Modern Application Development II - Project Report

Author Details

Name – Rishav Bairagya
Roll no – 22f3000947
Student email - 22f3000947@ds.study.iitm.ac.in
GitHub repo - https://github.com/Vahsir7/Quiz-master-v2.git

Project Demonstration video:	**[Click here link]**

Link: https://drive.google.com/file/d/1R7mER6E6IJyuTWztPEZU0rpmPO6vCDNv/view?usp=sharing

1. Project Details

Project Title: Quiz Master v2

Problem Statement

The goal of this project is to develop a comprehensive, multi-user web application that serves as an exam preparation site. The system supports two distinct user roles: a single **Administrator** with root access and multiple **Students** who can register and take quizzes.

- **Administrator Role**: The admin is responsible for all content management, including creating and managing subjects, chapters, quizzes, and questions. They must also have oversight of all registered users and access to summary analytics and send mail to respective students.
- **User Role**: Students must be able to register for an account, log in, browse available quizzes, attempt them under timed conditions, and view their scores and performance history.

The application is secure and provides a responsive user experience.

AI usage declaration: 40% usage (help used in CSS Styling and VueJs implementation). Percentage calculated based on the metric provided “App Dev Project AI/LLM percentage check” document.

Approach and Implementation

To address the problem statement, I have built APIs with Flask and a frontend Single Page Application (SPA) built with Vue.js. This approach aligns with the core requirements and allows for independent development and scaling of the two parts of the application.

Backend Architecture (Flask)

The backend was built using Flask, following a modular, blueprint-based structure to separate concerns for different parts of the application (authentication, admin functions, and student functions).

1. **Database and ORM**: As per the requirements, **SQLite** was used for data storage. **SQLAlchemy** was chosen as the Object-Relational Mapper (ORM) to programmatically define the database schema and manage all database interactions in an object-oriented way.
2. **API Design**: A comprehensive API was designed to handle all data operations (CRUD for subjects, chapters, quizzes, etc.). This decouples the backend from the frontend.
3. **Token-Based Authentication**: Security is managed using **JSON Web Tokens (JWT)**. Upon successful login, the server issues a token containing the user's ID and role. A custom decorator (@authentication) protects all sensitive endpoints and enforces role-based access control, ensuring admins and students can only access their designated resources.
4. **Asynchronous Backend Jobs**: For long-running tasks that should not block the user interface, such as

sending email reports or exporting CSV files, **Celery** was integrated with **Redis** as its message broker. This ensures the application remains responsive while these jobs run in the background. The implementation includes:
    - Daily reminders for new or pending quizzes.
    - Scheduled monthly performance reports sent via email.
    - User-triggered asynchronous export of quiz history to CSV.
5. **Performance and Caching**: To improve API response times and reduce database load, **Redis** was also implemented as a cache backend. Caching was strategically applied to read-heavy endpoints that fetch data which does not change frequently (e.g., lists of subjects and exams).

Frontend Architecture (Vue.js)

The frontend is a dynamic SPA built with Vue.js, providing a fast and interactive user experience.

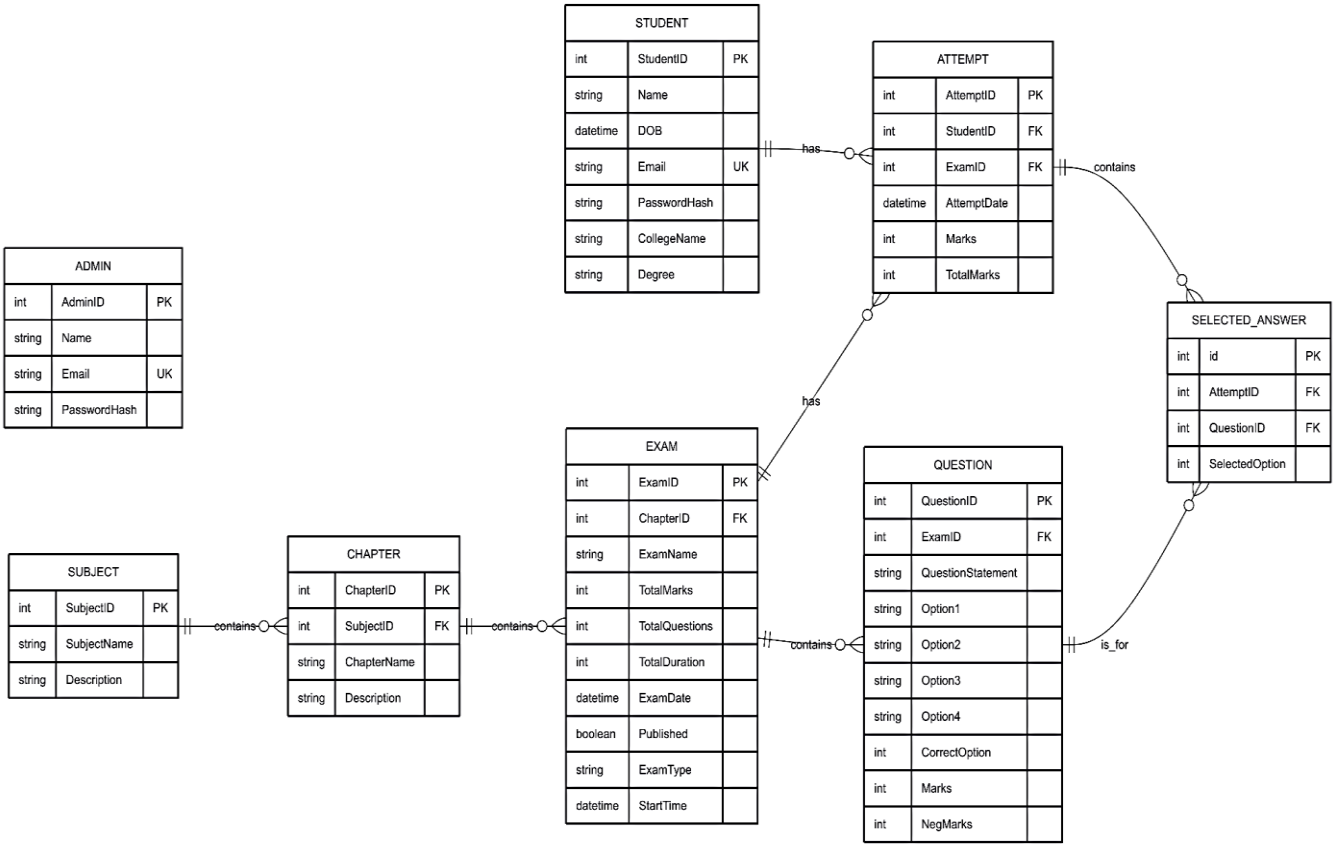
1. **Component-Based UI**: The user interface was built using reusable css components.
2. **Client-Side Routing**: **Vue Router** handles all navigation, mapping browser URLs to specific Vue components. It uses navigation guards to protect routes based on user authentication status and role, redirecting unauthorized users to the login page.
3. **State Management**: For managing complex, shared application state—particularly the timer, answers, and current question during an active quiz—**Vuex** was implemented as a centralized state management solution.
4. **Data Visualization**: The admin and student dashboards feature dynamic charts to visualize performance metrics. This was achieved using **Chart.js**, which integrates smoothly with Vue to create interactive bar, line, and doughnut charts.

2. Frameworks and Libraries Used

Category	Technology	Description
Backend	**Flask**	A lightweight and flexible Python web framework used to build the core API.
	SQLAlchemy	A powerful ORM that provides a high-level interface to the database.
	Celery	A distributed task queue for handling asynchronous background jobs.
	Redis	An in-memory data store used as the message broker for Celery and as a cache backend.
	PyJWT	A library for encoding and decoding JSON Web Tokens for secure authentication.
	Flask-CORS	Flask extension for handling Cross-Origin Resource Sharing.
	Flask-Bcrypt	A Flask extension for hashing passwords securely.
Frontend	**Vue.js**	A progressive JavaScript framework for building the interactive user interface.
	Vue Router	The official routing library for Vue.js, used for creating the SPA navigation.
	Vuex	The official state management library for Vue.js, used for managing application state.
	Axios	A promise-based HTTP client for making API requests to the backend.
	Chart.js	A library for creating responsive and interactive charts and graphs.
Database	**SQLite**	The mandated self-contained, serverless SQL database engine.

3. Database ER Diagram

The following diagram illustrates the database schema, which was programmatically created using SQLAlchemy models. It includes all required tables and their relationships, designed to be normalized for data integrity.



4. Project structure

```
Quiz-master-v2/
├── backend/                # Flask backend
│   ├── app/
│   │   ├── controllers/    # API route handlers
│   │   ├── models.py       # Database models
│   │   ├── config.py       # Configuration settings
│   │   ├── celery_tasks.py # Celery tasks
│   │   ├── decorators.py   # Request/response decorators
│   │   ├── extensions.py   # Utility functions
│   │   └── __init__.py     # App factory
│   ├── requirements.txt    # Python dependencies
│   ├── app.py              # Main application entry point
│   └── celery_worker.py    # Celery worker setup
├── frontend/               # Vue.js frontend
│   ├── src/
│   │   ├── components/     # Vue components
│   │   ├── views/          # Page components
│   │   ├── router/         # Vue Router setup
│   │   └── stores/         # Pinia stores
│   └── package.json        # Node.js dependencies
└── README.md               # Project documentation
```

5. API Resource Endpoints

The backend exposes the following RESTful API endpoints, protected by role-based JWT authentication.

Authentication (/api/auth)

Method	Endpoint	Description
POST	/login	Authenticates a user (student or admin) and returns a JWT.

Admin Endpoints (/api/admin)

Method	Endpoint	Description
GET	/dashboard	Fetches aggregated data for the admin dashboard.
GET, DELETE	/students	Gets a list of all students or deletes a specific student.
GET, POST	/subjects	Gets a list of all subjects or creates a new one.
PUT, DELETE	/subjects/<id>	Updates or deletes a specific subject.
GET, POST	/subjects/<id>/chapters	Gets all chapters for a subject or creates a new one.
PUT, DELETE	/chapters/<id>	Updates or deletes a specific chapter.
GET, POST	/exams	Gets a list of exams or creates a new one.
PUT, DELETE	/exams/<id>	Updates or deletes a specific exam.
PUT	/exams/<id>/publish	Toggles the published status of an exam.
GET, POST	/exams/<id>/questions	Gets all questions for an exam or creates a new one.
PUT, DELETE	/questions/<id>	Updates or deletes a specific question.
POST	/students/<id>/send-report	Triggers a Celery task to send a report to a student.
POST	/students/export	Triggers a Celery task to export all student data.

Student Endpoints (/api/student)

Method	Endpoint	Description
POST	/register	Registers a new student account.
GET	/<id>/dashboard	Fetches data for the student's personal dashboard.
GET, PUT, DELETE	/<id>	Gets, updates, or deletes the student's profile.

GET	/subjects	Gets a list of all subjects available to students.
GET	/chapters	Gets a list of chapters, optionally filtered by subject.
GET	/exams	Gets a list of all published exams.
POST	/<id>/exam/<id>/start	Starts a new quiz attempt for the student.
POST	/<id>/attempt/<id>/submit	Submits the student's answers for a quiz attempt.
GET	/<id>/attempt/<id>/results	Fetches the detailed results for a specific quiz attempt.
GET	/<id>/history	Gets the student's complete quiz attempt history.
POST	/<id>/history/export	Triggers a Celery task to email the student's history as a CSV.