Quiz Website

**By**

**Shruti Gaikwad– AF04959589**

**Sakshi Desai– AF04959587**

**Index**

|  |  |  |
| --- | --- | --- |
| **Sr.no** | **Topic** | **Page no** |
| 1 | Title of Project | 1 |
| 2 | Acknowledgement | 3 |
| 3 | Abstract | 4 |
| 4 | Introduction | 5 |
| 5 | Objective | 6 |
| 6 | System Analysis | 7 |
| 7 | System Design | 22 |
| 8 | Screenshots | 32 |
| 9 | Coding | 39 |
| 10 | Testing | 46 |
| 11 | Report | 47 |
| 12 | Future Scope | 48 |
| 13 | Conclusion | 48 |
| 14 | Bibliography | 49 |
| 15 | References | 49 |

# Acknowledgement

The project **“Quiz Website”** is the Project work carried out by

|  |  |
| --- | --- |
| **Name** | **Enrollment No** |
| **Shruti Gaikwad** | **AF04959589** |
| **Sakshi Desai** | **AF04959587** |

Under the Guidance.

We are thankful to my project guide for guiding me to complete the Project.

His suggestions and valuable information regarding the formation of the Project Report have provided me a lot of help in completing the Project and its related topics.

We are also thankful to my family member and friends who were always there to provide support and moral boost up.

# Abstract

# The Quiz Website project is an online platform where users can attempt quizzes on various topics. The system provides features like user authentication, quiz management, question randomization, timer-based attempts, scoring, and result analysis. Administrators can manage quizzes, add questions, and track user performance. The platform ensures accessibility, efficiency, and engagement for both learners and administrators.

# The system is developed using React.js for the frontend, ensuring a responsive and interactive user experience, while Node.js and Express.js handle the backend operations. The application uses MySQL as the database to securely store user data and information..

# This system replaces traditional paper-based quizzes with a digital solution that saves time, provides instant feedback, and allows scalability

# Introduction

In today’s digital learning environment, web applications have become an essential tool for education and training. Online quizzes are an efficient way to assess knowledge and provide real-time feedback to users. The Quiz Website aims to deliver a user-friendly platform where users can participate in quizzes, track their performance, and improve their learning outcomes. Administrators can easily manage quizzes and questions through a secure interface

Our project is a Quiz Website designed to make learning fun and interactive. The website allows users to take quizzes on different categories such as HTML, CSS, JavaScript, SQL, and more. It provides a user-friendly interface where learners can test their knowledge, get instant results, and track their progress. By combining education with technology, the platform aims to help students and beginners strengthen their concepts in an engaging way.

The website also maintains a record of recent scores, so users can monitor

their improvement over time. Overall, this project demonstrates how web

technologies can be used to create a practical, interactive, and scalable learning platform.

# Objectives

## Objective of the Present Work

The objectives of this project are as follows:

* To provide an interactive platform for users to practice quizzes on different categories like HTML, CSS, JavaScript, SQL, etc.
* To help learners test their knowledge and receive instant feedback with scores and correct answers.
* To allow users to track their recent quiz attempts and monitor their improvement over time.
* To design a user-friendly interface that makes learning simple, engaging, and accessible.
* To implement a secure and efficient system using React, , and SQL for smooth performance and data management.

**System analysis**

The Quiz Website is developed to design a user-friendly interface that makes learning simple, engaging, and accessible.

**Existing System**

In the existing learning systems, students mostly rely on traditional study methods such as textbooks, handwritten notes, and manual practice questions. While there are some online quiz platforms available, many of them are either complex to use, lack category-wise organization, or do not provide proper result tracking. Moreover, most free tools do not allow users to maintain a record of their past performance, making it difficult for learners to analyze their progress over time.

**Proposed System**

Proposed Quiz Website provides a simple, interactive, and user-friendly platform where learners can attempt quizzes in different categories such as HTML, CSS, JavaScript, SQL, and more. The app offers instant results, score tracking, and maintains recent performance history, allowing users to evaluate their learning journey. Built using React for frontend, for backend, and SQL for database, the system ensures secure data handling, smooth performance, and scalability for future enhancements

**.Feasibility Study**

* Technical: Built with React.js, Node.js, Express.js, and MySQL (all open-source).
* Operational: Easy to use with basic internet access.
* Economic: Cost-effective due to free technologies.

**Requirements**

**Functional**: Login/Signup, guide, track scores, provide instant highest scores

**Non-Functional**: Secure, fast, scalable, and mobile-friendly.

* 1. **PROBLEM DEFINITION**

In the existing learning systems, students mostly rely on traditional study methods such

as textbooks, handwritten notes, and manual practice questions. While there are

Some online quiz platforms available, many of them are either complex to use, lack

Wise organization, or do not provide proper result tracking. Moreover, most free tools

do not allow users to maintain a record of their past performance, making it difficult

For learners to analyze their progress over time.

* 1. **Preliminary Investigation Purpose**

Proposed Quiz Website provides a simple, interactive, and user-friendly platform where learners can attempt quizzes in different categories such as HTML, CSS, JavaScript, SQL, and more. The app offers instant results, score tracking, and maintains recent performance history, allowing users to evaluate their learning journey. Built using React for frontend, for backend, and SQL for database, the system ensures secure data handling, smooth performance, and scalability for future enhancements

**Benefits**

* Provides \*instant results\* after quizzes.
* Helps in \*tracking recent scores\*.
* Offers a \*simple and user-friendly interface\*.
* Supports \*category-wise quizzes\* (HTML, CSS, JS, SQL).
* Accessible \*anytime, anywhere\* with internet.

**Proposed System**

The proposed system is a quiz website that allows users to take category-wise

quizzes such as HTML, CSS, JavaScript, and SQL. It provides instant results, stores recent scores, and offers a simple, user-friendly interface. Built using React (frontend), Node.js with Express (backend), and MySQL (database), the system ensures smooth performance, secure

* 1. **Feasibility Study**
     + - The feasibility study ensures that the Quiz Website is practical, cost-effective, and user-friendly.
       - Technical Feasibility:
       - The app is built using React.js, Node.js, Express.js, and MySQL – all open-source, reliable, and widely supported technologies.
       - Operational Feasibility:
       - The system is simple to use with an intuitive interface. Users only need basic internet access and a device (mobile/PC).
       - Economic Feasibility:
       - Development and maintenance costs are minimal as the app uses free technologies, making it affordable and sustainable.
       - Modular development ensures different parts of the system are built incrementally, improving flexibility.
  2. Project Planning

Purpose of Project Planning

Project planning ensures that the development of the **Quiz Website** follows a structured approach. It helps define the workflow, resource allocation, timelines, and risks involved to deliver the system efficiently.

Phases Covered in the Plan

The Quiz Website is planned and developed in different phases to ensure smooth progress and timely completion.

Phases of Project Planning:

Requirement Analysis:

Collect user needs such as User name, Email, Quiz category selected, Answer Submitted, Quiz score and Result, Recent attempt

System Design:

Plan architecture, database structure (MySQL), and UI/UX layout using React.js.

Development:

Frontend: Build responsive interface with React.js,

Backend: Implement APIs and authentication with Node.js & Express.js.

Database: Store user and Score,Questions data in MySQL.

Testing:

Test functionality, security, and performance to ensure a smooth user experience.

Deployment:

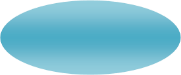
Launch the app on a server for user access.

Maintenance & Updates:

Updating questions to keep the system accurate and engaging, fix bugs, and add new features.

Project Planning Goal:

To deliver a user-friendly quiz website on time and ensure smooth learning with reliable performance



**Start**

Preliminary

System Analysis

System Design

Coding

Testing

Security

Implementation

**Stop**

* 1. Software Requirement Specification (SRS)

The **Software Requirement Specification (SRS)** outlines the fundamental requirements of the **Quiz Website** to ensure efficient functionality, usability, and maintainability.

System Overview

The Quiz Website provides an effective and interactive way for learners to test their knowledge and improve their skills. With instant results, score tracking, and a simple interface, it makes self-learning easy

**Frontend: Developed using React.js for a responsive and interactive user interface.**

**Backend: Built with Node.js and Express.js to handle authentication, and API requests.**

**Database: MySQL is used for storing user details, login credentials, and Score, Question information.**

**Users:**

**Registered Users – login and access quizzes by category, Attempt quiz and submit answers, and get instant result with scores, and track recent scores**

Software & Hardware Requirements Software Requirements

* + **Frontend**: React JS for responsive
  + **Backend**: Node JS for data handling and security

**Database**: MySQL for storing questions, scores, and user details

Hardware Requirements

* + **Processor**: Intel i5 or higher
  + **RAM**: Minimum 8GB
  + **Storage**: At least 100GB for database and media files
  + **Connectivity**: Internet access for real-time updates
  1. Functional Requirements

The **Quiz Website** must support the following key functions:

1. **The system should allow users to register and login.**
2. **The system should allow users to select quiz categories (HTML, CSS, JavaScript, SQL, etc.).**
3. **The system should display multiple-choice questions with options.**
4. **The system should allow users to submit answers.**
5. **The system should evaluate answers and calculate scores instantly.**
6. **The system should store and display recent scores for each user.**
7. **The system should allow users to reattempt quizzes.**
   1. ​Software Engineering Paradigm

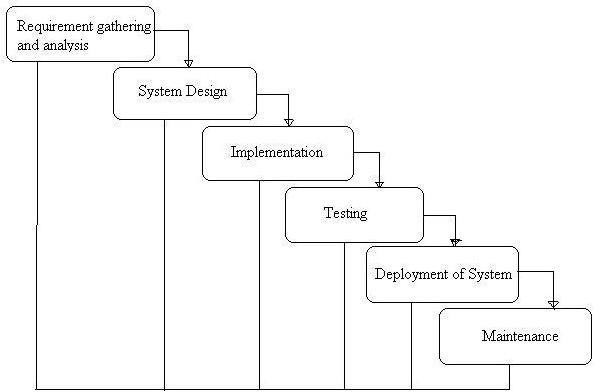
The development of the **Quiz Website** follows a structured approach to ensure efficiency, reliability, and maintainability. The chosen paradigm helps streamline the project by defining clear phases while allowing iterative improvements.

Development Model: Adapted Waterfall Model

The **Waterfall Model** is traditionally a linear approach, but for this project, an **iterative feedback mechanism** is incorporated. This helps refine earlier phases based on insights gathered during implementation.

Key Adaptations in the Waterfall Model:

1. **Structured Phase Progression** – Each stage follows a defined sequence, ensuring clarity in execution.
2. **Iterative Refinements** – Feedback loops allow adjustments, especially between testing and coding.
3. **Defined Milestones** – Each stage reaches completion before moving to the next phase.
4. **Flexible Adjustments** – Overlapping is permitted when necessary to enhance efficiency.



Phases of Development

### Requirement Analysis & System Study

* + Identifying project goals, challenges, and functional specifications.
  + Gathering stakeholder requirements and defining core functionalities.

### System Design

* + Structuring the **database, modules, and architecture**.
  + Designing **user interfaces** for optimal accessibility.

### Implementation (Coding)

* + Backend development using **Node JS**
  + Frontend design using  **React JS**.
  + Database integration with **MySQL**.

### Testing & Debugging

* + Unit testing, integration testing, and usability checks.
  + Debugging for performance improvements.

### Deployment & Maintenance

* + Hosting on a scalable environment.
  + Continuous updates for feature enhancements.
  1. Data Flow Diagram:

A Data Flow Diagram (DFD) is a traditional visual representation of the information flows within a system. A neat and clear DFD can depict the right amount of the system requirement graphically. It can be manual, automated, or a combination of both.

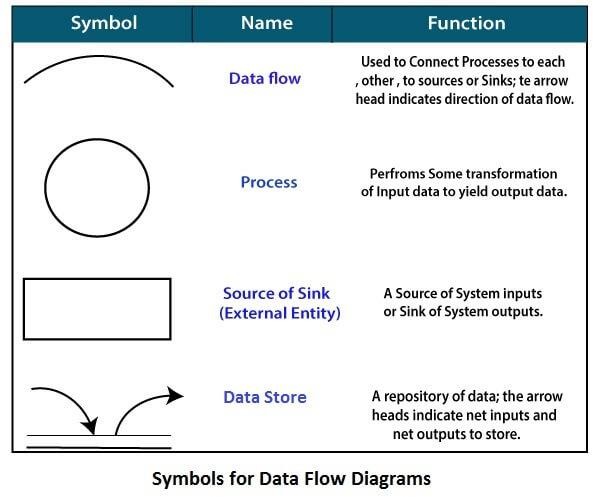
It shows how data enter and leaves the system, what changes the information, and where data is stored.

The objective of a DFD is to show the scope and boundaries of a system as a whole. It may be used as a communication tool between a system analyst and any person who plays a part in the order that acts as a starting point for redesigning a system. The DFD is also called as a data flow graph or bubble chart.

**The following observations about DFDs are essential:**

1. All names should be unique. This makes it easier to refer to elements in the DFD.
2. Remember that DFD is not a flow chart. Arrows is a flow chart that represents the order of events; arrows in DFD represents flowing data. A DFD does not involve any order of events.
3. Suppress logical decisions. If we ever have the urge to draw a diamond-shaped box in a DFD, suppress that urge! A diamond-shaped box is used in flow charts to represents decision points with multiple exists paths of which the only one is taken. This implies an ordering of events, which makes no sense in a DFD.
4. Do not become bogged down with details. Defer error conditions and error handling until the end of the analysis.

Standard symbols for DFDs are derived from the electric circuit diagram analysis and are shown in fig:

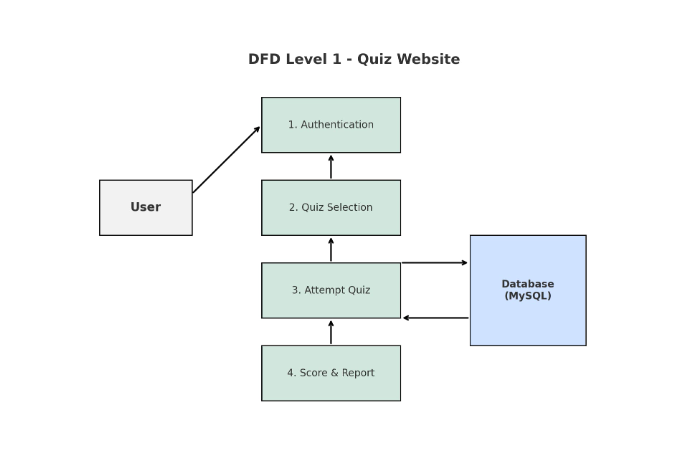


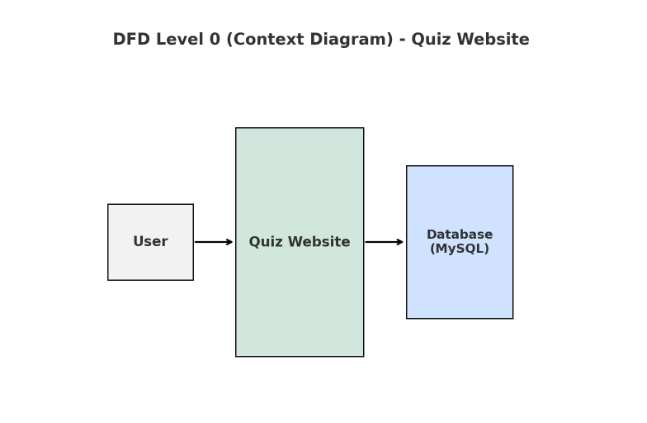
**Circle:** A circle (bubble) shows a process that transforms data inputs into data outputs.

**Data Flow:** A curved line shows the flow of data into or out of a process or data store.

**Data Store:** A set of parallel lines shows a place for the collection of data items. A data store indicates that the data is stored which can be used at a later stage or by the other processes in a different order. The data store can have an element or group of elements.

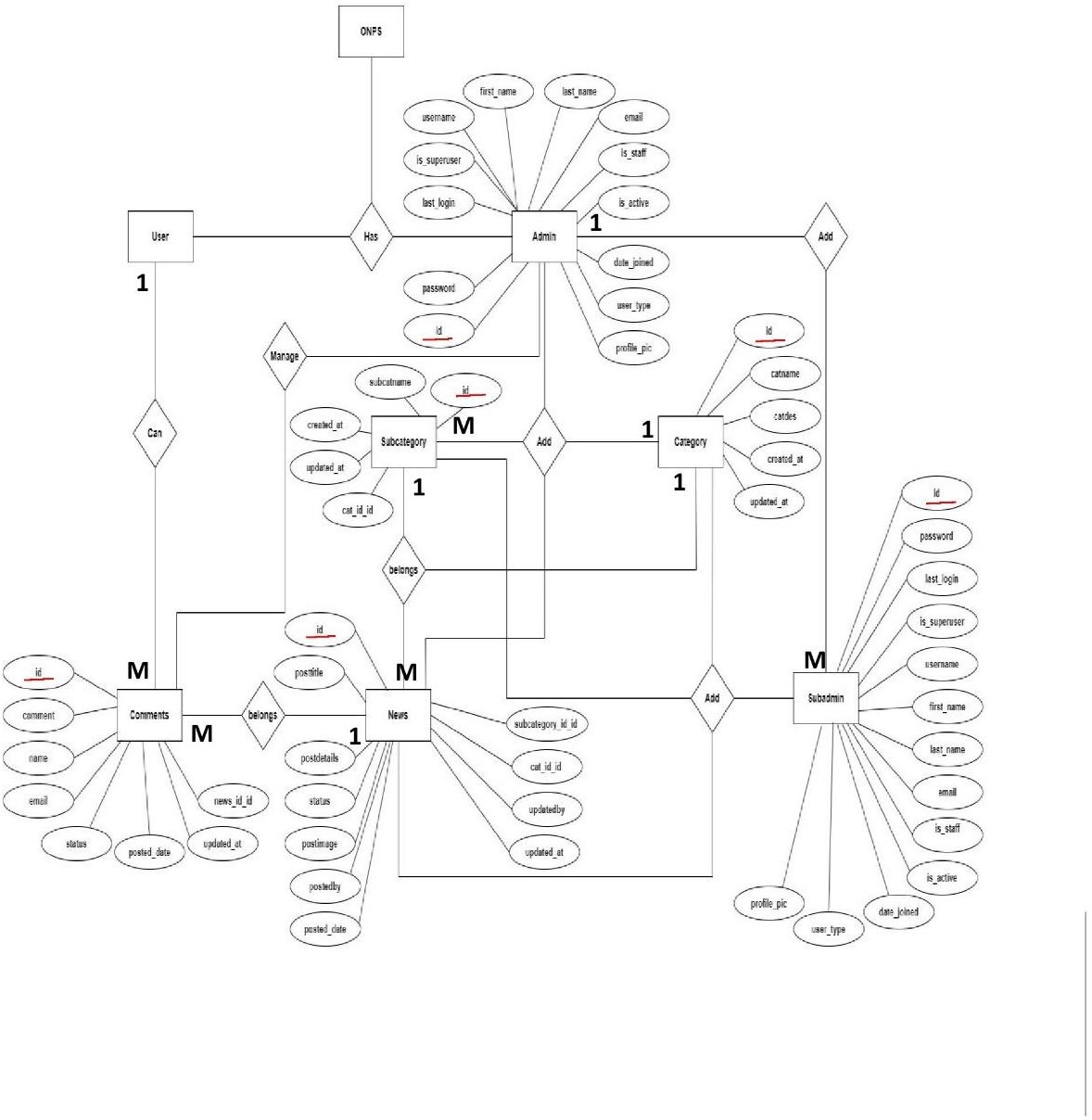
**Source or Sink**: Source or Sink is an external entity and acts as a source of system inputs or sink of system outputs.





ER diagram

The **ER Diagram** of the **Online News Portal System** shows key relationships between **Users, Admins, News Articles, Categories, Comments, and Sentiment Analysis**, ensuring structured data management and insightful moderation.



1. System design
   1. Module
      * The system design defines the architecture, components, and interactions of the **Quiz Website**. It ensures that the application is efficient, scalable, and easy to use.

**User Module**

* User Registration and Login
* Profile management
* View attempt history

### ****Quiz Module****

* Display Available quizzes
* Start/attempt Quiz

### ****Question Module (Optional / Future Scope)****

* Fetch questions from databases
* Display options
* Validate selected answer

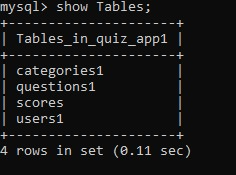
### ****Score/Result Module****

* Calculate score
* Show Result after quiz
* Store score in database
* View Past result
  1. DATA STRUCTURE OF ALL MODULES:

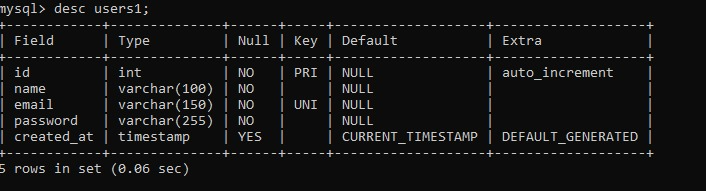
We have organized one database **Food Recipes** for system design. It can be accessed directly or sequentially by registered. The database determines files, record, fields, and characters. It can be easily controlled and updated. **“Online Food Recipe Web Application” contains 15 MySQL tables(In this MySQL 6 table is customized and 9 table) :**

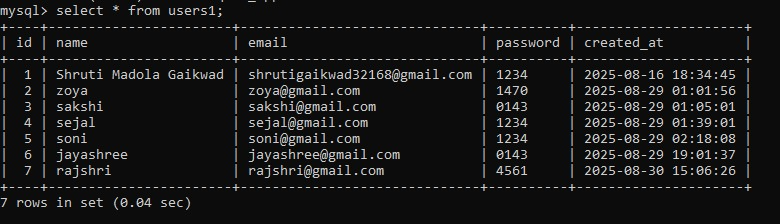
### Customized Tables Details

### Tables in Databases

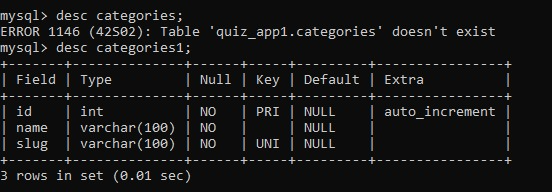


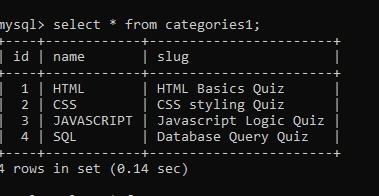
**UsersTables**.



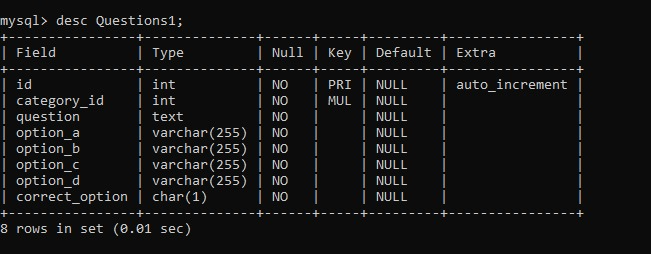


**Categories Table**

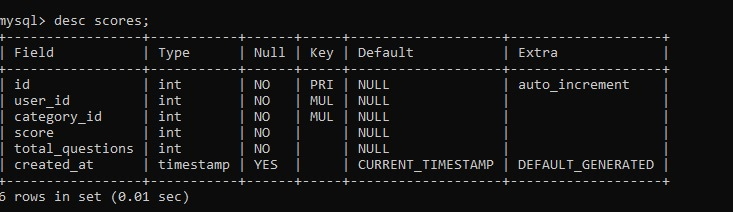




**Question Table**



**Scores table**



PROCEDURAL DESIGN:

Process logic (flowchart ) of each module

## User Panel Design

The **User Panel** is designed to provide an easy and interactive interface for users to register, attempt quiz and view their result. It includes the following components:

### ****Features:****

1. **Login & Registration** – Secure authentication for new and existing users.
2. **Dashboard/Homepage** – Displays available quiz categories and recent scores
3. **Question Browsing** – View and select categories such as Html, Css, js, Sql
4. **Quiz Attempt** – Attempt Question one by one
5. **Score and Result Page** – show final scores, correct/incorrect answer and performance
6. **Recent Score/History**– Save and display past quiz result for progress
7. **User Profile** – Manage personal details and view Quiz activity.

### ****Design Goals:****

* Simple and user-friendly navigation.
* Responsive interface for desktop and mobile.
* Personalized experience with Real time and Calculation

## Admin Panel Design

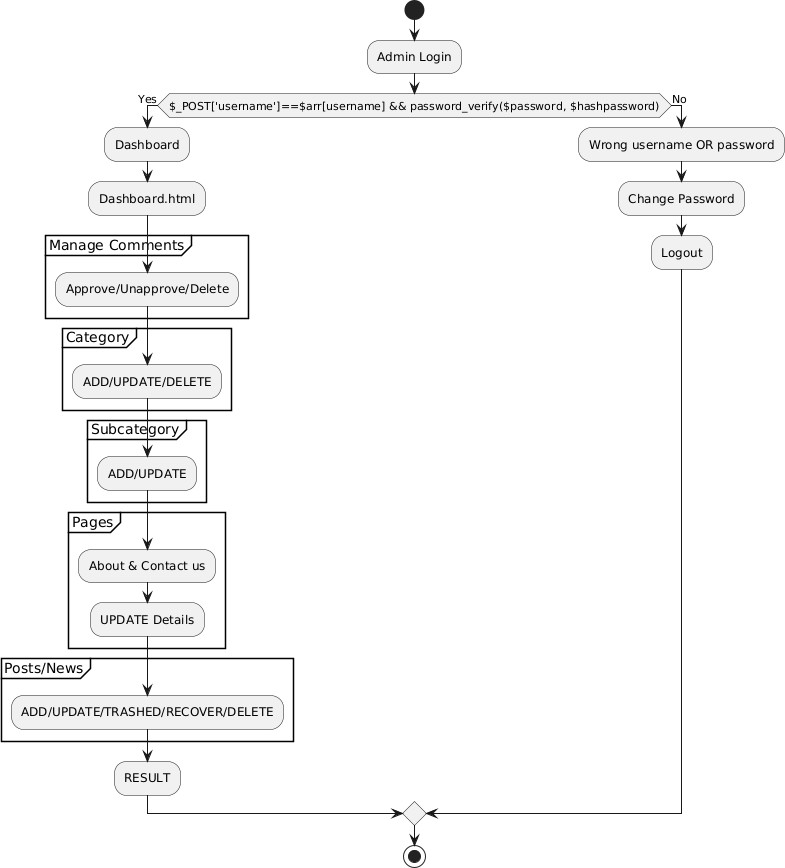
The **Admin Panel** is designed to manage the overall functioning of the Quiz Website. It provides tools for handling Quiz, users, and categories.

### ****Features:****

1. **Admin Login** – Secure access for administrators.
2. **Recipe Management** – Add, edit, update, or delete Questions.
3. **Category Management** – Create and manage Quiz categories (e.g., Html, css, MySQL).
4. **User Management** – View and Attempt Quiz.
5. **Score and Result Page** – show final scores, correct/incorrect answer and performance
6. **Dashboard/Homepage** – Displays available quiz categories and recent scores

### ****Design Goals:****

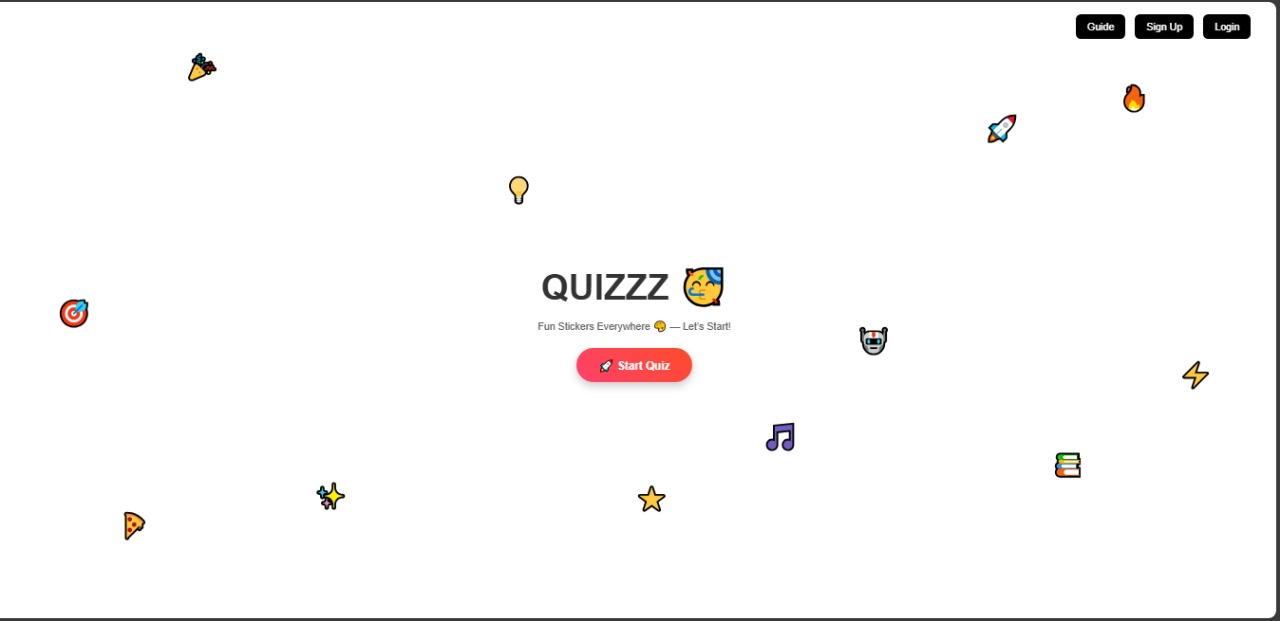
* Simple and user-friendly navigation.
* Responsive interface for desktop and mobile.
* Personalized experience with Real time and Calculation
* .

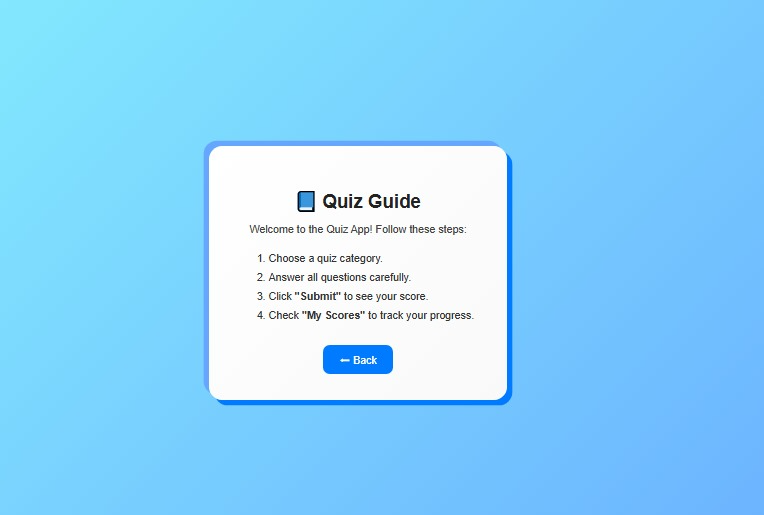


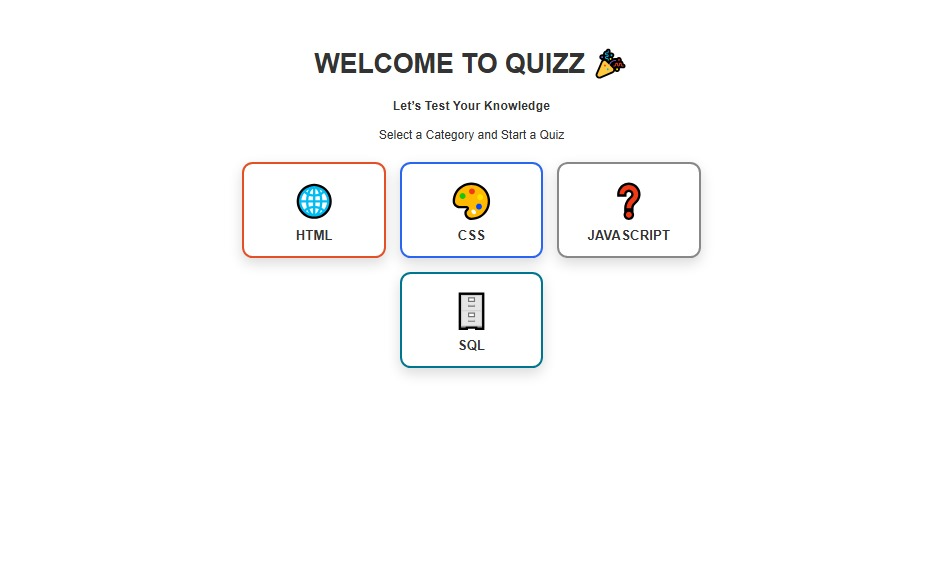
**Fig. 4.2: Admin Login Flow Chart.**

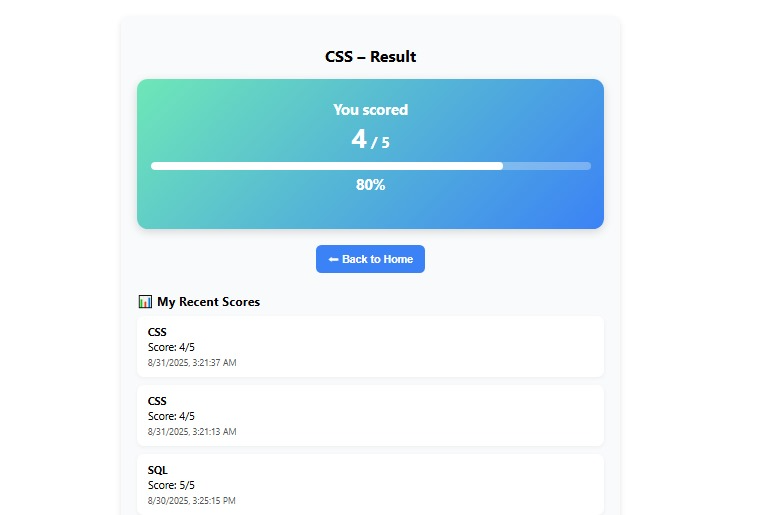
**SCREENSHOTS**

**Home Page**

****

****

****

****

Coding

Quiz Page

import React, { useEffect, useState } from "react";

import axios from "axios";

import { useParams, useNavigate, useLocation } from "react-router-dom";

function QuizPage() {

const { categoryId } = useParams();

const navigate = useNavigate();

const location = useLocation();

const categoryName = location.state?.categoryName || "Quiz";

const [questions, setQuestions] = useState([]);

const [idx, setIdx] = useState(0);

const [answers, setAnswers] = useState({});

const [loading, setLoading] = useState(true);

useEffect(() => {

axios

.get(`http://localhost:5000/questions/${categoryId}`)

.then((res) => setQuestions(res.data))

.catch((err) => {

console.error(err);

alert("Could not load questions");

})

.finally(() => setLoading(false));

}, [categoryId]);

if (loading) return <p style={{ textAlign: "center" }}>Loading questions...</p>;

if (questions.length === 0) return <p style={{ textAlign: "center" }}>No questions found.</p>;

const q = questions[idx];

const select = (letter) => {

setAnswers((prev) => ({ ...prev, [q.id]: letter }));

};

const next = () => {

if (idx < questions.length - 1) {

setIdx(idx + 1);

} else {

// ✅ Calculate score

let score = 0;

questions.forEach((qq) => {

if (answers[qq.id] === qq.correct\_option) score += 1;

});

// ✅ Save score in DB

axios

.post("http://localhost:5000/scores", {

user\_id: 1, // later replace with logged-in user id

category\_id: categoryId,

score: score,

total\_questions: questions.length,

})

.then((res) => {

console.log("Score saved:", res.data);

})

.catch((err) => {

console.error("Error saving score:", err);

});

// ✅ Navigate to result page with full info

navigate("/result", {

state: {

score,

total: questions.length,

categoryId, // Pass ID also

categoryName,

},

});

}

};

const selected = answers[q.id];

return (

<div style={{ maxWidth: 700, margin: "30px auto", padding: 16 }}>

<h2 style={{ textAlign: "center", marginBottom: "20px" }}>{categoryName} Quiz</h2>

<p style={{ textAlign: "center", marginBottom: "20px" }}>

Question {idx + 1} of {questions.length}

</p>

{/\* Question Card with Stickers \*/}

<div

style={{

position: "relative",

padding: "20px",

border: "1px solid #ddd",

borderRadius: "12px",

background: "#fff",

boxShadow: "0 4px 8px rgba(0,0,0,0.1)",

marginBottom: "20px",

overflow: "hidden",

}}

>

{/\* Stickers \*/}

<span style={{ position: "absolute", top: "10px", left: "10px", fontSize: "28px", opacity: 0.2 }}>🎯</span>

<span style={{ position: "absolute", top: "10px", right: "10px", fontSize: "28px", opacity: 0.2 }}>💡</span>

<span style={{ position: "absolute", bottom: "10px", left: "10px", fontSize: "28px", opacity: 0.2 }}>🤔</span>

<span style={{ position: "absolute", bottom: "10px", right: "10px", fontSize: "28px", opacity: 0.2 }}>📚</span>

{/\* Question Text \*/}

<p

style={{

fontSize: "20px",

fontWeight: "bold",

marginBottom: "16px",

position: "relative",

zIndex: 1,

}}

>

{q.question}

</p>

{/\* Options \*/}

{[

{ letter: "A", text: q.option\_a },

{ letter: "B", text: q.option\_b },

{ letter: "C", text: q.option\_c },

{ letter: "D", text: q.option\_d },

].map((opt) => (

<label

key={opt.letter}

style={{

display: "block",

padding: "10px 15px",

margin: "10px 0",

border: selected === opt.letter ? "2px solid #4CAF50" : "1px solid #ccc",

borderRadius: "8px",

cursor: "pointer",

background: selected === opt.letter ? "#e8f5e9" : "#fafafa",

transition: "0.3s",

position: "relative",

zIndex: 1,

}}

>

<input

type="radio"

name={`q-${q.id}`}

value={opt.letter}

checked={selected === opt.letter}

onChange={() => select(opt.letter)}

style={{ marginRight: 10 }}

/>

<strong>{opt.letter}.</strong> {opt.text}

</label>

))}

</div>

{/\* Buttons \*/}

<div style={{ marginTop: 16, display: "flex", justifyContent: "space-between" }}>

<button

onClick={() => setIdx(Math.max(0, idx - 1))}

disabled={idx === 0}

style={{

padding: "8px 16px",

border: "none",

borderRadius: "8px",

background: "#f0f0f0",

cursor: idx === 0 ? "not-allowed" : "pointer",

}}

>

Previous

</button>

<button

onClick={next}

disabled={!selected}

style={{

padding: "8px 16px",

border: "none",

borderRadius: "8px",

background: selected ? "#4CAF50" : "#ccc",

color: "#fff",

cursor: selected ? "pointer" : "not-allowed",

}}

>

{idx === questions.length - 1 ? "Submit" : "Next"}

</button>

</div>

</div>

);

}

export default QuizPage;

const express = require("express");

const mysql = require("mysql");

const cors = require("cors");

const app = express();

app.use(cors());

app.use(express.json());

// MySQL connection

const db = mysql.createConnection({

host: "localhost",

user: "root",

password: "shruti@123",

database: "quiz\_app1"

});

db.connect(err => {

if (err) {

console.log("Database connection failed:", err);

} else {

console.log("Connected to MySQL database");

}

});

// ✅ API 1: Fetch categories

app.get('/categories', (req, res) => {

db.query('SELECT \* FROM categories1', (err, results) => {

if (err) return res.status(500).json(err);

res.json(results);

});

});

// ✅ API 2: Fetch questions by category

app.get('/questions/:category\_id', (req, res) => {

const categoryId = req.params.category\_id;

db.query('SELECT \* FROM questions1 WHERE category\_id = ?', [categoryId], (err, results) => {

if (err) return res.status(500).json(err);

res.json(results);

});

});

// ✅ SIGNUP API

app.post("/signup", (req, res) => {

const { name, email, password } = req.body;

console.log("📩 Signup request received:", req.body);

if (!name || !email || !password || name.trim() === "" || email.trim() === "" || password.trim() === "") {

return res.status(400).json({ success: false, message: "All fields are required" });

}

const checkSql = "SELECT id FROM users1 WHERE email = ?";

db.query(checkSql, [email], (err, results) => {

if (err) return res.status(500).json({ success: false, message: "Database error" });

if (results.length > 0) {

return res.json({ success: false, message: "User already exists with this email" });

}

const sql = "INSERT INTO users1 (name, email, password) VALUES (?, ?, ?)";

db.query(sql, [name, email, password], (err, result) => {

if (err) return res.status(500).json({ success: false, message: "Error creating user" });

res.json({ success: true, message: "User registered successfully!" });

});

});

});

// ✅ LOGIN API

app.post("/login", (req, res) => {

const { email, password } = req.body;

if (!email || !password) {

return res.status(400).json({ success: false, message: "Email and password are required" });

}

const sql = "SELECT id, name, email FROM users1 WHERE email = ? AND password = ?";

db.query(sql, [email, password], (err, results) => {

if (err) return res.status(500).json({ success: false, message: "Server error" });

if (results.length > 0) {

res.json({ success: true, message: "Login successful", user: results[0] });

} else {

res.json({ success: false, message: "Invalid email or password" });

}

});

});

// ✅ SAVE SCORE

app.post("/scores", (req, res) => {

console.log("📩 /scores payload:", req.body);

const { user\_id, category\_id, score, total\_questions } = req.body;

const sql = `

INSERT INTO scores (user\_id, category\_id, score, total\_questions, created\_at)

VALUES (?, ?, ?, ?, NOW())

`;

db.query(sql, [user\_id, category\_id, score, total\_questions], (err, result) => {

if (err) {

console.error("❌ Error inserting score:", err);

return res.status(500).json({ success: false, message: "Failed to save score", error: err });

}

console.log("✅ Inserted score id:", result.insertId);

res.status(201).json({ success: true, id: result.insertId });

});

});

// ✅ GET ALL SCORES (for testing or "Scores" page)

app.get("/scores", (req, res) => {

const sql = `

SELECT s.id, s.score, s.total\_questions, s.created\_at,

u.name AS user\_name,

c.name AS category\_name

FROM scores s

JOIN users1 u ON u.id = s.user\_id

JOIN categories1 c ON c.id = s.category\_id

ORDER BY s.created\_at DESC

`;

db.query(sql, (err, rows) => {

if (err) {

console.error("❌ Error fetching all scores:", err);

return res.status(500).json({ success: false, message: "Failed to fetch scores" });

}

console.log("📤 /scores -> rows:", rows.length);

res.json(rows);

});

});

// ✅ GET USER SCORE HISTORY

app.get('/scores/user/:userId', (req, res) => {

const { userId } = req.params;

const sql = `

SELECT s.id, s.score, s.total\_questions, s.created\_at,

c.name AS category\_name

FROM scores s

JOIN categories1 c ON c.id = s.category\_id

WHERE s.user\_id = ?

ORDER BY s.created\_at DESC

`;

db.query(sql, [userId], (err, rows) => {

if (err) {

console.error("❌ Error fetching scores:", err);

return res.status(500).json({ success: false, message: "Failed to fetch scores" });

}

console.log("📤 /scores/user -> rows:", rows.length);

res.json(rows);

});

});

// ✅ GET HIGHEST SCORES PER CATEGORY

app.get("/highscores", (req, res) => {

const sql = `

SELECT c.name AS category, MAX(s.score) AS highest\_score, MAX(s.total\_questions) AS total\_questions

FROM scores s

JOIN categories1 c ON s.category\_id = c.id

GROUP BY c.id

ORDER BY c.id;

`;

db.query(sql, (err, result) => {

if (err) {

console.error("Error fetching high scores:", err);

return res.status(500).json({ error: "Database error" });

}

res.json(result);

});

});

// ✅ Start server

app.listen(5000, () => {

console.log("Server running on port 5000");

});

Testing

**Unit Testing:** Unit testing where individual program units or object classes are tested. Here by using this testing we have focused on testing the functionality of methods.

**Module Testing:** Where this is the combination of unit program is called module. Here we tested the unit program (5-6 programs) is where the module programs have dependency.

**Sub-system Testing:** Then we combined some module for the Preliminary System Testing in our Project.

**System Testing:** Where it is the combination of two or more sub-system and then it is tested. Here we tested the Entire system as per the requirements.

**Acceptance Testing:** Normally this type of testing is done to verify if system meets the customer specified requirements. After submitting this project to User then they tested it and to determine whether to accept application. It is the system testing performed by the customer(s) to determine whether they should accept the delivery of the system

# 6. Future scope

The **Quiz Website** can be further enhanced with additional features and improvements to increase usability and user engagement. Some possible future enhancements include:

**1. User Authentication & Roles – Enhance the system with different roles like admin, teacher, and student for better management.**

**2. Question Bank Expansion – Add more categories and difficulty levels for a richer quiz experience.**

**3. Leaderboard & Gamification – Implement leaderboards, badges, and rewards to increase user engagement.**

**4. Mobile Application – Develop a mobile app version for easier access on Android/iOS.**

**5. AI-based Recommendations– Suggest quizzes to users based on their past performance and interests.**

**6. Timed & Adaptive Quizzes – Include time-bound and adaptive quizzes where question difficulty changes based on user performance.**

**7. Analytics Dashboard – Provide teachers/admins with detailed analytics of student performance.**

**Conclusion**

The Quiz Website provides an effective and interactive platform for conducting online quizzes. It simplifies the process of creating, managing, and attempting quizzes for both users and administrators. With features like category-based quizzes, score tracking, and user-friendly navigation, the system enhances learning and assessment.

Although the current system fulfills the basic requirements, there is significant scope for improvement and advanced features. By implementing additional functionalities such as leaderboards, analytics, and mobile accessibility, the Quiz Website can become a more robust and comprehensive solution for online assessments.

**Bibliography**

Web Development with React and Node.js

MySQL Documentation

JavaScript: The Definitive Guide

# REFERENCES

## ****References****

* **React Documentation – https://react.dev/**
* **Node.js Documentation – https://nodejs.org/**
* **MySQL Official Site – https://www.mysql.com/**
* **W3Schools – https://www.w3schools.com/**­­­­­­­­