# Project Report: Quiz Master – V2

**Name:** Amirtha Varshini Elavarasu
**Roll No.:** 24dp1000036
**Student Email:** 24dp1000036@ds.study.iitm.ac.in

## About Myself

I am in the final term of my programming diploma, building a strong foundation in web development. With a Bachelor of Commerce (B.Com), I have transitioned into the tech industry to expand my technical expertise and career opportunities.

## Project Description

This project is a web application where students can take quizzes by subject and chapter, and view their scores. Admins manage quiz content and user data. The app includes daily G-Chat reminders for inactive users or new quizzes, monthly HTML reports via email, and CSV export of quiz data for admin analysis.

## Technologies Used

### Backend Technologies
• Python - The primary language for backend development.
• Flask – A lightweight web framework used to build the application.

### Flask Extensions & Libraries
• Flask-SQLAlchemy – ORM for database interaction.
• Flask-Migrate – Handles database migrations via Alembic.
• Flask-Caching – Enhances API performance via caching.
• Flask-RESTful – Simplifies REST API creation.
• Flask-Security – Manages user auth, roles, and sessions.
• Flask-CORS – Handles Cross-Origin Resource Sharing.

### Database Management
• SQLAlchemy – ORM for defining models and executing queries.
• joinedload – Used for efficient data retrieval.
• SQLite – Development and testing database.

### Asynchronous Tasks (Celery)
• Celery – Schedules background jobs (reminders, reports, exports).
• @task, @shared_task – Defines background jobs.
• AsyncResult – Monitors task state.
• crontab – Schedules recurring tasks (daily, monthly).

### Redis
• Used as message broker and result backend.
• Broker URL: "redis://localhost:6379/0"
• Result Backend: "redis://localhost:6379/1"

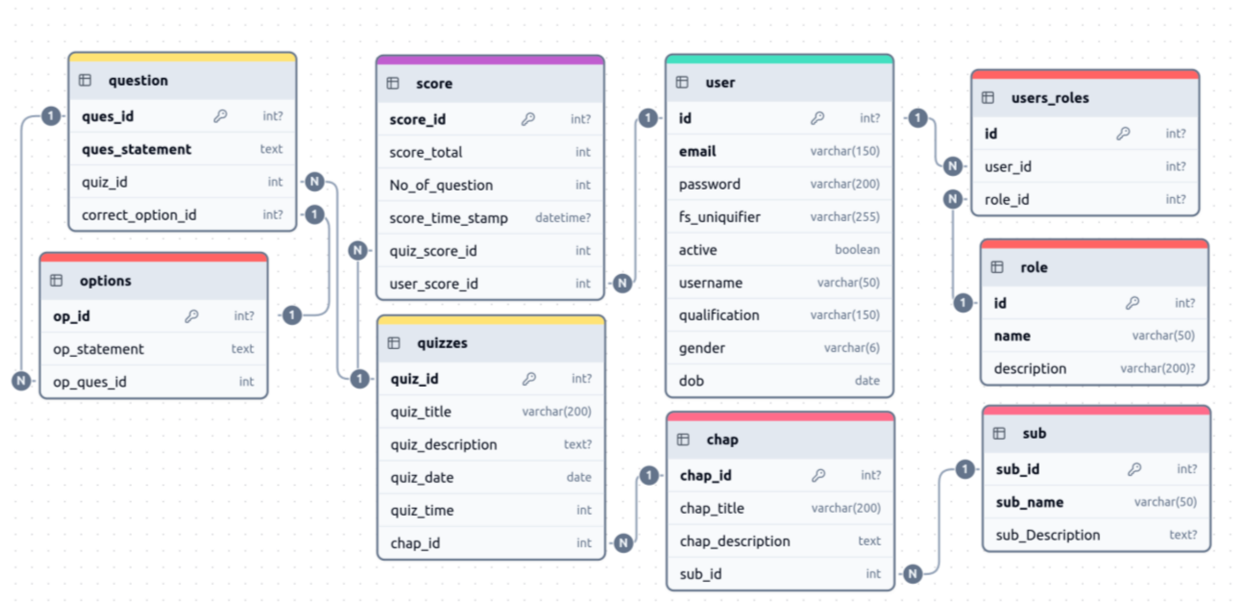
### Email & Notification System
• smtplib, email.mime modules – For sending HTML and plain-text emails.
• MailHog – Testing email locally.
• Google Chat Webhooks – Sends G-Chat reminders and updates.

### Security & Authentication
• Flask-Security – Role-based access using decorators.
• hash_password, UserMixin, RoleMixin – Auth helpers.

### Utility Libraries
• datetime, calendar – For quiz scheduling and reports.
• collections, random – Report generation and styling.
• logging, jinja2.Template – Logging and email rendering.
• os – File handling and environment variables.

### Frontend Technologies
• Vue CLI – Project setup, hot reload, and modularity.
• Vue 3 – Builds reactive user interfaces.
• Vue Router – Manages component routing and navigation.
• Axios – Communicates with backend APIs.
• Chart.js – Visualizes performance data.
• Vue Components – Handles UI sections (Navbar, Quiz, Admin, etc.)

## DB Schema Design



• Follows 3rd Normal Form (3NF) to eliminate redundancy.
• Scalable schema for subjects, quizzes, and users.
• Admin and User roles separated for security.
• Efficient querying via foreign keys and ORM mappings.

## API Design

Provides secure, role-based access to quiz features, scoring, user data, and export functionality.
• /api/admin_dashboard – Admin view of Subjects, Chapters, Quizzes.
• /api/user_dashboard – User profile and quizzes list.
• /api/registration – Register new user.
• /api/login – Login existing user and return auth token.
• /api/logout – Logs out current user.
• /api/subject – CRUD for Subjects (Admin).
• /api/chapters – CRUD for Chapters (Admin).
• /api/quizzes – CRUD for Quizzes (Admin).
• /api/questions – CRUD for Questions (Admin).
• /api/search – Role-based search (Users, Quizzes, Chapters, Scores).
• /api/quizview/:sub_id/:chap_id – Subject-Chapter quiz view.
• /api/questions_page/:quiz_id – Returns questions for quiz.
• /api/quiz_submission/:quiz_id – Submits answers and returns score.
• /api/score_page/:user_id – Returns user's score history.
• /api/summary_page – Analytics summary for Admin and User.
• /api/export_csv – Admin CSV export trigger.
• /api/csv_result/:id – Admin download of CSV file by job ID.

## Architecture and Features

Follows MVC (Model-View-Controller) architecture:
• Model – backend/app/models/ – SQLAlchemy ORM.
• View – frontend/src/components/ – VueJS SPA components.
• Controller – backend/app/routes/ – Flask route handlers.

## Celery Usage

• Daily Reminder – G-Chat messages to inactive users or new quiz alerts.
• Monthly Report – Email quiz summary as HTML/PDF.
• User CSV Export – Background generation and download of quiz attempts.

## Behavior Flow & SPA Integration

Frontend (Vue) triggers API → Flask Controller handles → interacts with Models → returns JSON → Vue renders result.
Backend and frontend are fully decoupled. Role-based access managed by Flask-Security. Redis used for caching.

## Project Demo Link

https://drive.google.com/file/d/1FuhFZyLouu9Z43Kdko8DgVdz6Uar7s3c/view?usp=sharing