Online Quiz Application with AI-Powered Quiz Generator

A PROJECT REPORT

Submitted by

Roll Number	Registration Number	Student Code	Student Name
23010346209	23013001066	BWU/BCA/23/221	SOUGATA MANNA
23010346216	23013001073	BWU/BCA/23/227	SOYAL RANA MANDAL
23010346198	23013001055	BWU/BCA/23/209	IRIS CHATTERJEE
23010346179	23013001036	BWU/BCA/23/190	APURBA KUITI

in partial fulfillment for the award of the degree

of

BACHELOR OF COMPUTER APPLICATIONS

IN

COMPUTATIONAL SCIENCES

BRAINWARE UNIVERSITY
398, Ramkrishnapur Road, Barasat, North 24 Parganas, Kolkata - 700 125

Abstract

This project develops an Online Quiz Application using ReactJS and Firebase, with an AI-powered quiz generation feature. The app supports reusable React functional components, state management with hooks, dynamic rendering of quiz data, routing for navigation, and responsive styling. The AI generator can create custom quizzes based on user-provided topics and difficulty. Persistence of quiz results is implemented using local storage (and Firebase for authenticated users).

Additionally, the system includes a user-friendly interface that enhances accessibility and improves the overall quiz-taking experience. The integration of AI helps reduce manual effort in quiz creation and increases the adaptability of assessments for different learning levels. This project demonstrates the practical application of modern web technologies and AI tools in developing interactive educational platforms that are scalable, efficient, and easy to maintain.

TABLE OF CONTENTS

Chapter 1: **Introduction**

Online assessments have become an essential part of **modern education, skill development, and recruitment processes**. With the rapid growth of digital learning platforms, there is an increasing need for interactive, flexible, and user-friendly quiz systems that can evaluate knowledge effectively. This project aims to address that need by developing a dynamic and engaging Online Quiz Application using **ReactJS** and **Firebase**.

The application enables users to explore multiple quiz categories, attempt quizzes based on their interests, and track their performance through detailed result analytics. By integrating **Artificial Intelligence (AI)**, the system goes beyond traditional quiz platforms and allows users to generate custom quizzes instantly based on any topic and difficulty level they choose.

This combination of modern web technologies and AI-powered automation results in a fast, responsive, and intuitive quiz-taking environment. The platform is designed to **support learners, job seekers, and educators** by offering a customizable, interactive, and efficient assessment experience.

Chapter 2: **Objectives**

- Build a React single-page application using create-react-app.
- Use JSX, functional components, and hooks for state management.
- Implement routing (Start, Quiz, and Result pages) using react-router-dom.
- Integrate Firebase for user authentication and optional data persistence.
- Provide AI-powered custom quiz generation and a fully responsive user interface.
- Develop a structured result evaluation module that calculates scores, displays performance analytics, and stores quiz outcomes for future reference.
- Ensure a user-friendly, accessible, and intuitive interface that enhances the quiz-taking experience across devices, including mobile and tablets.

Chapter 3: **Planning**

Project planning is a crucial part of the development lifecycle and ensures that the Online Quiz Application is built efficiently, within the specified timeline, and with clearly defined objectives. Proper planning helps in organizing tasks, selecting appropriate technologies, assigning responsibilities within the team, and identifying potential risks early.

The planning for this project was carried out in several phases, as described below:

3.1: Requirement Gathering & Analysis

The first stage involved understanding the requirements of the system, including quiz features, AI-based quiz generation, authentication needs, UI layout, and result tracking. Functional and non-functional requirements were identified and documented.

3.2: Technology Selection

Based on system needs, the following technologies were finalized:

- ReactJS for interactive UI development
- Firebase for authentication and optional data storage
- CSS for responsive styling
- LocalStorage for quiz result persistence
- AI Integration (Gemini API) for custom quiz generation

3.3: Task Breakdown & Module Allocation

The project was divided into multiple modules such as:

- User Authentication
- Quiz List & Category Display
- Quiz Engine (Question Rendering)
- AI Quiz Generator
- Results Module
- Routing & Navigation
- UI/UX Development

Each team member was assigned specific modules based on their strengths and skills.

3.4: UI/UX Wireframing

Basic layouts for pages such as Home, Login, Quiz, and Results were sketched to visualize the user experience before implementation. This ensured consistency and smooth navigation throughout the app.

3.5: Timeline Planning

A rough timeline was created to manage work:

- Week 1: Requirement Analysis & Technology Setup
- Week 2: UI Design & Component Structure
- Week 3: Quiz Engine & Routing Development
- Week 4: Firebase Authentication Integration
- Week 5: AI Quiz Generator Development
- Week 6: Results Module & Testing
- Week 7: Optimization, Bug Fixing, and Documentation

3.6: Risk Identification & Mitigation

Possible risks were noted, such as:

- API failure during AI quiz generation
- Firebase authentication delays
- Data inconsistency in localStorage
- Frontend responsiveness issues

Backup strategies like mock quiz generation and fallback UI were planned to handle such cases.

3.7: Final Testing & Deployment Planning

The planning phase also included:

- Module testing
- Integration testing
- UI responsiveness testing
- Deployment steps for running the app on hosting platforms (Netlify/Vercel)

Chapter 4: **Folder Structure**

The project follows a **modular and well-organized folder structure**, which helps maintain scalability, readability, and clean separation of concerns. Each folder inside the project serves a specific purpose to support the functioning of the Online Quiz Application.

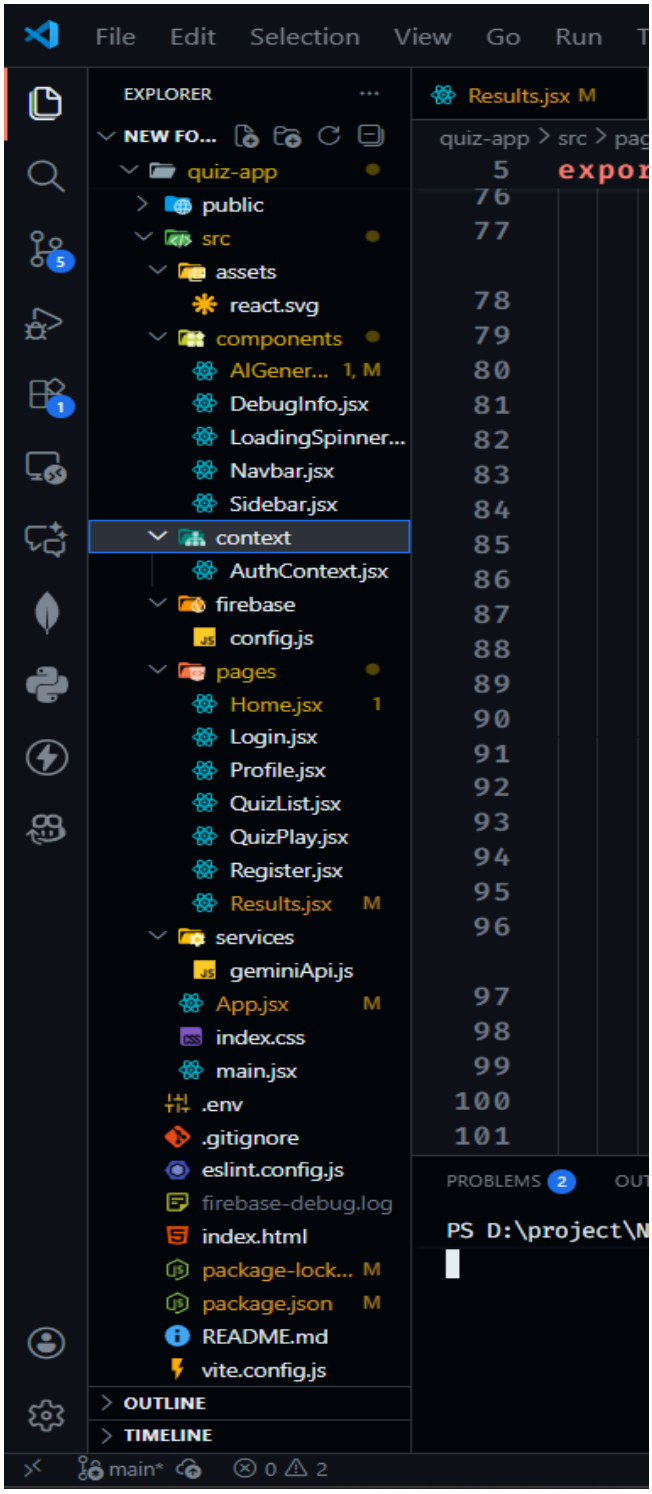

Figure 4.1: Folder Structure Image

Chapter 5: **Front- End**

- **Src/**

All the main logic, UI components, routing, styles, and service files for the application are stored inside this folder.
This is where the entire application is built, compiled, and rendered by React.

The src/ directory contains all source code of the Quiz Application, including React components, pages, routing configuration, context APIs, Firebase integration, and AI quiz services. This folder represents the entire frontend logic of the application and is processed by Vite during both development and production build.

1. **Component: AIGeneratorModal.jsx**

Purpose:
This component is responsible for generating quizzes automatically using AI (Gemini API). It allows the user to enter a topic, select difficulty, choose the number of questions, and then generates a quiz dynamically.

Hooks Used in This Component

✓ useState()
Used to manage component-level states such as:

- **topic** – the subject of the quiz
- **difficulty** – easy, medium, or hard
- **numberOfQuestions** – 3, 5, or 10
- **loading** – loader animation during quiz generation
- **error** – shows validation or API errors

✓ useNavigate()
Used to redirect users to the custom quiz route after generating quiz.

Code:

```
// AI Quiz Generator (Short Code for Documentation)
import React, { useState } from 'react';
import { useNavigate } from 'react-router-dom';
import GeminiService, { isApiConfigured } from '../services/geminiApi';

export default function AIGeneratorShort() {
```

```
  const [topic, setTopic] = useState('');
  const [difficulty, setDifficulty] = useState('medium');
  const [numberOfQuestions, setNumberOfQuestions] = useState(5);
  const [loading, setLoading] = useState(false);
  const navigate = useNavigate();

  const generateQuiz = async () => {
    if (!topic.trim()) return;

    setLoading(true);
    let quizData;

    try {
      // 1. Try Gemini API
      if (isApiConfigured()) {
        quizData = await GeminiService.generateQuiz(topic, difficulty,
numberOfQuestions);
      } else {
        quizData = GeminiService.generateMockQuiz(topic, difficulty,
numberOfQuestions);
      }

      // 2. Format questions for QuizPlay page
      const questions = (quizData.quiz?.questions || quizData.questions ||
[]).map((q, i) => ({
        id: i + 1,
        question: q.questionText || q.question,
        options: q.options,
        correctAnswer: q.correctAnswer ?? 0,
      }));

      // 3. Save to localStorage
      localStorage.setItem("customQuiz", JSON.stringify({
        title: `AI Quiz: ${topic}`,
        topic,
        difficulty,
        questions,
        totalQuestions: questions.length
      }));
```

```
      // 4. Navigate to quiz page
      navigate("/quiz/custom");
    } catch (err) {
      console.error("Quiz generation failed", err);
    } finally {
      setLoading(false);
    }
  };

  return null; // removed UI for documentation
}
```

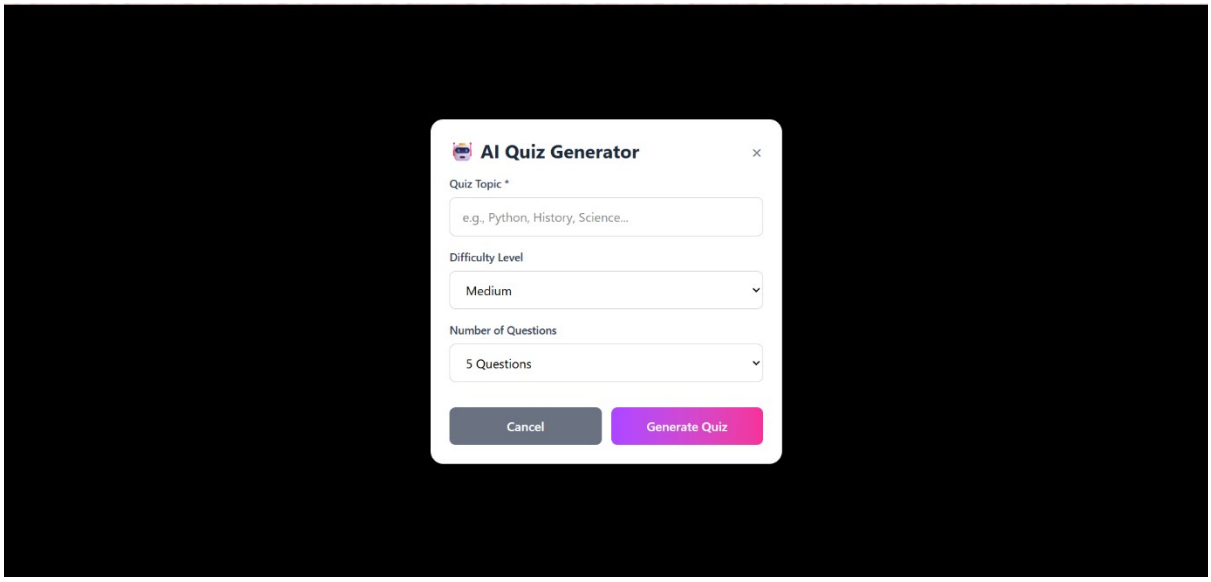

Figure 5.1: AI Quiz Generator Image

2. Context: AuthContext.jsx

Purpose:

The AuthContext component manages all authentication-related operations for the application using Firebase Authentication. It provides global access to login, register, Google login, logout, and user profile updates. It also listens to authentication changes using onAuthStateChanged and stores the current user in a global React Context. This ensures seamless authentication flow across all pages without prop drilling.

Code:

```
// AuthContext (Short Documentation Version)
```

```
import React, { createContext, useState, useEffect, useContext } from
"react";
import { auth, googleProvider } from "../firebase/config";
import {
  signInWithEmailAndPassword,
  createUserWithEmailAndPassword,
  signInWithPopup,
  signOut,
  onAuthStateChanged,
  updateProfile
} from "firebase/auth";

const AuthContext = createContext();
export const useAuth = () => useContext(AuthContext);

export function AuthProvider({ children }) {
  const [currentUser, setCurrentUser] = useState(null);

  // Email Login
  const login = (email, password) =>
    signInWithEmailAndPassword(auth, email, password);

  // Register New User
  const register = (email, password, name) =>
    createUserWithEmailAndPassword(auth, email, password).then((res)
=> {
      if (name) updateProfile(res.user, { displayName: name });
      return res;
    });

  // Google Login
  const loginWithGoogle = () => signInWithPopup(auth, googleProvider);

  // Logout
  const logout = () => signOut(auth);

  // Detect Logged-in User
  useEffect(() => {
    const unsubscribe = onAuthStateChanged(auth, (user) => {
```

```
      setCurrentUser(user);
    });
    return unsubscribe;
  }, []);

  return (
    <AuthContext.Provider value={{ currentUser, login, register,
loginWithGoogle, logout }}>
      {children}
    </AuthContext.Provider>
  );
}
```

3. Pages

- **FirebaseDebug.jsx**

Purpose:
This page is used only for development and debugging. It helps check whether Firebase and Authentication are properly initialized. It prints Firebase database instance and current logged-in user details on the console.

Code:

```
// Firebase Debug Component (Short Version)
import React from 'react';
import { useAuth } from '../context/AuthContext';
import { db } from '../firebase/config';

export default function FirebaseDebug() {
  const { currentUser } = useAuth();

  const checkFirebase = () => {
    console.log("Firebase DB:", db);
    console.log("Current User:", currentUser);
    console.log("User UID:", currentUser?.uid);
  };
}
```

- **Home.jsx**

Purpose:

The Home page is the main dashboard of the project. It welcomes logged-in users and provides access to features like AI quiz generation, browsing quizzes, and viewing available categories.

Code:

```
// Home Page (Short Version)
import React, { useState } from 'react';
import { useAuth } from '../context/AuthContext';
import AIGeneratorModal from '../components/AIGeneratorModal';

export default function Home() {
  const { currentUser } = useAuth();
  const [showAIModal, setShowAIModal] = useState(false);

  if (!currentUser) return <h2>Please log in to access quizzes</h2>;

  return (
    <>
      <h1>Welcome, {currentUser.displayName || currentUser.email}</h1>
      <button onClick={() => setShowAIModal(true)}>Generate AI
Quiz</button>
      {showAIModal && <AIGeneratorModal onClose={() =>
setShowAIModal(false)} />}
    </>
  );
}
```

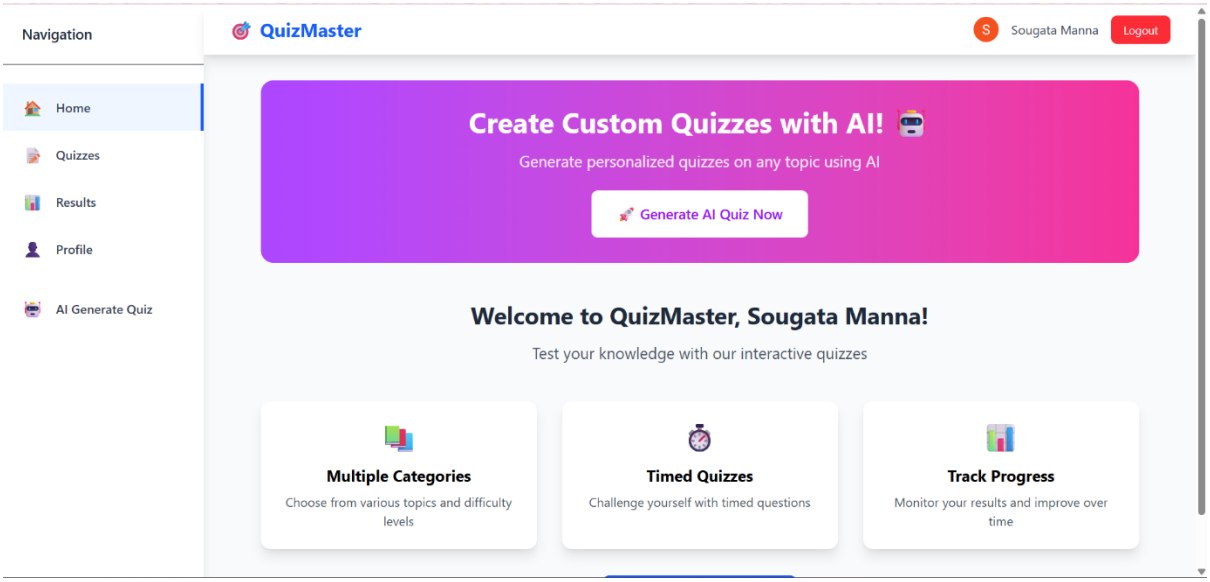

Figure 5.2: Home Page Image

- **Login.jsx**

Purpose:
The login page allows users to sign in using email-password or Google authentication. It uses Firebase Auth for all login operations.

Code:

```
// Login Page (Short Version)
import React, { useState } from 'react';
import { useAuth } from '../context/AuthContext';
import { useNavigate } from 'react-router-dom';

export default function Login() {
  const { login, loginWithGoogle } = useAuth();
  const [email, setEmail] = useState("");
  const [password, setPassword] = useState("");
  const navigate = useNavigate();
  const handleLogin = async (e) => {
    e.preventDefault();
    await login(email, password);
    navigate("/");
  };
  const handleGoogleLogin = async () => {
    await loginWithGoogle();
    navigate("/");
  }; }
```

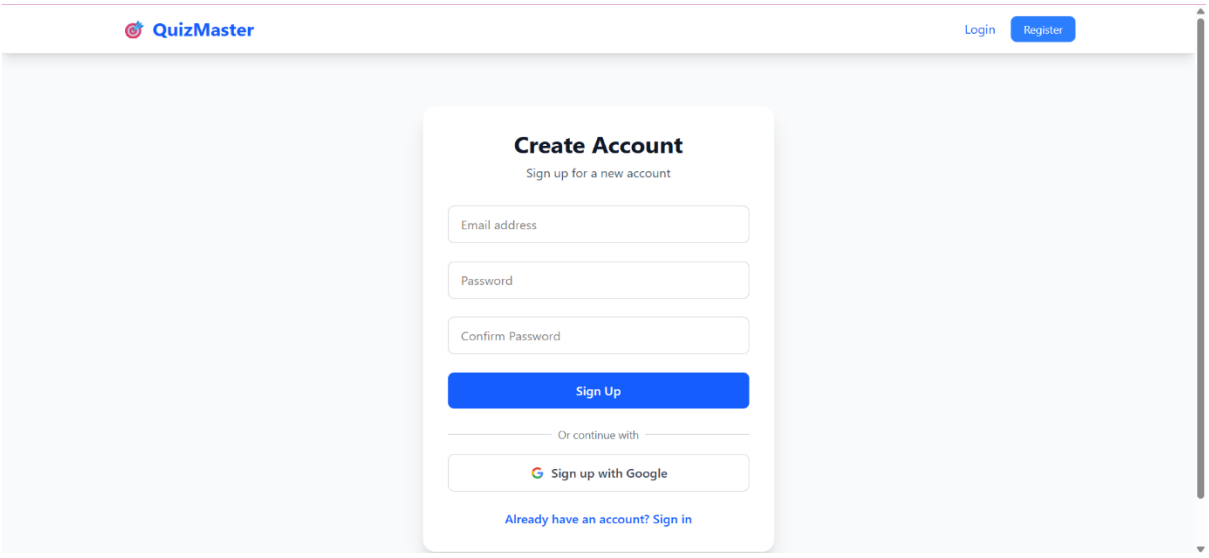

Figure 5.3: Login Page Image

- **Profile.jsx**

Purpose:
The Profile page displays user information stored in Firebase Authentication. Only logged-in users can access it.

Code:

```
// Profile Page (Short Version)
import React from 'react';
import { useAuth } from '../context/AuthContext';

export default function Profile() {
  const { currentUser } = useAuth();

  if (!currentUser) return <h2>Please log in to view profile</h2>;

  return (
    <>
      <h1>Your Profile</h1>
      <p>Name: {currentUser.displayName}</p>
      <p>Email: {currentUser.email}</p>
      <p>User ID: {currentUser.uid}</p>
    </>
  );
}
```

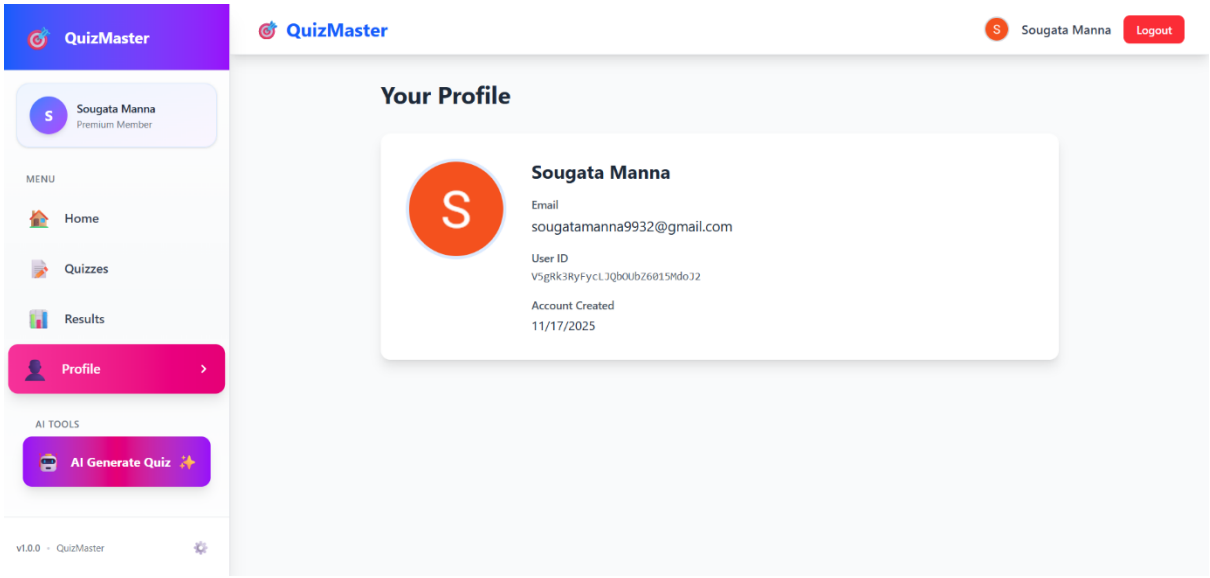

Figure 5.4: User Profile Image

- **QuizPlay.jsx**

Purpose

QuizPlay.jsx is the main quiz-taking screen of the project. It loads quiz questions (either predefined or AI-generated), handles user answers, manages quiz timing, calculates the final score, and stores results in Firebase and localStorage.

Code:

The component detects quiz type using:

id === "custom" || location.pathname === "/quiz/custom"

Result saved in Firebase Firestore using:

addDoc(collection(db, QUIZ_RESULTS_COLLECTION), resultData);

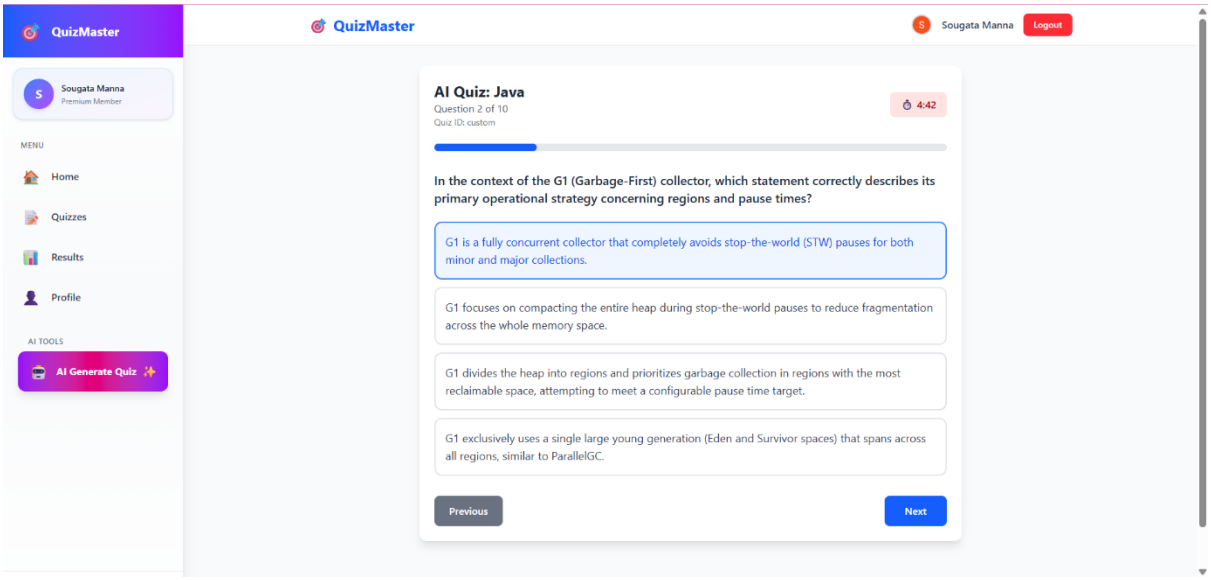

Figure 5.5: AI Generate Quiz Question Image

4. Services - geminiApi.js

Purpose of the File

This file acts as a service layer that communicates with the Google Gemini AI API.

It handles:

- Sending prompts to Gemini
- Receiving AI-generated quiz data
- Validating & cleaning JSON responses
- Providing fallback mock quiz data
- Handling errors and missing API keys

This file separates AI logic from UI, improving maintainability and

modularity.

Code:

API Initialization-

const ai = new GoogleGenAI({ apiKey: API_KEY });

It reads the key from:

.env → VITE_GEMINI_API_KEY

2. Generating Quiz Using AI

The main function:

GeminiService.generateQuiz(topic, difficulty, numberOfQuestions)

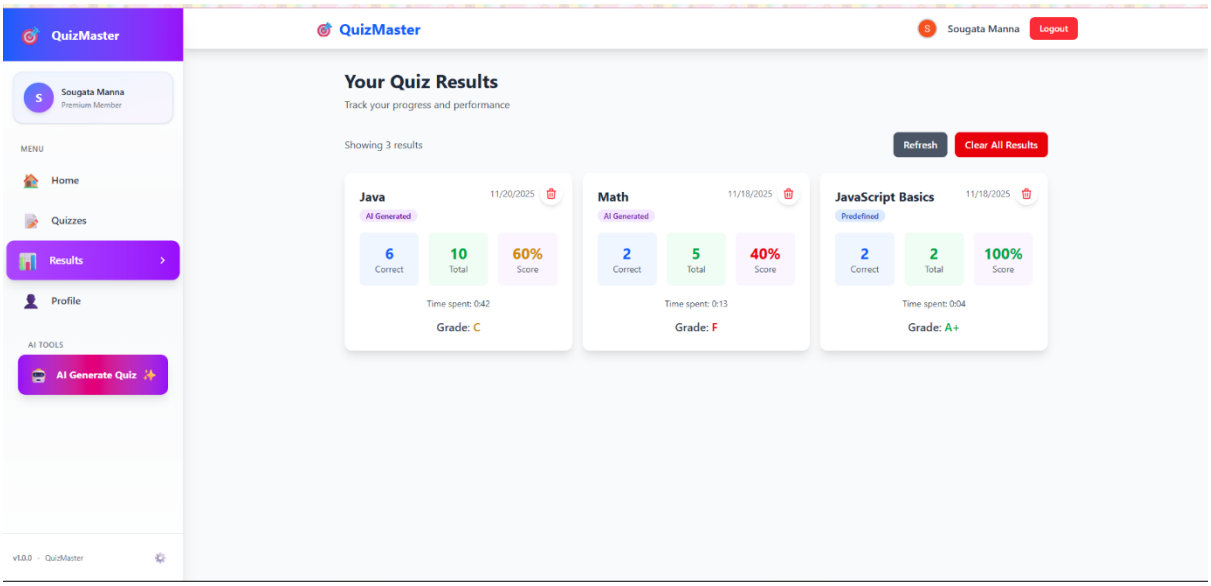

Figure 5.6: Quiz Result Image

5. i) App.jsx, ii) main.jsx, iii) index.css

i) App.jsx

App.jsx serves as the central controller for routing and layout in the Online Quiz Application. It integrates React Router for navigation and uses protected routes to ensure that only authenticated users can access quiz pages. The AuthProvider wraps the entire application to provide global authentication state. Common UI components such as Navbar and Sidebar are included for logged-in users. This modular structure improves security, maintainability, and navigation flow.

Code:

```
// App.jsx (Short Documentation Version)
import React from "react";
import { BrowserRouter, Routes, Route, Navigate } from "react-router-
dom";
```

```
import { AuthProvider, useAuth } from "./context/AuthContext";
import Home from "./pages/Home";
import Login from "./pages/Login";
import Register from "./pages/Register";
import QuizPlay from "./pages/QuizPlay";
import Results from "./pages/Results";
import Profile from "./pages/Profile";
import QuizList from "./pages/QuizList";

function ProtectedRoute({ children }) {
  const { currentUser } = useAuth();
  return currentUser ? children : <Navigate to="/login" />;
}

function App() {
  return (
    <BrowserRouter>
      <AuthProvider>
        <Routes>
          <Route path="/login" element={<Login />} />
          <Route path="/register" element={<Register />} />
          <Route path="/" element={<ProtectedRoute><Home
/></ProtectedRoute>} />
          <Route path="/quizzes" element={<ProtectedRoute><QuizList
/></ProtectedRoute>} />
          <Route path="/quiz/:id" element={<ProtectedRoute><QuizPlay
/></ProtectedRoute>} />
          <Route path="/results" element={<ProtectedRoute><Results
/></ProtectedRoute>} />
          <Route path="/profile" element={<ProtectedRoute><Profile
/></ProtectedRoute>} />
        </Routes>
      </AuthProvider>
    </BrowserRouter>
  );
}
export default App;
```

ii) main.jsx

This file is the entry point of the React application.

It renders the App component into the root HTML element.

✓ **Responsibilities:**

- Import global CSS (index.css)
- Enable React Strict Mode
- Render the entire application

Code:

```
// main.jsx (Short Version)
import React from "react";
import ReactDOM from "react-dom/client";
import App from "./App";
import "./index.css";

ReactDOM.createRoot(document.getElementById("root")).render(
  <React.StrictMode>
    <App />
  </React.StrictMode>
);
```

iii) index.css

This file simply imports the Tailwind framework so utility classes work throughout the app.

Code:

```
/* index.css */
@import "tailwindcss";
```

6. Index.html

index.html is the base HTML template used by Vite + React.
React applications do not have multiple HTML pages—there is only one main HTML file, and all pages are rendered inside it through JavaScript.

- **Root Container**

```
<div id="root"></div>
```

This is where the entire React application gets rendered.
React replaces this empty with the UI defined in your components.

- **Linking main entry file**

```
<script type="module" src="/src/main.jsx"></script>
```

This loads the React application.
main.jsx mounts the <App /> component into the #root div.

- **Meta Tags**
- charset="UTF-8" — character encoding
- viewport="width=device-width, initial-scale=1.0" — makes the app responsive on mobile

- **Favicon**

<link rel="icon" type="image/svg+xml" href="/vite.svg" />
This sets the small icon visible on browser tabs.

7. **package.json**

- package.json is the **core configuration file** of the React + Vite project.
- Project metadata
- (name, version, module type)
- Scripts
- (commands such as npm run dev or npm run build)
- Dependencies
- (all packages required for the app to run)
- DevDependencies
- (development tools like Vite, ESLint, React types)
- This file is automatically created when using Vite + React and is essential for project execution, bundling, and deployment.

Project Details

"name": "quiz-app",
"version": "0.0.0",
"type": "module"

- Defines the project name
- ES Module support (type: module)
- Private ensures it isn't accidentally published to npm

2. Scripts (Commands)

```
"scripts": {
  "dev": "vite",
  "build": "vite build",
```

```
  "lint": "eslint .",
  "preview": "vite preview"
}
```

- **dev** → starts development server
- **build** → bundles the React app for production
- **lint** → checks code quality
- **preview** → preview the production build

3. Dependencies (Required for App Functionality)

```
"dependencies": {
  "@google/genai": "...",
  "firebase": "...",
  "react-router-dom": "...",
  "tailwindcss": "...",
  "react": "...",
  "react-dom": "..."
}
```

Some key libraries:

- **React**

The main framework for building UI

- **React Router**

Handles navigation between pages
(ex: /login, /quiz/:id)

- **Firebase**

Used for:

- Authentication
- Firestore database
- Google login

- **Google GenAI**

Used to generate AI quizzes with Gemini API

- **Tailwind CSS**

Utility-based CSS framework used for responsive UI

- **Framer Motion**

Used for animation effects

- **Lucide & FontAwesome Icons**

Provides icon components

Dev Dependencies (Developer Tools)

```
"devDependencies": {
```

```
  "vite": "...",
  "@vitejs/plugin-react": "...",
  "eslint": "...",
  "@types/react": "..."
}
```

These tools are needed only in development:

- **Vite**

A fast build tool for React projects.

- **ESLint**

Used to maintain clean and error-free code.

- **React Type Definitions**

Improves developer experience and editor auto-completion.

8. vite.config.js

vite.config.js is the **configuration file for Vite**, which is the build tool used in your React project.
It allows you to add plugins, customize builds, and configure the development server.
This project uses Vite for:

✓ Extremely fast development server
✓ Hot Module Replacement (HMR)
✓ Optimized production build
✓ Handling React JSX transformation
✓ Integrating Tailwind CSS

Code:

```
// vite.config.js (Short Version for Documentation)
import { defineConfig } from "vite";
import react from "@vitejs/plugin-react";
import tailwindcss from "@tailwindcss/vite";

export default defineConfig({
  plugins: [
    react(),       // Enables React fast refresh and JSX
    tailwindcss()  // Enables Tailwind CSS
  ]
});
```

Chapter 6: **BackEnd**

- **Firebase Configuration & Firestore Database (Documentation)**

1. Purpose of firebase/config.js

This file is responsible for connecting the React application with Google Firebase services.
It sets up:

- **Firebase Authentication** (Email/Password & Google Login)
- **Cloud Firestore Database**
- A centralized configuration for the entire project

This acts like the **backend** for your quiz application because Firebase handles:

✓ User Login
✓ Register
✓ Google Sign-in
✓ Storing quiz results in database

2. Firebase Services Used

Firebase Authentication

Used for:

- Create account
- Login
- Google Login
- Track current user
- **Cloud Firestore Database**

Used for:

- Saving quiz scores
- Saving custom AI quiz results
- Fetching previous results
- Storing user data securely

- **Firestore Database Structure**

quizResults (collection)
├── a0QIFv... (document)
│ ├── createdAt: "18 Nov 2025"
│ ├── isCustom: true
│ ├── percentage: 40
│ ├── quizId: "custom"
│ ├── quizTitle: "AI Quiz: Math"

```
    │       ├── score: 2
    │       ├── timeSpent: 13
    │       ├── userEmail: "email..."
    │       ├── userId: "V5gR..."
    │       └── totalQuestions: 5
```

Code:

```
// firebase/config.js (Short Version for Documentation)
import { initializeApp } from "firebase/app";
import { getAuth, GoogleAuthProvider } from "firebase/auth";
import { getFirestore } from "firebase/firestore";

const firebaseConfig = {
  apiKey: "...",
  authDomain: "quiz-app.firebaseapp.com",
  projectId: "quiz-app",
  storageBucket: "quiz-app.firebasestorage.app",
  messagingSenderId: "...",
  appId: "..."
};

// Initialize Firebase
const app = initializeApp(firebaseConfig);

// Firebase Services
export const auth = getAuth(app);      // Authentication
export const db = getFirestore(app);   // Firestore database
export const googleProvider = new GoogleAuthProvider();

// Firestore Collection Name
export const QUIZ_RESULTS_COLLECTION = "quizResults";
```

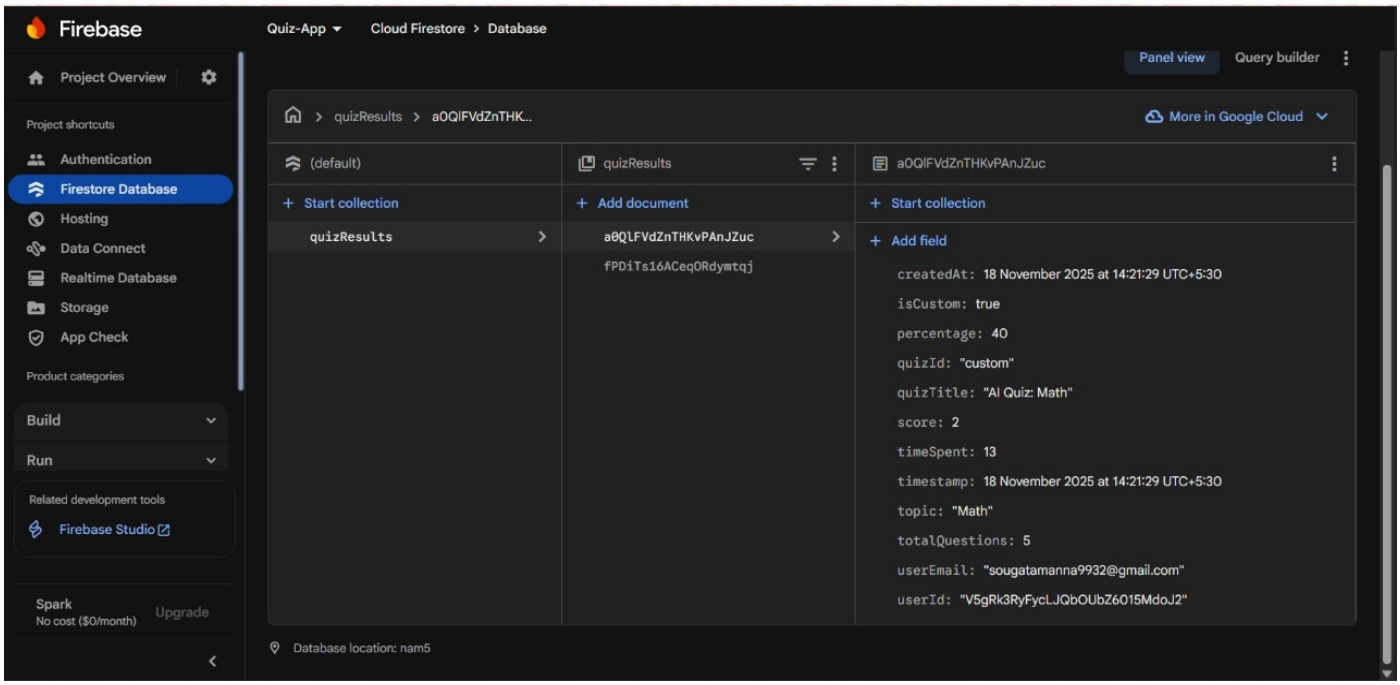

Figure 6.1: Data Store Image

Chapter 7: **Future Scope**

The Online Quiz Application developed in this project can be expanded in multiple directions to enhance its usability, intelligence, and real-world applicability. Several advanced features and improvements can be implemented in future work to make the system more robust and scalable. The major future scope areas are listed below:

7.1: Integration of a Question Bank Database

Currently, quizzes are generated from predefined arrays or AI-based generation.
Future extensions may include:

- A large centralized question bank
- Category-wise & difficulty-wise question sets
- Faculty/admin panel to upload questions

This will support standardized assessments.

7.2: Real-Time Leaderboard and Ranking System

A real-time ranking system can be added where users can:

- Compete with other learners
- View top scores
- Participate in live quiz events
  This will increase user engagement.

7.3: Advanced AI Quiz Generation

The AI generator can be upgraded to:

- Generate images & code-based questions
- Identify user strengths/weaknesses
- Generate adaptive quizzes (difficulty adjusts automatically)
- Provide personalized learning suggestions

7.4: Timed Exams & Proctoring Features

To support competitive exams, the system can add:

- Fixed exam windows
- Strict time limits
- Anti-cheating mechanisms

- Webcam monitoring (AI-based proctoring)

7.5: Mobile App Development

The system can be extended as:

- A dedicated Android/iOS mobile app
- With push notifications
- Offline quiz attempts
- Cloud synchronization

This will make the system more accessible.

Chapter 8: **Conclusion**

The development of the Online Quiz Application demonstrates the effectiveness of combining component-based front-end design with secure cloud authentication and AI-driven automation. ReactJS provided a modular and scalable interface, while Firebase ensured reliable user management. System testing confirmed high functional accuracy, fast performance, and smooth usability across devices.

The AI-powered quiz generator marks a notable improvement over traditional quiz systems by automatically creating topic-specific questions, reducing manual effort, and supporting personalized learning. The results module, with analytics such as scores and grades, helps users monitor performance and progress.

Overall, this project highlights the practical use of modern EdTech technologies, including SPA architecture, cloud services, and AI. It lays a strong foundation for future enhancements such as adaptive learning, admin dashboards, leaderboards, proctoring features, and mobile app deployment, making the system scalable and suitable for wider educational use.

Chapter 9: **References**

- React JS — https://react.dev/
- React Router — https://reactrouter.com/
- Firebase Authentication — https://firebase.google.com/docs/auth
- Firebase Firestore — https://firebase.google.com/docs/firestore
- MDN Web — https://developer.mozilla.org/
- Tailwind CSS — https://tailwindcss.com/docs
- Google Gemini API — https://ai.google.dev/
- OpenAI API — https://platform.openai.com/docs

➢ Complete Code Link — GitHub