Quiz Master -V2

Project Report

Name: Sundram Kumar Ray
Roll Number: 23f2002880
Email: 23f2002880@ds.study.iitm.ac.in

Project Details: Project Statement

It is a multi-user app (one requires an administrator and other users) that acts as an exam preparation site for multiple courses.

1.Vue CLI Frontend Implementation

The frontend is built using **Vue CLI 3+**, enabling component-based architecture and routing with vue-router.

Routing

I used vue-router to manage navigation between different views:

```
const routes = [
  { path: '/login', component: LoginPage },
  { path: '/dashboard', component: UserDashboard },
  { path: '/admin', component: AdminDashboard },
  { path: '/quiz/:id', component: QuizPage },
  { path: '*', component: NotFoundPage }
]
```

Routes are dynamically guarded based on user roles and authentication tokens.

Components

Each screen is broken into reusable Vue components, e.g., AdminDashboard.vue, AdminQuizzes.vue, UserSummary.vue. These components communicate via props and Vuex store when necessary.

2.API Communication (Frontend to Backend)

APIs are consumed using **Axios** via a centralized service. Each component makes requests like:

```
axios.get('/api/user/quizzes')
```

Example APIs:

- POST /api/login/user → User login
- GET /api/user/quizzes → List of quizzes for the user
- GET /api/user/export-status/<task_id> → Poll export job status
- GET /api/subjects → List all subjects (cached with Redis)

All protected routes use Authorization: Bearer <JWT_TOKEN> headers.

3.JWT Authentication

I used Flask-JWT-Extended on the backend. Upon login, a JWT token is issued and stored in the frontend (usually in localStorage or memory).
The token includes identity and custom claims:

```
{
  "sub": "admin",
  "role": "admin",
  "exp": 1715550000
}
```

This token is used to protect API routes via @jwt_required() decorators in Flask.

4.Redis Caching

I used **Redis (DB 3)** for caching heavy-read APIs like subjects or chapters.

Example:

```
@cache.cached(timeout=300, key_prefix='all_subjects')
def get_subjects():
    return Subject.query.all()
```

Cache Invalidation

On POST/PUT/DELETE, the cache is invalidated manually:

```
cache.delete(all_subjects')
```

I monitored Redis via redis-cli monitor and confirmed keys like flask_cache_all_subjects are set.

5.Background Jobs (Celery)

I used **Celery with Redis broker** to manage asynchronous and scheduled jobs.

a. Daily Reminder Job

Function: Sends reminders to users via email if:

- They haven't attempted available quizzes.
- New quizzes are created.

```
@celApp.task
def send_unattempted_quiz_reminders():
    for user in users:
        # Check unattempted quizzes
        # Compose email
        send_email(sender, user.email, subject, message)
```

Scheduled daily via:

```
@celApp.on_after_configure.connect
def setup_periodic_tasks(sender, **kwargs):
```

```
    sender.add_periodic_task(crontab(hour=18, minute=30), send_unattempted_quiz_reminders.s()
)
```

b. Monthly Activity Report (HTML Email)

Function: Sends HTML report of all attempted quizzes at the end of each month.
Details include:

- Quiz ID, subject → chapter, date
- Score, average, status per question

HTML is generated dynamically and saved temporarily. The email is sent using smtplib:

```
send_email(sender, user.email, subject, html_path)
```

Scheduled monthly via:

```
sender.add_periodic_task(crontab(day_of_month=1, hour=10), send_monthly_summary.s())
```

c User-Triggered Async Job - Export Quiz Data as CSV

A user can trigger export via frontend (button click). This hits:

```
POST /api/user/export-csv
```

Which runs this Celery task:

```
@celApp.task
def export_user_csv(user_id):
    # Fetch user quiz attempts
    # Write to CSV
    return csv_path  # downloaded later
```

The frontend polls:

```
GET /api/user/export-status/<task_id>
```

Once status: SUCCESS, a download link is enabled.

```
<a :href="csvDownloadLink">Download CSV</a>
```

Files are served via:

```
send_from_directory("static/exports", filename, as_attachment=True)
```

Summary

This documentation outlines key pieces of the quiz application:

- Vue CLI SPA with Vue Router and Axios for API communication
- JWT for auth

- Redis for caching
- Celery for daily reminders, monthly summary, and CSV export
- All the other functionalities are same as Quiz Master V1

This stack makes the app scalable, responsive, and efficient for both users and administrators.

Video Link :
https://drive.google.com/file/d/1uEOs6gtkfuQkAdQdHOpIUDpAXXmMqVR5/view?usp=sharing

YAML file for API definition :

{Security: All routes that require login are protected with bearerAuth (JWT).

File Split: The above YAML can be split or modularized by tags (User, Admin).}

https://github.com/23f2002880/quiz_master_v2_23f2002880/blob/main/openapi_quizmaster.yaml

Approximate AI Percentage usage

Component / Tool	Purpose / Role in Project	MAD-2 Usage %	Notes
Flask (App + API)	Backend logic, routing, request handling	10%	In MAD-2, Flask is mainly used to serve REST APIs
SQLite + SQLAlchemy	Data modeling, relationships, persistence	5%	Mostly similar usage; MAD-2 may have fewer direct queries due to APIs
HTML + Jinja2	Server-side template rendering	0%	Heavy in MAD-1; replaced by Vue in MAD-2
Bootstrap / CSS	Styling and layout	6%	UI libraries used in both, more flexible in MAD-2
JavaScript / AJAX (vanilla)	Interactivity, client-side logic	0%	More in MAD-1 due to Jinja forms; Vue replaces this in MAD-2
Authentication (Login/Auth)	Login/logout, role-based control	3%	May shift from Flask-Login (MAD-1) to token-based (MAD-2)
Admin CRUD Logic	Manage users/quizzes/services, etc.	4%	More frontend-driven in MAD-2
User Logic (Quiz/Booking)	Core user flow: attempts, bookings, usage	5%	Built in Vue + API in MAD-2
Validation (Client + Server)	Form and input validation	1%	JS/Flask-WTF/etc. in MAD-1, Vue+backend in MAD-2

Charts / Data Visualization	Display results/scores/analytics	1%	Optional in both
Testing / Debugging	Ensuring correctness and robustness	0%	Rarely automated; more manual
APIs / External Integration	Fetching or posting data to external/internal APIs	4%	Emphasis increases in MAD-2
Vue.js (Core + Router)	Reactive frontend, routing, page rendering	20%	Vue is central in MAD-2, not used in MAD-1
Axios/fetch/etc. (API Comm.)	Handling async API calls from frontend	1%	Replaces traditional form POSTs in MAD-2
Vuex / Pinia (State Management)	Global state handling in frontend	0%	Optional; used in larger MAD-2 apps