COLLEGE CODE : 9623

COLLEGE NAME : Amrita College of Engineering And Technology

DEPARTMENT : Computer Science and Engineering

STUDENT NM-ID:

ROLL NO : 23CS042

DATE : 12-09-2025

Completed the project named as

Phase 2 Solution design and architecture

PROJECT NAME: INTERACTIVE QUIZ APP

SUBMITTED BY,

NAME: James Deroses Judwin.J
MOBILE NO:7358599082

Interactive Quiz App

Phase 2 - Solution Design & Architecture

This phase involves designing the technical foundation and architecture of the Interactive Quiz App. It ensures that the application is scalable, secure, user-friendly, and capable of handling real-time quiz interactions.

Section 1: Tech Stack Selection

Frontend: React.js (for fast, responsive

UI), Tailwind CSS (for styling).

Backend: Node.js with Express (for REST APIs).

Database: MongoDB (flexible document storage for quiz questions, users, scores).

Authentication: JWT-based authentication.

Hosting: Vercel (frontend), AWS/Heroku (backend), MongoDB Atlas (database).

Reasoning: This stack provides speed, scalability, and ease of integration.

Section 2: UI Structure Design

Home Screen: Login/Signup options, available quizzes list.

Quiz Screen: Question display, options, timer, next button.

Result Screen: Score summary, correct/incorrect answers, retry option.

Leaderboard Screen: Global & friend-based rankings.

Admin Panel: Add/edit/delete questions, manage users, track quiz stats.

Wireframes ensure the app is intuitive, reducing user learning curve.

Section 3: API Schema Design

Example API Endpoints:

POST /auth/signup → Register new users.

POST /auth/login → Authenticate user, return JWT.

GET /quiz/:id → Fetch quiz details & questions.

POST /quiz/:id/submit → Submit answers & calculate score.

GET /leaderboard → Get top scores.

Schema Design:

User Schema: { username, email, password, scores }

Quiz Schema: { title, category, questions[], timer }

Score Schema: { userId, quizId, score, timeTaken }

Section 4: Data Handling Approach

Store quiz questions in MongoDB collections.

Use indexes for faster search queries.

Caching with Redis for frequently accessed data (leaderboards).

Backup strategy: Automated backups every 24 hours.

Data validation: Ensure each quiz has a fixed structure.

Section 5: Component / Module Diagram

Modules include:

1. Authentication Module – signup, login, JWT verification.

2. Quiz Engine Module – handles quiz flow, question navigation, scoring.

3. Leaderboard Module – ranking, score tracking.

4. User Profile Module – history, achievements.

5. Admin Module – quiz creation, monitoring.

Diagram:

User → Frontend (React) → Backend (Node.js/Express) → Database (MongoDB)

Section 6: Basic Flow Diagram

1. User logs in.

2. User selects a quiz.

3. App fetches quiz questions from backend.

4. User answers questions → Backend validates & stores responses.

5. Backend calculates score & updates

leaderboard.

6. Result displayed on UI.

Section 7: Scalability Considerations

Horizontal scaling with multiple Node.js instances.

Load balancer to handle high traffic.

Database sharding for very large question banks.

Serverless functions for lightweight

operations.

Section 8: Security Measures

JWT-based authentication with expiry.

Password hashing with bcrypt.

HTTPS for secure communication.

Input sanitization (prevent SQL injection, XSS).

Rate limiting (prevent brute-force attacks).

Section 9: Testing Strategy

Unit Testing: Test each module (quiz engine, scoring logic).

Integration Testing: Test API endpoints with Postman.

UI Testing: Ensure quizzes load properly across devices.

User Acceptance Testing (UAT): Pilot with small group.

Automated Testing Frameworks: Jest (backend), Cypress (frontend).

Section 10: Deployment Plan

CI/CD pipeline with GitHub Actions.

Frontend Deployment: Vercel.

Backend Deployment: AWS/Heroku.

Database Hosting: MongoDB Atlas.

Monitoring: LogRocket for frontend, AWS CloudWatch for backend.

Section 11: Future Enhancements

Real-time multiplayer quizzes.

AI-based question recommendation.

Gamification (badges, achievements).

Offline mode with local caching.
