Careers, Pricing Documents, and Events that could be updated in real-time.
To Increase Efficiency: To introduce a "Departmental CMS" to decentralize content management, allowing individual departments to take ownership of their web presence.
To Enhance User Engagement: To implement modern frontend features like a pausable hero slider, an AI chatbot, interactive lists, and a professional, responsive design to improve the experience for public visitors.



TOOLS AND PLATFORMS

Backend:

- **Language:** PHP 8+
- **Framework:** Laravel 10+
- **Database:** MySQL
- **Package Manager:** Composer
- **API Standard:** RESTful API
- **Authentication:** Laravel Sanctum (Token-based)

Frontend:

- **Library:** React 19+
- **Build Tool:** Vite
- **Routing:** React Router DOM
- **API Client:** Axios
- **State Management:** React Hooks(useState, useEffect), React Context, TanStack Query (React Query) for server state caching.
- **UI & Styling:** Custom CSS with Flexbox and Grid, react-bootstrap(for Modals)

Development Environment:

- **Local Server:** XAMPP (Apache, MySQL, PHP)
- **Code Editor:** Visual Studio Code
- **Terminal:** PowerShell / Git Bash

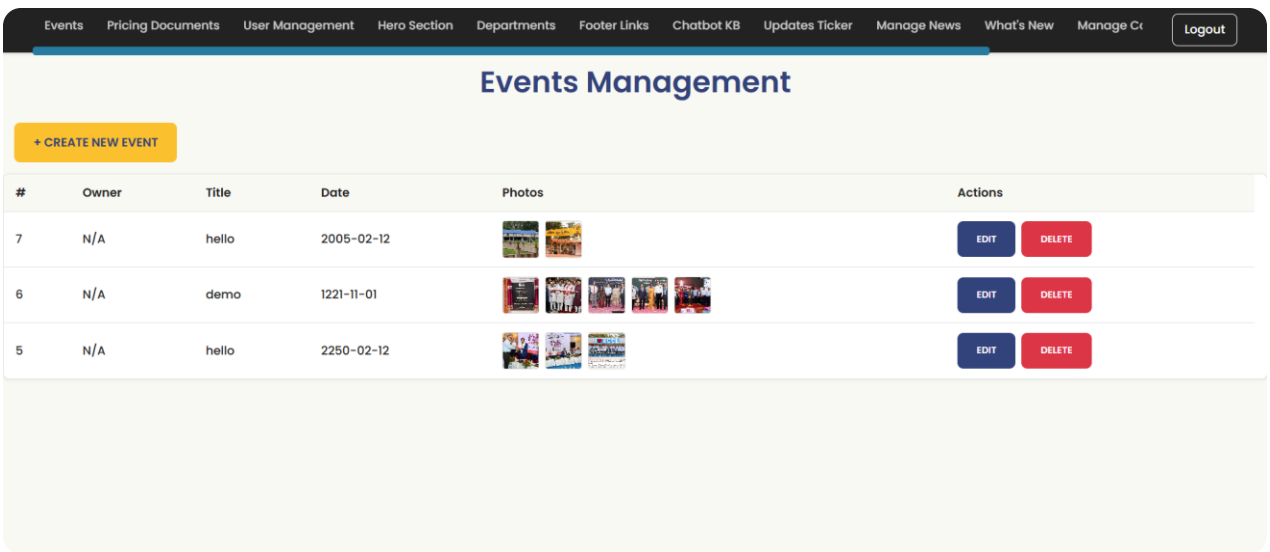
WORKING OF THE SYSTEM (HIGH-LEVEL ARCHITECTURE)

The application is built on a **decoupled (or headless) architecture**.

- **The Backend (The "Body"):** The Laravel application acts as a central content hub. It has no public-facing views of its own. Its sole purpose is to manage the database and provide a secure REST API. It handles all business logic, data validation, file storage, and user authorization.

- **The Frontend (The "Head"):** The React application is a completely separate project that runs in the user's browser. It is responsible for all visual presentation. It is "unaware" of the database or server-side logic; it only knows how to request data from and send data to the Laravel API endpoints.

- **Communication:** The two applications communicate securely over HTTP. When a user logs in, the React app sends credentials to a Laravel API endpoint, receives back a secure token, and stores it. For all subsequent requests to protected routes, React attaches this token, which Laravel Sanctum then validates.



DETAILED FRONTEND STRUCTURE (REACT)

The frontend is a modern Single-Page Application (SPA) designed for performance and maintainability.

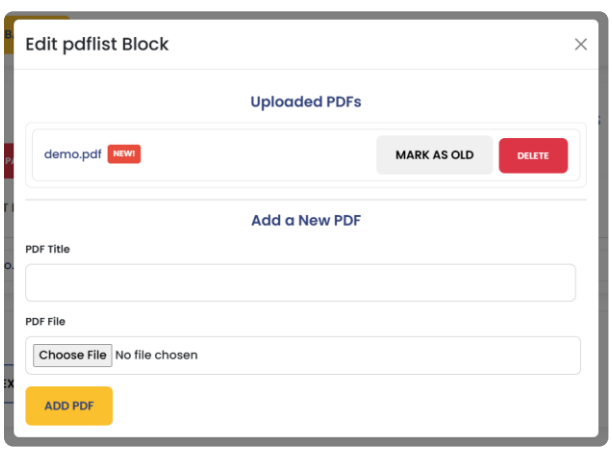
Component-Based Design: The UI is broken down into small, reusable components located in `src/components/`. This includes common elements (Header, Footer, Icon, ProgressBar) and feature-specific components (Ticker, FacebookFeed).

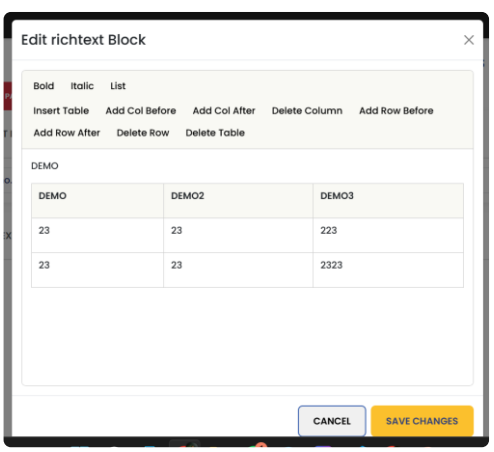
Routing: App.jsx serves as the main router, using `react-router-dom` to define all URL paths. A custom `RequireAuth.jsx` component acts as a gatekeeper for protected admin routes, checking the user's role before granting access.

State Management:

- **Local State:** `useState` and `useEffect` are used for managing the state within individual components (e.g., form inputs, modal visibility).
- **Global Auth State:** React Context (`AuthProvider`) is used to provide the logged-in user's information to the entire application.
- **Server State Caching: TanStack Query** is used for all major data-fetching operations (Hero Slides, Tickers, Events). This provides a robust in-memory cache, which eliminates redundant API calls, prevents reloading when navigating back to a page, and handles background data synchronization.

API Communication: A centralized Axios instance (`src/api/axiosConfig.js`) is configured with the base URL of the Laravel API. This makes API calls consistent and easy to manage.





DETAILED BACKEND STRUCTURE (LARAVEL)

The backend is a robust REST API built on the Model-View-Controller (MVC) pattern.

Database Schema (Model): The database structure is defined through **Migrations**.
Tables for `users`, `departments`, `directors`, `news_items`, `events`, `pricing_documents`, etc., were created.
Eloquent Models represent each table, and relationships (`belongsTo`, `hasMany`) are defined to create logical links between data (e.g., a `DepartmentPage` belongs to a `Department`).

API Endpoints (Routes & Controllers):

- **Routes:** The `routes/api.php` file defines all available API endpoints. Public routes are open, while admin routes are grouped under the `auth:sanctum` middleware.
- **Controllers:** Each major feature has a dedicated controller (e.g., `DirectorController`, `NewsController`). These controllers handle the request lifecycle: receiving a request, validating data, performing the business logic, and returning a JSON response.

Authorization (Policies & Gates): The security model is a key feature.

- **Gates:** Used for simple, role-based checks (e.g., the `isSuperAdmin` gate checks if a user's role is 'super-admin').
- **Policies:** Used for complex, model-based authorization. For example, the `DepartmentPagePolicy` contains a rule that checks if the logged-in user's `department_id` matches the `department_id` of the page they are trying to edit. This ensures a departmental admin can *only* manage their own content.

File Management: The `Storage` facade is used for all file uploads. A symbolic link (`php artisan storage:link`) makes the `storage/app/public` directory accessible from the web, allowing images and PDFs to be served to the frontend.

AI CHATBOT ARCHITECTURE

A significant feature of this project is the "CoalBot," an AI-powered assistant built as a standalone microservice to ensure scalability and separation of concerns.

Technology Stack: The chatbot's core logic is built in **Python**, utilizing a web framework like Flask or FastAPI to create its own API. This runs on a separate server process from the main Laravel application.

Hybrid Response System: The chatbot employs a two-tiered logic system for answering user queries, designed for both speed and accuracy:

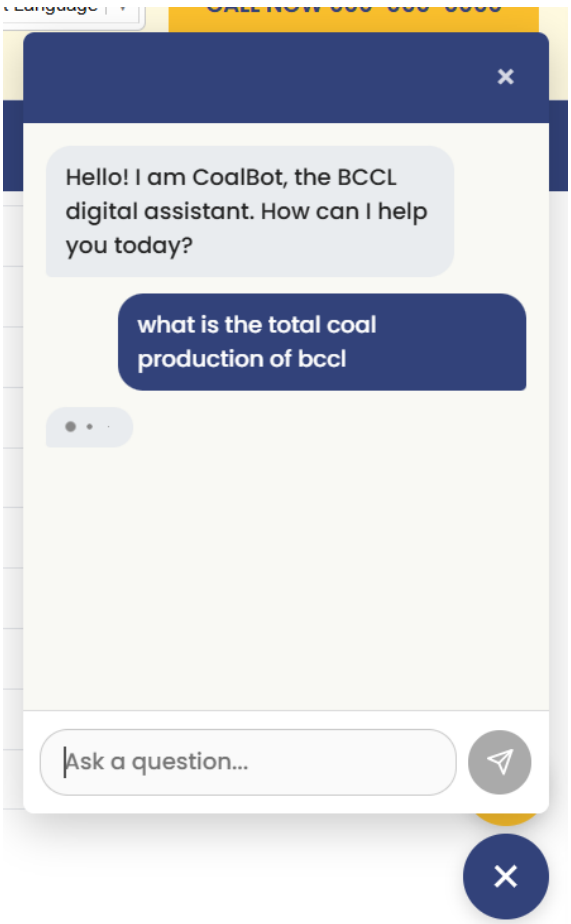
- **Managed Knowledge Base:** When a query is received, it is first checked against a curated knowledge base (a JSON file managed by the Laravel CMS). This contains instant, pre-approved answers to common questions (e.g., "What are the latest tenders?", "Who is the CMD?").
- **Generative AI Fallback:** If the query is not found in the local knowledge base, the request is securely forwarded to the **Google Gemini AI** model. This allows the chatbot to handle a virtually limitless range of complex, conversational questions.

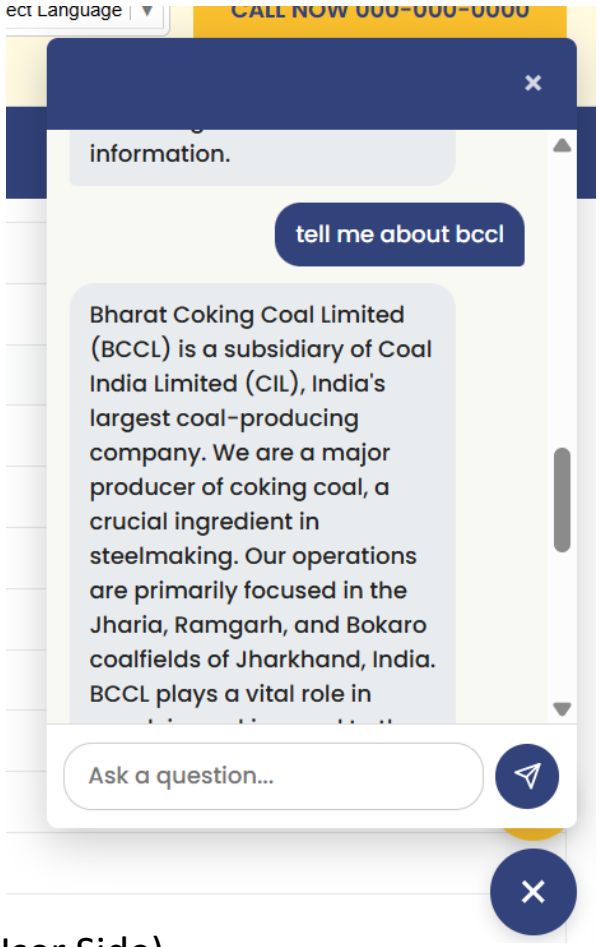
Request Flow: The interaction follows a clear microservice pattern:

1. The user types a message into the **React** chat widget.
2. The React app sends the message to a dedicated endpoint on the **Laravel API**.
3. The Laravel controller first checks its local knowledge base. If no match is found, it acts as a proxy, securely forwarding the request to the **Python chatbot server**.
4. The Python server gets the response from the Gemini AI and sends it back to Laravel.
5. Laravel returns the final answer to the React frontend, which displays it to the user.

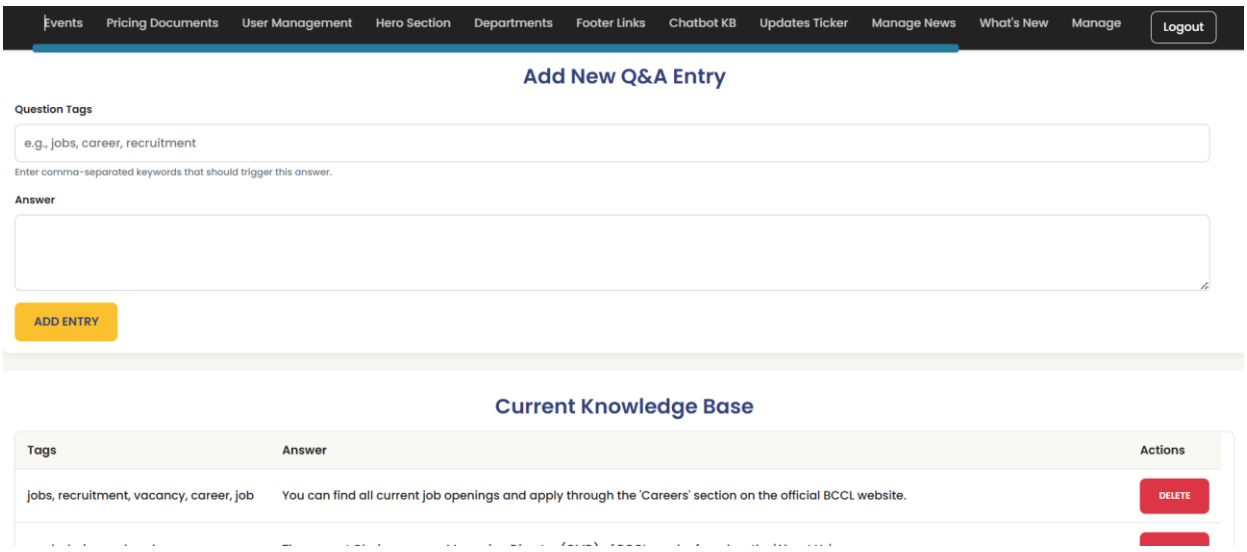
This architecture ensures that the main website's performance is not impacted by the AI model's processing time and allows the chatbot to be developed and scaled independently.

WORKING CHATBOT





Front-End (User Side)



Front-End (Admin Panel)

FEATURES AND FUNCTIONALITY

The final application delivers a complete suite of features for both public users and administrators.

Public Website:

- Fully responsive, multi-page corporate website.
- Dynamic homepage with a pausable hero slider and configurable, scrolling news/update tickers.
- Dedicated, paginated pages for Careers, News, Pricing Documents, and Events.
- A detailed Board of Directors page with a special layout for the CMD.
- A comprehensive sitemap and a dynamic footer.
- An integrated AI Chatbot and Google Translate widget.

Super Admin Panel:

- Full control over all content modules (News, Events, Careers, etc.).
- User management interface to create/delete admins and assign roles.
- Department management interface.
- Board of Directors management with drag-and-drop reordering.
- Homepage Ticker configuration panel to select multiple content sources and set item limits.

Departmental Admin Panel:

- A simplified, dedicated dashboard.
- Ability to manage a custom banner image.
- A page builder to create and manage their own sidebar navigation and page content using text, image, and PDF list blocks.
- Control over "New!" badges for their PDF uploads.

AI Chatbot (CoalBot):

Integrated a chat widget with a hybrid response system, connecting the React frontend to a standalone **Python microservice** that leverages both a managed knowledge base and the Google Gemini AI.

ARCHITECTURAL DECISIONS

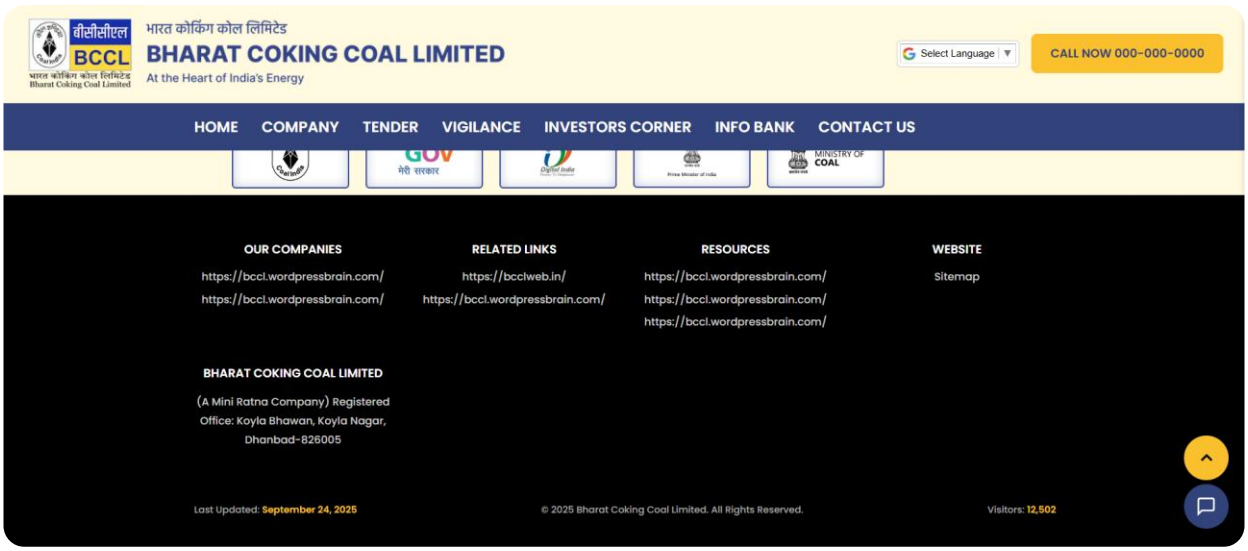
Key strategic choices were made early in the project to define its structure and capabilities:

Decoupled (Headless) Architecture: The decision was made to build the application as a decoupled system with a **React frontend** and a **Laravel backend API**.

> **Reasoning:** This separates the presentation layer (the "head") from the data and logic layer (the "body"). This allows for independent development, deployment, and scaling of the frontend and backend. It also creates a more modern, faster, Single-Page Application (SPA) experience for the end-user.
> **Outcome:** A high-performance public website and a secure, centralized API that can potentially serve other applications (like a mobile app) in the future.

Microservice for AI Chatbot: The Python-based AI chatbot was intentionally designed as a standalone microservice that runs on its own server process.

> **Reasoning:** AI and machine learning models can be resource-intensive. Isolating the chatbot prevents its processing load from impacting the performance and stability of the main Laravel web server.
> **Outcome:** A resilient system where the main website's functionality is not dependent on the AI service. The chatbot can be updated, scaled, or even taken down for maintenance without affecting the rest of the BCCL site.



Granular Authorization with Policies & Gates: A multi-layered authorization system was implemented in Laravel to handle the different user roles.

Reasoning: The application required both simple role checks and complex, object-specific permissions.

Outcome:

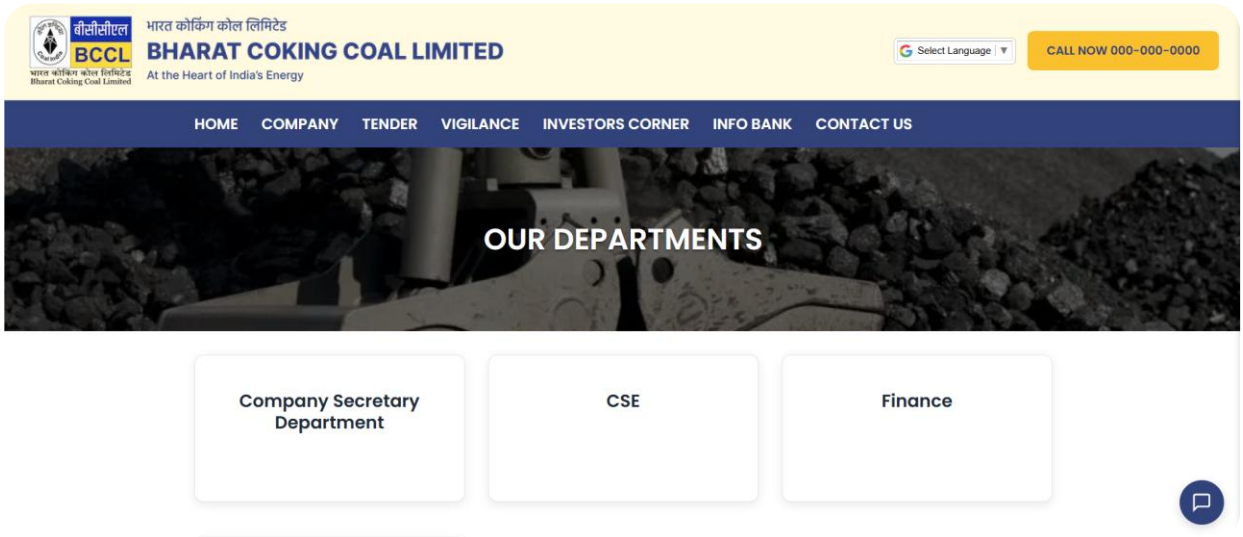
Laravel Gates were used for simple, role-based checks (e.g., `isSuperAdmin`) that apply across the application.

Laravel Policies were used for complex, model-based rules. The prime example is the `DepartmentPagePolicy`, which contains the logic: *"A user can only update or delete a page if that user's* `department_id` *matches the page's* `department_id`*."* This provides fine-grained control and is the core of the secure Departmental CMS.

Server-State Management with TanStack Query: After initial development, the decision was made to refactor the frontend's data-fetching logic using the TanStack Query (React Query) library.

Reasoning: Initial data fetching with `useState` and `useEffect` led to unnecessary re-loading when navigating between pages, resulting in a sluggish user experience.

Outcome: TanStack Query introduced a robust client-side cache. This provides an instant loading experience when users revisit pages, while intelligently re-fetching data in the background. It also significantly simplified the frontend code by automating loading and error state management.



TECHNICAL CHALLENGES & SOLUTIONS

The development process involved diagnosing and resolving a wide range of real-world technical issues:

Environment & Server Configuration:

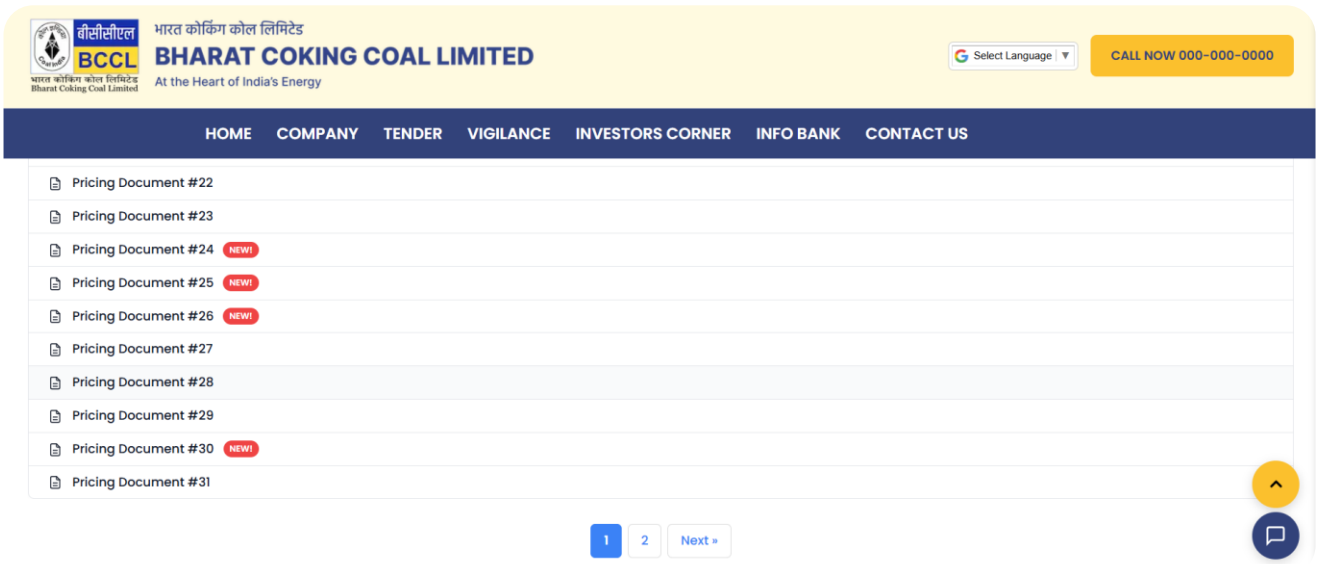
Challenge: The application crashed due to missing PHP extensions on the local server.

Solution: Diagnosed a Call to undefined function **imagecreatefromjpeg()** error, identified the missing **GD library**, and successfully enabled it by editing the server's php.ini file. Similarly, solved file upload validation errors by increasing the **upload_max_filesize** and **post_max_size** directives in php.ini.

Routing and API Mismatches (4xx Errors):

Challenge: Persistent 404 (Not Found) and 405 (Method Not Allowed) errors when the frontend tried to communicate with the backend.

Solution: Performed systematic debugging using browser Network tools. This revealed mismatches between the frontend's Axios calls (e.g., sending a POST for an update) and the backend's **routes/api.php** definitions (which expected PUT/PATCH). The issue was resolved by creating explicit routes to handle the specific requests from the frontend. A recurring issue with "zombie" server processes was solved by performing a "hard reset" (killing all php.exe processes and clearing all caches).



Database Integrity & Migrations (500 Errors):

Challenge: The application frequently crashed with 500 Internal Server Errors, traced back to database issues.

Solution: Inspected Laravel logs (**storage/logs/laravel.log**) to find the root cause.

Fixed Column not found and Duplicate column errors by refactoring and cleaning up redundant **database migrations**.

Resolved Duplicate entry errors by implementing logic in the controller to generate a unique **slug** (e.g., title-plus-id).

Solved a critical silent bug where content would not delete. The cause was a missing **onDelete('cascade')** constraint in a migration, which was identified and fixed by running php artisan migrate:fresh.

Frontend State and Rendering:

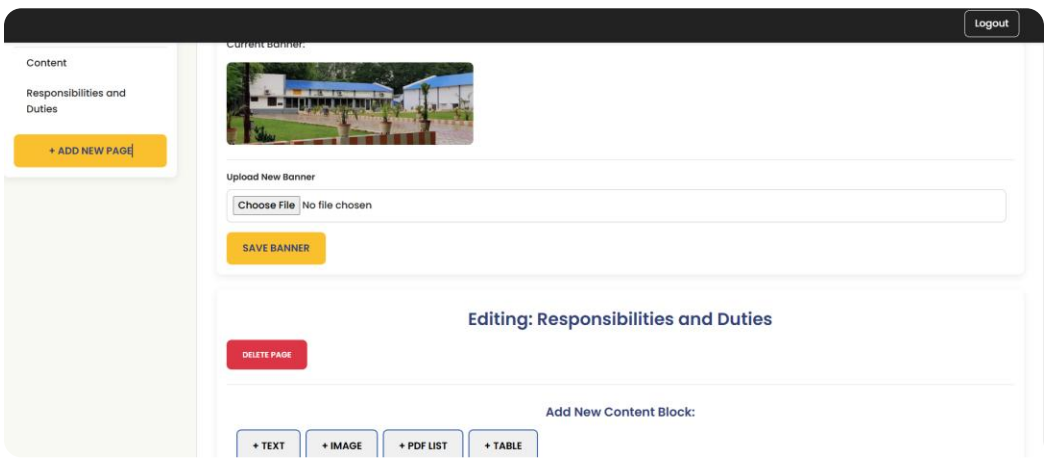
Challenge: The UI would crash with a "White Screen of Death" or display incorrect, "stale" data.

Solution:

Diagnosed and fixed the WSOD by resolving a CSS class name conflict between two different ticker components.

Solved the "stale state" bug in the Departmental CMS page builder, where child components were not updating, by implementing a useEffect hook to synchronize the component's internal state with incoming props. This is a critical React concept that was successfully applied.

Fixed a bug where a 0 was rendered on the page by changing a conditional render from **condition && <Component/>** to the more robust ternary operator **condition ? <Component/> : null.**



CONCLUSION

This project successfully met and exceeded all initial objectives. It demonstrates the power of a decoupled architecture to create a fast, modern user experience with a robust and secure backend.

The development journey involved overcoming numerous real-world challenges, from server configuration and database design to advanced frontend state management and API security.

The resulting application is a feature-rich, scalable, and maintainable platform that provides significant value to BCCL and its users. The process also represents a significant personal achievement in learning and mastering full-stack development with PHP, Laravel, and React.