BeyondChats – Full Stack Engineer / Technical Product Manager Assignment

Candidate: Rushikesh Dharme

GitHub Repository — Frontend Demo — Backend API

Introduction

This document presents my submission for the BeyondChats assignment. The objective was to design and implement a system that demonstrates backend engineering, frontend integration, data scraping, and AI-driven content workflows — while making realistic engineering trade-offs under time constraints.

The solution focuses on correctness, system design clarity, and production-oriented deployment rather than superficial completeness.

—

Project Overview

The project is structured into three logical parts:

1. **Laravel Backend**
    - Scrapes the 5 oldest BeyondChats blog articles
    - Stores structured data in MySQL
    - Exposes RESTful CRUD APIs
2. **React Frontend**
    - Fetches data from Laravel APIs
    - Displays original and generated articles
    - Simple, responsive UI focused on clarity
3. **AI Content Processing (Designed & Documented)**
    - Node.js service to analyze latest articles
    - Competitor content discovery
    - LLM-based article improvement
    - Publishing enhanced content back to backend

—

Tech Stack

Backend

- Laravel 9 (PHP 8)
- MySQL (Cloud-hosted)
- Guzzle HTTP Client

- Symfony DomCrawler

Frontend

- React (Create React App)
- Fetch API
- CSS (responsive layout)

AI / Automation (Planned)

- Node.js
- OpenAI-compatible LLM
- Google Search / SERP APIs

—

System Architecture & Data Flow

Phase 1: Scraping and Storage

1. Laravel fetches the BeyondChats blog listing page
2. Extracts all blog URLs
3. Selects the 5 oldest articles
4. Scrapes article title and main content
5. Stores data in the `articles` table with metadata

—

Phase 2: AI-Based Content Improvement

1. Node.js fetches the latest article from backend API
2. Searches for top-ranking competitor articles
3. Extracts and summarizes competitor content
4. Uses an LLM to enhance the original article
5. Stores generated content as a new article version
6. Attaches competitor reference URLs

This phase is intentionally designed rather than over-engineered, reflecting realistic product iteration strategies.

—

Phase 3: Frontend Presentation

1. React frontend consumes backend APIs
2. Articles are rendered as cards
3. Visual distinction between original and generated content

—

Database Design

articles table

Column	Type
id	bigint (PK)
title	varchar
content	longtext
source_url	varchar
source_type	enum (original, generated)
reference_urls	json (nullable)
created_at	timestamp
updated_at	timestamp

—

API Endpoints

Scraping

- `GET /api/scrape-articles`

  Articles CRUD

- `GET /api/articles`
- `GET /api/articles/id`
- `POST /api/articles`
- `PUT /api/articles/id`
- `DELETE /api/articles/id`

—

Deployment

- Backend deployed on Render using Docker
- MySQL hosted on a cloud-managed provider
- Frontend deployed on Netlify

All components are publicly accessible and production-configured.
—

Engineering Trade-offs

- Prioritized reliable backend pipelines over rushed AI integration
- Chose simple frontend tooling to reduce instability
- Designed AI workflow for scalability rather than demo-only behavior

—

Conclusion

This submission reflects my real-world engineering mindset:

- Build incrementally
- Make conscious architectural decisions
- Prefer correctness, clarity, and maintainability

Thank you for reviewing my work. I would be happy to walk through any part of the system in more detail.

Rushikesh Dharme