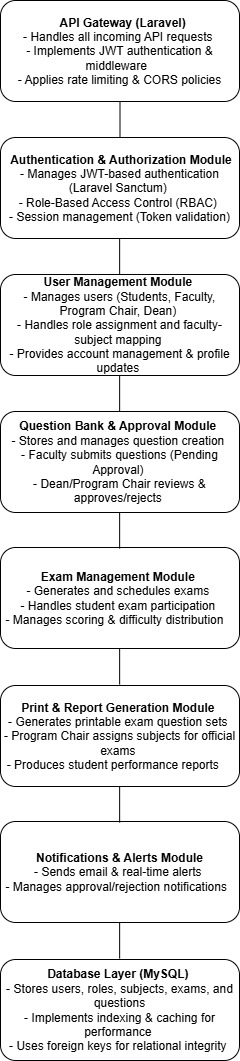
**Backend System Architecture (Modular Monolithic)**



**Detailed Explanation of the Modular Monolithic Backend System Architecture for CAPS**

This **backend architecture** follows a **Modular Monolithic** approach, where different functionalities are split into **separate modules** but still operate within a **single Laravel application**. This ensures **maintainability, security, and scalability** while keeping the system **efficient and easy to manage**.

**1️. API Gateway (Laravel)**

**Purpose:** The **entry point** for all frontend requests and external API calls.  
**Responsibilities:**

* **Handles all incoming API requests** from frontend clients (web app, mobile app).
* **Implements JWT authentication** (via Laravel Sanctum).
* **Applies middleware** for security (CORS, rate limiting).
* **Routes requests to the correct module** based on the request type.

**How it Works:**

1. The frontend sends a **request** (e.g., user login, fetch questions, submit an exam).
2. The **API Gateway validates authentication** (via JWT token).
3. If the request is **authorized**, it is passed to the relevant module.
4. The module processes the request and **sends a response back**.

**2️. Authentication & Authorization Module**

**Purpose:** Ensures that only **authenticated and authorized** users can access the system.  
**Responsibilities:**

* **JWT-based authentication** (via Laravel Sanctum).
* **Role-Based Access Control (RBAC)** for users (Student, Faculty, Program Chair, Dean).
* **Session management** (token validation and expiration).
* **Encrypts passwords** using **Laravel’s Hashing** system.

**How it Works:**

1. A **user logs in** and receives a **JWT token**.
2. The token contains **role-based access data** (e.g., "role": "Faculty").
3. All future requests **must include the token** in the Authorization: Bearer {token} header.
4. Middleware **verifies** the token and **checks user role** before processing.

**Security Features:**

* **Token Expiration** → Prevents long-term unauthorized access.
* **Rate Limiting** → Prevents brute-force attacks.
* **CORS & CSRF Protection** → Ensures secure API requests.

**3️. User Management Module**

**Purpose:** Handles **user registration, profile management, and role assignments**.  
**Responsibilities:**

* Manages **students, faculty, program chairs, and deans**.
* Assigns **roles and permissions** dynamically.
* Handles **faculty-subject assignments** (which faculty manages which subjects).
* Provides an **admin panel** for managing user accounts.

**How it Works:**

1. An **admin (Dean)** assigns **roles** to new users.
2. Faculty members are **linked to subjects** for question creation.
3. Users can **update their profiles** (change password, update email, etc.).
4. The system **ensures only authorized users** can access or modify user data.

**4️. Question Bank & Approval Module**

**Purpose:** Manages **question creation, approval workflow, and modifications**.  
**Responsibilities:**

* **Faculty** submits questions (**Pending Approval** status).
* **Program Chair/Dean** reviews and approves/rejects questions.
* Tracks **modifications** to questions and choices.
* Stores **difficulty level** and **coverage type** (e.g., Midterm, Final).

**How it Works:**

1. A **Faculty member** creates a new question.
2. The **question is submitted for approval** (Pending status).
3. The **Program Chair or Dean reviews** the question.
4. If approved, the question is **available for exams**.
5. If rejected, the faculty **must revise** the question before resubmitting.

**Security Features:**

* **Versioning** → Keeps track of **modified questions**.
* **Approval Logs** → Logs all approval/rejection decisions.

**5️. Exam Management Module**

**Purpose:** Generates and manages **exams, practice sessions, and grading**.  
**Responsibilities:**

* Creates **Practice Exams & Qualifying Exams**.
* Manages **student exam sessions** (start, submit, auto-save).
* Implements **difficulty-based question selection** (e.g., 20% Easy, 50% Moderate, 30% Hard).
* Handles **exam scoring and analytics**.

**How it Works:**

1. A **student selects a subject** and starts an exam.
2. The system **randomly selects questions** based on difficulty distribution.
3. The student **answers questions** and submits the exam.
4. The system **auto-grades** multiple-choice questions.
5. The student **reviews their performance** after submission.

**Performance Optimization:**

* Uses **database indexing** to **speed up question retrieval**.
* Implements **caching** for frequently accessed exams.

**6️. Print & Report Generation Module**

**Purpose:** Allows **faculty and program chairs** to print question sets and generate reports.  
**Responsibilities:**

* **Faculty prints personal exam sets** for students.
* **Program Chair selects subjects for official exams**.
* Generates **performance analytics reports**.

**How it Works:**

1. A **faculty member** selects **questions to print**.
2. The system **formats the questions into a PDF**.
3. The **Program Chair approves the final exam selection**.
4. **Reports are generated** for student performance analysis.

**Security Features:**

* **Access Control** → Only **Program Chair/Dean** can print official exams.
* **Logging System** → Tracks all **printed exams and reports**.

**7️. Notifications & Alerts Module**

**Purpose:** Manages **real-time notifications and email alerts** for users.  
**Responsibilities:**

* Sends **email alerts** for approvals, exam results, pending requests.
* Manages **real-time notifications** in the system.
* **Faculty gets notified** when questions are **approved/rejected**.
* **Students get notified** when **exam results are available**.

**How it Works:**

1. A faculty member **submits a question**.
2. The **Dean/Program Chair approves/rejects** the question.
3. The faculty **receives an email notification**.
4. A student **completes an exam**, and **gets notified** when results are ready.

**Performance Optimization:**

* Uses **Laravel Queues** for **asynchronous email sending**.
* Uses **WebSockets** for **real-time notifications**.

**8️. Database Layer (MySQL)**

**Purpose:** Stores **users, roles, subjects, exams, questions, and approvals**.  
**Responsibilities:**

* Ensures **data integrity** with **foreign keys**.
* Uses **indexing for fast query execution**.
* Implements **query caching** for frequently accessed data.

**Database Optimization:**

| **Tables** | **Indexing Strategy** |
| --- | --- |
| users | INDEX (userCode), INDEX (email) |
| questions | FULLTEXT INDEX (questionText) |
| exams | INDEX (studentCode, subjectCode) |
| notifications | INDEX (userCode, isRead) |

**How it Works:**

1. When a **user logs in**, the system **retrieves user data quickly**.
2. When an **exam starts**, the system **fetches randomized questions efficiently**.
3. When a **report is generated**, the system **optimizes database queries**.

**Why This Architecture?**

**Modular & Maintainable** → Each module is independent and scalable.  
**Secure** → Uses JWT authentication, RBAC authorization, and rate limiting.  
**Performance Optimized** → Uses indexing, caching, and efficient queries.  
**Future-Ready** → Can transition to **Microservices** if needed in the future.