Full Stack Developer - CASE STUDY

Dynamic LiveOps Management System

1 Overview

You are tasked with building a modular full-stack web application using Spring Boot for the backend and Vue.js for the frontend.
The system will serve as a LiveOps management platform, enabling the creation, listing, editing, and deletion (CRUD) of various dynamic content collections through a generic interface.

The application must support multiple content models without requiring custom CRUD logic for each one.

Technical Stack

- Backend: Spring Boot
- Frontend: Vue.js
- Database: MongoDB

---

2 Requirements

Backend

- Provide a generic CRUD API capable of operating on any collection based on a collection name parameter.
- The backend should be capable of dynamically handling different content structures at runtime. While the actual models are maintained on the backend, the system must be designed in a way that these structures can be processed generically.
- Include a generated API that reads the structure of backend model classes and stores them in MongoDB. This API is not triggered from the frontend, but should be called manually (e.g., via Swagger or Postman) after models are created to register their structure.
- Include an endpoint that returns the structure/schema of each collection, so the frontend can render forms and tables dynamically.

Frontend

Build a dynamic UI that:

- Lists all available content collections in the sidebar.
- Renders form fields and data tables dynamically based on structure data retrieved from the backend, without relying on any predefined models.
- Reuses the same components across collections in a model-agnostic way.
- Supports full Create, Read, Update, and Delete (CRUD) functionality for each collection.
- Is responsive and adapts to various screen sizes and modern browsers.
- Automatically refreshes the displayed state after performing CRUD operations to reflect the latest changes.

LiveOps Models

The following models must be included in the project and will be used as examples for LiveOps content:

Offer: Represents a purchasable item with eligibility requirements.
PurchaseProduct: Defines purchasable content linked to offers.
Cascade: Represents a step-based progression system with grouped steps and multiple rewards.

These models are provided as separate JSON files (offer_model.json, purchaseproduct_model.json, cascade_model_multiple_rewards.json) for reference and structure demonstration.

Enum Definitions:
- TradeType: CURRENCY, ITEM, BADGE, XP
- EventType: LOGIN, MISSION_COMPLETE, LEVEL_UP, TIME_SPENT
- Requirement: minLevel, daysSinceRegistration, playTimeHours, lastLoginDaysAgo, hasSubscription, region, hasPremiumAccess

Deliverables

1. **A working full-stack application uploaded to a public GitHub repository. The candidate should share the repository link.**
2. **Setup instructions (README) to run the project locally.**
3. **API documentation (Postman collection or Swagger preferred).**

Evaluation Criteria

- **Proper use of generic patterns for CRUD operations.**
- **Dynamic rendering of forms and tables based on backend-provided structures.**
- **Ability to manage multiple collections without hardcoded logic.**
- **Clean and user-friendly frontend.**
- **Follows clean code and SOLID principles.**
- **Backend implementation using Clean Architecture is expected.**
- **Writing unit and/or integration tests for backend logic will be considered a plus.**

You have 7 days to complete this work.

Do not hesitate to contact us for the parts you do not understand.

We wish you good luck and hope you enjoy it while doing.

All materials shared in Case Study are intellectual property rights belonging to Ace Games.

In terms of the confidentiality of the project, do not share the files and your outputs with third parties.

Thank you for your sensitivity.