Aryan Mahida – 92200133011 **Jay Mangukiya – 92200133040**

Documentation and Reporting

1. Technical Report

1.1 System architecture overview:

This section will summarize the project's technical aspects including system architecture and tech stack.

The QuizUP system follows a layered modular architecture with clear separation of concerns, enabling independent development, testing, and deployment of components. The architecture is structured into five primary layers:

- **Presentation Layer**: React Frontend, mobile application
- **Application Layer**: REST API server (expressJS, port 3000) and Match server (socket.io, port 3001)
- **Data Layer**: PostgreSQL (Port 5432)
- **Infrastructure Layer**: Docker containers, prometheus monitoring

1.2 Core modules:

1.2.1 Frontend module (React + Typescript):

- **Purpose**: User interface and experience management.
- **Structure**: Components, services (websocket clients), hooks, contexts, types, utils.
- **Features**: Quiz builder, real time gameplay, question bank, login/signup, quiz interface, leaderboard.
- **Benefits**: Component reusability due to react single page application, reduced duplication, easy role extension.
- **Command**: npm run dev -- --host

1.2.2 API/Backend server (express + sequelize):

- **Purpose**: Business logic, data management and easier data handling.
- **Endpoints**: Authentication, categories, quizzes, questions, matches, users, leaderboard etc.
- **Benefits**: Clear layering (Controller -> Service -> Model), service reuse, easy extensibility.
- **Command**: npm run dev

1.2.3 Match server (websocket + socket.io):

- **Purpose**: Real time match orchestration.
- **Core classes**: MatchEngine, PlayerManager, QuestionManager, ScoreManager, AIEngine.
- **Event flow**: authenticate -> create/join -> player_ready -> match_started -> submit_answer -> results.
- **Benefits**: Real time, supports multiple modes (friend, AI, tournament), easily extendable.
- **Command**: npm run dev:match

1.2.4 Database (PostgreSQL + sequelize)

The platform uses PostgreSQL with sequelize ORM to manage core entities like users, quizzes, questions, matches, and attempts. Sequelize provides object oriented concepts such as abstraction, reuse so it is easier to model the database. Performance is optimized through indexes on ELO ratings (for leaderboard scoring), match statuses, quiz question order, category hierarchies etc.

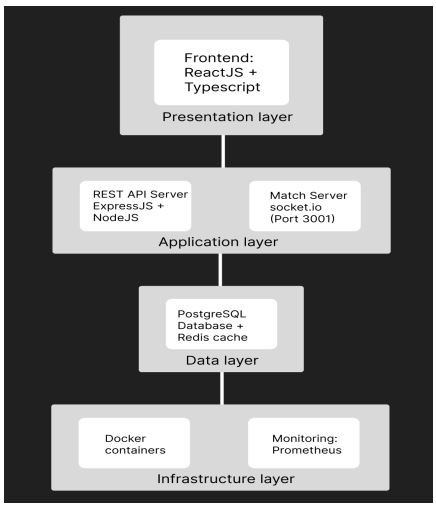
1.3 Module interaction:

The flow of the different modules will be like this

Data Flow: Frontend actions pass through the API to the database, while real time match data flows through websocket connections to the match server.

- **REST:** Frontend - API (with JWT auth)
- **Websocket:** Frontend - Match Server
- **ORM:** API - Database

Overall system architecture:



2. User manual:

2.1 System requirements:

- **Operating system**: Windows, macOS, Linux
- **Node.js**: Version 16 or higher
- **PostgreSQL**: Version 12 or higher
- **Docker**: Version 20.10+ (for Docker deployment)

2.2 Installation methods:

Method 1: Local development setup

Step 1: Clone and setup

```
# clone the repository
git clone < https://github.com/Jaymangukiya22/Capstone-Project>
cd Capstone-Project

# install backend dependencies
cd backend
npm install

# install frontend dependencies
cd ../Frontend-admin
npm install
```

Step 2: Database

1. Install and start PostgreSQL
2. Create a new database named quizup_db
3. Copy .env.example to .env in the backend folder
4. Update database connection details:

```
DATABASE_URL="postgresql://username:password@localhost:5432/quizup_db"
JWT_SECRET="***"
REDIS_URL="redis://localhost:6379"
```

Step 3: Start the services

```
# terminal 1 - Start backend (Port 3000)
cd backend
npm run dev
```

```
# terminal 2 - Start match server (Port 3001)
cd backend
npm run dev:match

# terminal 3 - Start frontend (Port 5173)
cd Frontend-admin
npm run dev -- --host
```

Method 2: Docker deployment

Step 1: Environment Setup

```
# copy environment file
cp .env.example .env

# update .env with your settings
APP_ENV=development
```

Step 2: Build and Run

```
# build and start all services
docker-compose up --build

# run in background
docker-compose up -d –build
```

2.3 Accessing the application

Web Interface

- **Admin side**: http://localhost:5173
- **API backend**: http://localhost:3000
- **Match server**: ws://localhost:3001

2.4 Login credentials

Create new account and sign in using those credentials.

3. Code documentation:

The codebase consists of structure like this, divided into frontend and backend.

```
Frontend-admin/
├── src/
│   ├── components/                    # Reusable UI components
│   │   ├── ui/                        # shadcn/ui base components
│   │   ├── admin/                     # Admin dashboard components
│   │   ├── student/                   # Student interface components
│   │   └── shared/                    # Cross role shared components
│   ├── services/                      # API and websocket clients
│   ├── hooks/                         # React hooks
│   ├── contexts/                      #React contexts
│   ├── types/                         # Typescript definitions
│   └── utils/                         # Helper functions
Backend/src/
├── controllers/                       # HTTP request handlers
├── services/                          # Entire logic code
├── models/                            # Sequelize ORM models
├── routes/                            # API route definitions
├── middleware/                        # Auth, validation, logging etc.
├── utils/                             # Helper utilities
└── config/                            # Database & app configuration
```

One of the code snippets:

apiService.ts

```
/**
 * Apiservice - HTTP wrapper for backend API calls
 * Handles auth headers and error responses
 */
export class ApiService {
  async get<T>(endpoint: string): Promise<T> {
    // Perform authenticated GET request
  }
}
```

The codebase follows standard documentation practices (JSDoc for Typescript, inline comments for functions/classes), making it maintainable for future scalability and reuse.