Project Report: Quiz Master - V2

Author

- **Name**: Vishal Sharma
- **Roll Number**: 23f2004341
- **Student Email**: 23f2004341@ds.study.iitm.ac.in

Self Introduction

I am a full-time student in this program, based in UP, India. This is my first experience building and coding a web-based application. I found the process both challenging and rewarding, especially as I navigated the complexities of backend logic, database integration, and front-end design from scratch. The journey taught me the importance of structured planning, debugging, and leveraging community resources to overcome obstacles.

Description

According to my understanding, the aim of this project was to develop an educational web application, Quiz Master - V2, that allows users to take quizzes on various subjects and tracks their progress while providing administrators with tools to manage content (subjects, chapters, quizzes, and questions) and monitor user performance. The application includes features for user authentication, quiz-taking with a countdown timer, graphical representations of progress, scheduled jobs for reminders and reports, and comprehensive caching for performance optimization, making it a comprehensive learning management tool. The project evolved from a basic quiz system to include advanced admin functionalities, visual analytics, and automated backend processes, reflecting my growth in understanding web development concepts.

Technologies Used

Technology	Purpose
Python	Backend development to handle business logic, route management, and database interactions. Its simplicity and extensive libraries made it ideal for a beginner project.
Flask	Framework for linking backend Python code to front-end Vue.js application. Provides RESTful API endpoints and handles HTTP requests/responses.
Flask-SQLAlchemy	Facilitated the integration of SQLite with the Flask app, allowing easy creation of database tables, relationships, and queries.
SQLite	Chosen as the database to store and retrieve data (users, subjects, quizzes, etc.), providing a lightweight solution suitable for this project.
Vue.js	Frontend framework for building interactive user interfaces with component-based architecture and reactive data binding.
Chart.js	Used to create interactive charts (e.g., user performance, score progression, analytics), providing visual insights into quiz results.
Bootstrap 5	Helped improve the app's aesthetics and provided a quick solution for styling instead of writing extensive CSS code.
Redis	Used for caching frequently accessed data and as a message broker for Celery tasks, improving application performance.
Celery	Handles asynchronous and scheduled tasks like CSV exports, daily reminders, and monthly reports.
Axios	HTTP client for making API calls from Vue.js frontend to Flask backend.

Development Process

The development of Quiz Master - V2 followed an iterative approach:

1. **Planning**: Outlined the database schema and defined user and admin roles based on the project requirements, including scheduled jobs and caching strategies.
2. **Backend Development**: Implemented the Flask API with SQLAlchemy models, starting with authentication and quiz logic, then added Celery tasks, Redis caching, and comprehensive analytics.
3. **Frontend Development**: Created Vue.js components for user and admin interfaces, integrating Bootstrap for styling and Chart.js for visualizations.
4. **Database Integration**: Set up SQLAlchemy models with proper relationships and implemented programmatic database creation with predefined admin user.
5. **Scheduled Jobs**: Implemented Celery tasks for daily reminders, monthly reports, and CSV exports with proper error handling and email notifications.
6. **Caching Implementation**: Added Redis caching for performance optimization with cache invalidation strategies.
7. **Testing**: Tested individual features (e.g., quiz timer, charts, CSV export) with real data, identifying and fixing issues like circular imports and data type mismatches.
8. **Refinement**: Incorporated feedback from debugging sessions, adding proper error handling, input validation, and performance optimizations.

Challenges Faced

1. **Circular Import Issues**: Complex dependencies between Flask app, Celery tasks, and database models required refactoring into separate modules (`celery_app.py`, `extensions.py`) to avoid circular imports.
2. **Database Schema Alignment**: Ensuring frontend field names matched backend model attributes (e.g., `question_statement` vs `question_text`, `option1-4` vs `option_a-d`) required careful coordination.
3. **Celery Task Context**: Implementing Flask app context within Celery tasks was challenging, requiring factory pattern implementation to ensure proper database access.
4. **Quiz Scoring Logic**: Type mismatches between string user inputs and integer model fields required proper type conversion for accurate scoring.
5. **Chart Data Mapping**: Frontend chart components needed to match backend API response structures, requiring consistent data format across endpoints.
6. **Redis Configuration**: Setting up Redis for both caching and Celery broker/backend required proper configuration and error handling.
7. **Email Configuration**: Gmail SMTP setup with app passwords and environment variables required careful configuration for scheduled email tasks.
8. **Frontend-Backend Synchronization**: Ensuring Vue.js components correctly consumed Flask API endpoints with proper error handling and loading states.

DB Schema Design : https://drive.google.com/file/d/1UNaLiRAiGVtO7easnW_JYNEmrZx0C73B/view?usp=sharing

The database schema for Quiz Master - V2 is designed to support the relationships between users, subjects, chapters, quizzes, questions, and progress tracking. The structure is as follows:

One-to-Many Relationships:

- Each **Subject** can have multiple **Chapters** (e.g., "Mathematics" has "Algebra", "Calculus").
- Each **Chapter** can have multiple **Quizzes** (e.g., "Algebra" has "Quiz 1", "Quiz 2").
- Each **Quiz** can have multiple **Questions** (e.g., "Quiz 1" has multiple-choice questions).
- Each **User** can have multiple **Score** entries (e.g., one per completed quiz).

Tables:

- **User**: Stores user details (id, email, password_hash, full_name, qualification, dob, role, created_at).
- **Subject**: Stores subjects (id, name, description, created_at).
- **Chapter**: Stores chapters with a foreign key to subject (id, name, subject_id, description, created_at).
- **Quiz**: Stores quizzes with a foreign key to chapter (id, title, chapter_id, date_of_quiz, duration, remarks, created_at).
- **Question**: Stores questions with a foreign key to quiz (id, quiz_id, question_statement, option1, option2, option3, option4, correct_option, created_at).
- **Score**: Tracks user quiz scores (id, user_id, quiz_id, total_score, timestamp).

Design Notes:

- **Role-based Access**: Users have 'user' or 'admin' roles for proper access control.
- **Timestamps**: All tables include created_at timestamps for audit trails.
- **Foreign Keys**: Proper relationships maintained with cascade delete options.
- **Data Types**: Appropriate data types for each field (e.g., INTEGER for scores, TEXT for questions).
- **Constraints**: NOT NULL constraints on required fields and UNIQUE constraints where appropriate.

Architecture and Features

Backend Code Structure:

- `app.py`: Main Flask application entry point with factory pattern
- `api.py`: RESTful API endpoints for all CRUD operations
- `models.py`: SQLAlchemy database models and relationships
- `celery_app.py`: Celery tasks for scheduled jobs and async operations
- `extensions.py`: Shared Flask extensions and utility functions
- `config.py`: Application configuration and environment settings
- `database.py`: Database initialization and admin user creation

Frontend Structure:

- **Vue.js Components**: Modular component-based architecture
- **Router**: Vue Router for navigation between user and admin sections
- **API Integration**: Axios for HTTP requests to Flask backend
- **Charts**: Chart.js integration for data visualization
- **Styling**: Bootstrap 5 for responsive design

Static Files:

- CSS and JavaScript libraries (Bootstrap, Chart.js) are managed via npm packages
- Custom styling with gradient backgrounds and modern UI elements

Features Implemented

User Features:

- **Authentication**: Register and login with email and password
- **Dashboard**: View available quizzes with subject and chapter information
- **Quiz Attempt**: Take quizzes with countdown timer and automatic submission
- **Progress Tracking**: View performance statistics and score history
- **Analytics**: Interactive charts showing performance by subject and score trends
- **CSV Export**: Download quiz history as CSV file
- **Recent Activity**: View last 5 quiz attempts with detailed information

Admin Features:

- **User Management**: View, edit, and delete user accounts
- **Subject Management**: Create, edit, and delete subjects
- **Chapter Management**: Create, edit, and delete chapters under subjects
- **Quiz Management**: Create, edit, and delete quizzes with date and duration settings
- **Question Management**: Add, edit, and delete multiple-choice questions
- **Analytics Dashboard**: Comprehensive charts showing user performance and system statistics
- **Search Functionality**: Search across users, subjects, chapters, quizzes, and questions
- **CSV Export**: Export all users' statistics for analysis
- **Cache Management**: View cache statistics and clear cache when needed

Backend Jobs (Celery Tasks):

- **Daily Reminders**: Automated email reminders to inactive users
- **Monthly Reports**: Generate and email comprehensive activity reports
- **CSV Export**: Asynchronous quiz history export with email notification
- **Email Testing**: Test email functionality for debugging

Performance Features:

- **Redis Caching**: Cache frequently accessed data (subjects, chapters, analytics)
- **Cache Invalidation**: Automatic cache clearing when data is modified
- **Optimized Queries**: Efficient database queries with proper indexing
- **Background Tasks**: Non-blocking operations for better user experience

Technical Highlights

Security Features:

- Password hashing with Flask-Bcrypt
- Role-based access control
- Session management with Flask-Login
- Input validation and sanitization

Performance Optimizations:

- Redis caching for database queries
- Celery for background task processing
- Efficient database relationships
- Responsive frontend design

Scalability Considerations:

- Modular code structure
- Separation of concerns
- Configurable environment settings
- Extensible API design

Conclusion

The Quiz Master - V2 project represents a comprehensive learning management system that successfully integrates modern web technologies. The application demonstrates proficiency in full-stack development, database design, and system architecture. The implementation of scheduled jobs, caching strategies, and real-time analytics showcases advanced backend development skills, while the responsive Vue.js frontend provides an excellent user experience.

The project successfully addresses all requirements from the project statement, including user authentication, quiz management, analytics, scheduled jobs, and performance optimization. The iterative development process and problem-solving approach demonstrate strong software engineering practices and the ability to overcome technical challenges.

This project has been an invaluable learning experience, providing hands-on experience with modern web development technologies and best practices. The skills acquired through this project will be beneficial for future software development endeavors.

Video_link: https://drive.google.com/file/d/1iSJTE3PQ4PKebMJJHIn5xPtnZ92lhAkl/view?usp=sharing