9530

St. MOTHER THERESA ENGINEERING COLLEGE

COMPUTER SCIENCE ENGINEERING

NM-ID: 2ECA55BDB9A57A9818941CABBEFBDEE2

REG NO: 953023104087

DATE:15-09-2025

Completed the project named as

Phase 2

FRONT END TECHNOLOGY

INTERACTIVE QUIZ APP

SUBMITTED BY,

PON APARNA M

9042688167

Phase 2 — Solution Design & Architecture

1. Tech Stack Selection :

The selection of technologies is critical for scalability, performance, and security.

❖ **Frontend**

- React.js (component-based, responsive UI)
- Libraries: React Router, Axios, Redux/Context API

❖ **Backend**

- Node.js with Express.js (efficient REST API handling)

❖ **Database**

- MongoDB (flexible JSON-like schema for quizzes & users)

❖ **Security**

- Authentication: JWT for secure login/session handling
- Password Security: bcrypt.js for hashing
- Middleware: Helmet & CORS for security

❖ **Hosting**

- Frontend Hosting: Vercel/Netlify
- Backend Hosting: Render/Heroku
- Database Hosting: MongoDB Atlas

2. UI Structure & API Schema Design :

❖ **UI Structure**

- **Home Page**: App intro, login/signup
- **Quiz Dashboard**: Displays quizzes & leaderboard preview
- **Quiz Screen**: Multiple-choice questions, timer, navigation

- **Result Page**: Score, correct/incorrect answers, leaderboard

❖ **API Schema (Sample Endpoints)**

Endpoint	**Method**	**Description**
/api/auth/signup	POST	Register a new user
/api/auth/login	POST	Authenticate user
/api/quizzes	GET	Fetch list of quizzes
/api/quiz/:id	GET	Fetch quiz questions by ID
/api/quiz/:id/submit	POST	Submit answers & calculate score
/api/leaderboard	GET	Display top scorers

3. Data Handling Approach :

Efficient data handling ensures consistency, accuracy, and real-time updates.

Quiz Schema Example :

```
{
  "quizId": "1",
  "title": "JavaScript Basics",
  "questions": [
    {
      "questionText": "Which keyword is used to declare variables in
JS?",
      "options": ["var","let","const","All of the above"],
      "correctAnswer": "All of the above"
    }
  ]
}
```

❖ **Data Flow:**

- User selects quiz → API fetches questions
- User submits answers → Backend checks correctness
- Score calculated → Stored in DB
- Leaderboard updated → Fetched on request

4. Component / Module Diagram :

❖ **Frontend Modules**

- **AuthComponent** → Handles login/signup
- **QuizComponent** → Renders questions
- **TimerComponent** → Manages countdown
- **ResultComponent** → Shows results & leaderboard

❖ **Backend Modules**

- **AuthController** → Authentication logic
- **QuizController** → Quiz fetch & submit
- **ResultController** → Score calculation
- **LeaderboardController** → Ranking logic

5. Basic Flow Diagram :

[User] → [Login/Signup] → [Select Quiz] → [Answer Questions] → [Submit Quiz] → [Score Calculation] → [Show Result & Leaderboard]

6. Extended Design Notes :

- ❖ **Error Handling**
  - Handle incomplete answers with proper messages
  - Retry mechanism for database connection failures
- ❖ **Scalability Considerations**
  - Cache frequently accessed quizzes
  - Paginate leaderboard data
- ❖ **Future Enhancements**
  - Real-time multiplayer quizzes
  - AI-driven question generation
  - Gamification: badges, achievements