

TRIVIATREK

Project submitted to the
SRM University – AP, Andhra Pradesh
for the partial fulfillment of the requirements to award the degree of
Bachelor of Technology
In
Computer Science and Engineering
School of Engineering and Sciences
Submitted by

P. Pavithra - AP23110011283

CH. Akhila - AP23110011298

A. Asritha - AP23110011317

J. Harnitha Revathi - AP23110010459

For the course Full Stack Development (Course code :- SEC 160)



Department of Computer Science and Engineering

**SRM University-AP
Neerukonda, Mangalagiri, Guntur
Andhra Pradesh – 522240
[December, 2025]**

INTRODUCTION

Welcome to TriviaTrek - a fun and interactive way to learn new things. This platform turns normal quiz questions into an enjoyable learning experience. Instead of reading long notes or using boring methods, TriviaTrek helps you practice and test your knowledge in an exciting way.

Made with a simple and user-friendly design, the platform allows you to choose different topics, answer questions, and instantly see how well you performed. Whether you want to prepare for exams, improve your general knowledge, or just enjoy a quick quiz during your free time, TriviaTrek makes learning easy and engaging.

Built using modern technology, TriviaTrek works smoothly on all devices and provides a clear and enjoyable interface. From science and maths to sports and daily GK, the platform offers something for everyone. With TriviaTrek, learning becomes fun, fast, and meaningful.

Scenario based Intro

Imagine your exams are only a few days away, and you want a quick and fun way to revise. You open **TriviaTrek** and choose a topic like General Science. The quiz starts instantly, and you begin answering short, simple questions that help refresh your memory.

Whenever you select an answer, TriviaTrek immediately tells you if you're right or wrong. This gives you quick feedback and helps you understand which areas you're strong in and which topics need more revision. Once the quiz ends, you receive a clear summary of your score.

But the learning doesn't stop there. TriviaTrek also tracks your progress, showing you how many quizzes you've taken, your overall accuracy, your best-performing topics, and the areas that need more practice. This turns studying into a daily, enjoyable challenge.

Curious to try more? You explore other categories like Maths, Sports, History, and Technology. Each quiz feels like a small, exciting game where you aim to beat your previous score. As your accuracy improves, you start gaining confidence - without even realizing how much you're learning along the way.

This is the power of **TriviaTrek** - a fun, engaging, and smart way to learn, revise, and track your growth anytime you want.

TARGET AUDIENCE

TriviaTrek is built for a wide range of learners who prefer interactive and engaging ways to study. The platform mainly focuses on:

- School and college students who need quick revision tools for exams
- Students preparing for competitive exams who want regular practice
- General knowledge enthusiasts who enjoy testing and improving what they know
- Self-learners who prefer short, quiz-based learning instead of long notes
- Users who want to track their learning progress, including quiz attempts, accuracy scores, and topic-wise strengths

TriviaTrek supports anyone who wants to learn at their own pace while enjoying a clean, simple, and game-like quiz experience.

PROJECT GOALS AND OBJECTIVES

The main goal of the **TriviaTrek** is to provide an engaging, game-like learning platform where students can revise, test their knowledge, and track their progress in a simple and enjoyable way.

Objectives include:

- **Interactive Learning:** Allow students to learn through quick quizzes that make studying more fun and less stressful.
- **Topic Exploration:** Provide well-organized quiz categories so users can easily choose what they want to practice.
- **Instant Feedback:** Show correct and incorrect answers immediately to support faster learning and better understanding.
- **Progress Tracking:** Help users monitor quizzes attempted, accuracy, strengths, and areas needing improvement.
- **User-Friendly Design:** Ensure a clean, simple, and responsive interface that works smoothly on all devices.

Key Features

❖ Quiz Selection System

Users can choose quizzes from a wide range of topics such as Science, Maths, Sports, History, and more.

❖ Instant Answer Feedback

Each question provides immediate right/wrong feedback to help users learn quickly.

❖ Progress Tracking Dashboard

Shows total quizzes attempted, accuracy percentage, best-performing subjects, and areas that need improvement.

❖ Score Summary Report

After every quiz, users receive a detailed summary of correct answers, mistakes, and overall score.

❖ Category-Based Learning

Well-organized quiz categories make it easy for learners to focus on specific subjects.

❖ Gamified Experience

Quiz attempts feel like a game, encouraging users to beat their previous scores and stay motivated.

❖ Complete CRUD Functionality

Supports creating, viewing, updating, and managing:

- Quiz questions
- Categories
- User progress
- Scores
- Attempts history

❖ Modern UI + Responsive Design

Designed with clean, simple layouts that work smoothly on mobile, tablet, and desktop.

PREREQUISITES

The development of **TriviaTrek** requires the following tools and knowledge:

1. Node.js and npm

Node.js is a powerful JavaScript runtime environment that allows you to run JavaScript code on your local machine. It provides a scalable and efficient platform for building web applications.

Install Node.js and npm as they are required to run JavaScript on the server-side.

- **Download:** <https://nodejs.org/en/download/>
- **Installation Instructions:** <https://nodejs.org/en/download/package-manager/>

2. React.js

React.js is a popular JavaScript library for building interactive user interfaces. It enables developers to create reusable UI components, making it easier to build dynamic and responsive web applications.

- **Create a new React app:**

```
npm create vite@latest
- Enter your project name and select preferred frameworks.
```

- **Navigate to project directory:**

```
cd project-name here cd triviatrek
npm install
```

- **Run the React app:**

```
npm run dev
```

- The app can be accessed at: <http://localhost:5173>

3. Basic Web Development Knowledge

- **HTML:** For creating the structure of your web pages.
- **CSS:** For styling, layout, and visual design.
- **JavaScript:** For interactivity, DOM manipulation, and dynamic content.
- **React.js Fundamentals:** Good understanding of components, props, state, and hooks is essential for developing the TriviaTrek frontend.
- **API Knowledge:** Ability to fetch data from APIs (like the Trivia API) and handle responses effectively

4. Code Editor or IDE

- **Recommended:** Visual Studio Code
- These editors provide syntax highlighting, extensions, and a smooth development workflow.
- **Download VS Code:** <https://code.visualstudio.com/download>
- **Other options:** Sublime Text (<https://www.sublimetext.com/download>), WebStorm (<https://www.jetbrains.com/webstorm/download>)

5. Web Browser

- A modern browser such as Google Chrome, Firefox, or Edge for testing and debugging the application.

6. Version Control

- Git for tracking changes and managing code versions.
- Hosting platforms like GitHub for collaboration and repository management.
- **Git Download:** <https://git-scm.com/downloads>
- **GitHub:** <https://github.com/>

7. Basic Knowledge of Responsive Design

- Understanding media queries and flexible layouts to ensure the application works well on desktop, tablet, and mobile devices.

PROJECT STRUCTURE:

```
✓ FULL_STACK -PROJECT
  > node_modules
  ✓ triviatrek
    > dist
    > node_modules
    > public
  ✓ src
    > admin
    > assets
    > components
    > context
    > hooks
    > pages
    # App.css
    ⚡ App.jsx
    # index.css
    ⚡ main.jsx
    ⚙ .env
    ⚡ .gitignore
    ⚡ eslint.config.js
    ↗ index.html
    {} package-lock.json
    {} package.json
    ⓘ README.md
    ⚡ vite.config.js
    {} db.json
    JS fetchTrivia.js
    {} package-lock.json
    {} package.json
```

PROJECT FLOW

Project Demo Link:

https://drive.google.com/file/d/1D36E7Z7OjWC7-SNdk4nucUx4Dt_tCKVo/view?usp=sharing

Use the code in:

https://github.com/Asritha123-coder/Trivia_Trek

Milestone 1: Project Setup and Configuration

Before starting development on **TriviaTrek**, it is important to set up all required tools, libraries, and project structure. This milestone lays a stable foundation for building the interactive quiz platform using React.js, Tailwind CSS, Axios, React Router, and a lightweight backend.

1. Install Required Tools and Software

Essential Tools for Development:

- **React.js** – Library for building dynamic frontend interfaces.
- **Tailwind CSS** – Utility-first CSS framework for responsive design.
- **Axios** – Library for making HTTP requests to APIs.
- **React Router DOM** – For managing navigation between pages.
- **JSON-Server** – Lightweight backend to simulate API and perform CRUD operations.

Node.js & npm:

- Required to run JavaScript locally and manage project dependencies.
- **Download:** <https://nodejs.org/en/download/>
- **Installation Guide:** <https://nodejs.org/en/download/package-manager/>

Installing and Running JSON Server for TriviaTrek

JSON Server is a lightweight tool to simulate a backend API using a simple JSON file. It allows your React frontend to perform CRUD operations without setting up a full backend.

1. Install JSON Server

Install globally using npm:

```
npm install -g json-server
```

2. Create Database File (db.json)

In your project folder, create a file named db.json with a structure like this:

```
{
  "categories": [],
  "subcategories": [],
  "questions": [],
  "quizAttempts": [],
  "users": []
}
```

You can populate it with initial sample data for testing quizzes and users.

3. Run JSON Server

Start the server with the following command:

```
json-server --watch db.json --port 3000
```

- `--watch db.json` → Watches your JSON file for changes
- `--port 3000` → Runs the server on port 3000

Access the API at: **http://localhost:3000/**

Milestone 2: Web Development

Objective:

Develop the frontend of **TriviaTrek** by setting up the React application, configuring routing, and installing all necessary libraries to enable core functionality.

1. Setup React Application

Create React Project:

- Initialize the React application using Vite or Create React App.
- Choose React with JavaScript as the framework.
- Navigate to the project folder and install dependencies:

```
cd triviatrek  
npm install
```

2. Configure Routing:

- Install **React Router DOM** to manage navigation between different pages:

```
npm install react-router-dom
```

- Set up routes for main pages such as:
 - Home / Dashboard
 - Quiz Categories
 - Individual Quiz Page
 - Score Summary / Results
 - Profile / Progress Tracking

3. Install Required Libraries:

- **Axios:** For fetching quiz data from APIs or JSON backend.

```
npm install axios
```

- **Tailwind CSS:** For styling the application and creating a responsive layout.

```
npm install -D tailwindcss postcss autoprefixer  
npx tailwindcss init -p
```

- **React Icons (Optional):** For UI icons.

```
npm install react-icons
```

- **React Toastify (Optional):** For showing notifications and alerts.

```
npm install react-toastify
```

App.js Component

App.jsx sets up all routes for the Triviatrek, shows the navbar for non-admin pages, protects login-required/admin routes, and loads the correct page based on the URL.

```
function App() {  
  return (  
    <BrowserRouter>  
      <AppWrapper />  
    </BrowserRouter>  
  );  
}  
export default App;
```

```

function AppWrapper() {
  const location = useLocation();
  const isAdminRoute = location.pathname.startsWith("/admin");
  return (
    <div>
      {}
      {!isAdminRoute && <Navbar />}
      <Routes>
        {}
        <Route path="/" element={<Navigate to="/home" replace />} />
        <Route path="/home" element={<Home />} />
        <Route path="/login" element={<Login />} />
        <Route path="/signup" element={<Signup />} />
        <Route path="/Category" element={<Category />} />
        <Route path="/Difficulty/:id" element={<Difficulty />} />
        <Route path="/leaderboard" element={<Leaderboard />} />
        <Route path="/about" element={<About />} />
        <Route path="/profile" element={<Profile />} />
        {}
        <Route
          path="/quiz-loading" ...
        >
        </Route>
      />
      <Route
        path="/quiz/:id/:level" ...
      >
      </Route>
    </Routes>
  </div>
);
}

```

Code Description (App Component)

- Defines the main App function, which serves as the entry point of the React application.
- Wraps the entire application inside the **BrowserRouter** component. This enables routing, allowing the app to navigate between different pages without reloading the browser.

- Renders the AppWrapper component inside BrowserRouter. AppWrapper contains all route definitions, navigation rules, protected routes, and page rendering logic.
- Ensures that all pages, components, and navigation functions inside the app can access React Router features like:
 - useNavigate()
 - useLocation()
 - <Routes> and <Route>
- Exports the App component as the default export, so it can be imported and used in the main entry file (typically `main.jsx`).

Login Component (useEffect Navigation Logic)

```
useEffect(() => {
  if (isAuthenticated) {
    if (userRole === "admin") {
      navigate("/admin/dashboard", { replace: true });
    } else {
      navigate("/home", { replace: true });
    }
  }
}, [isAuthenticated, userRole, navigate]);
```

Code Description

- Uses the useEffect hook to automatically run navigation logic whenever the values of isAuthenticated, userRole, or navigate change.
- Checks if the user is authenticated.
This ensures that redirection only happens for logged-in users.
- If the user is logged in and their role is "**admin**", the app automatically navigates them to the admin dashboard using:
`navigate("/admin/dashboard", { replace: true })`
This ensures that admins are immediately taken to the admin section after logging in.
- If the user is authenticated but not an admin, the app redirects them to the home page using:
`navigate("/home", { replace: true })`
This ensures normal users do not access admin pages.

- The replace: true option prevents the previous page (like login) from staying in browser history, improving user flow and preventing back-navigation issues.
- The dependency array [isAuthenticated, userRole, navigate] ensures the effect runs only when authentication status or user role changes, preventing unnecessary redirects.

Quiz Component:

```

const topicLabel = useMemo(() => {
  if (!isCustom) return `Category ${id}`;
  return (
    customMeta?.subcategoryName ||
    customMeta?.name ||
    `Custom ${subCategoryId}`
  );
}, [isCustom, id, customMeta, subCategoryId]);

const selectOption = (opt) => {
  if (timeLeft <= 0) return;
  setUserAnswers((prev) => ({ ...prev, [index]: opt }));
};

const toggleFlag = (i) => setFlagged((p) => ({ ...p, [i]: !p[i] }));
const goToQuestion = (i) => setIndex(i);

const handlePrev = () => { if (index > 0) setIndex((p) => p - 1); };
const handleNext = () => {
  if (index + 1 >= total) {
    setShowSubmitModal(true);
    return;
  }
  setIndex((p) => p + 1);
};

const calculateScore = () => {
  let s = 0;
  for (let i = 0; i < questions.length; i++) {
    const q = questions[i];
    if (!q) continue;
    const u = userAnswers[i];
    if (u && u === q.correct_answer) s++;
  }
  return s;
};

```

Code Description:

- Uses **useMemo** to compute `topicLabel` efficiently.
 - If the quiz is not custom, returns: `Category ${id}`.
 - If custom, uses `subcategoryName`, `name`, or a fallback `Custom ${subCategoryId}`.
 - Recomputes only when `isCustom`, `id`, `customMeta`, or `subCategoryId` change.
- Defines **selectOption**, which updates the selected answer for the current question.
 - Prevents changes if `timeLeft <= 0`.
 - Stores the chosen option in `userAnswers`.
- Defines **toggleFlag**, which marks or unmarks a question as flagged for review.
- Defines **goToQuestion**, which directly jumps to a specific question index.
- Defines `handlePrev`, which moves to the previous question only if `index` is greater than 0.
- Defines **handleNext**, which moves to the next question.
 - If the user reaches the last question, triggers the **submit modal** instead of moving forward.
- Defines **calculateScore**, which calculates the total correct answers.
 - Loops over each question.
 - Compares the user's answer with the correct answer.
 - Increments score for every correct match.
 - Returns the final score.

Results Component:

```
import { useLocation } from "react-router-dom";
export default function Results() {
  const { state } = useLocation();
  const { questions, userAnswers } = state;
  const getScore = () => {
    let score = 0;
    questions.forEach((q, idx) => {
      if (userAnswers[idx] === q.correct_answer) score++;
    });
    return score;
};
```

```

return (
  <div className="p-6">
    <h1 className="text-3xl font-bold mb-4">Quiz Results</h1>
    <h2 className="text-xl mb-6">
      Score: {getScore()} / {questions.length}
    </h2>

    {questions.map((q, index) => {
      const userAns = userAnswers[index];
      const correctAns = q.correct_answer;
      const isCorrect = userAns === correctAns;

      return (
        <div key={index} className="mb-6 p-4 border rounded-xl bg-gray-50">
          <p className="font-semibold mb-2">
            {index + 1}. {q.question}
          </p>

          <p
            className={`${p-2 rounded mb-2
              ${isCorrect ? "bg-green-200" : "bg-red-200"}}`}
          >
            <strong>Your Answer:</strong> {userAns}
          </p>

          {!isCorrect && (
            <p className="p-2 rounded bg-green-100 border border-green-300">
              <strong>Correct Answer:</strong> {correctAns}
            </p>
          )}
        </div>
      );
    )})
  </div>
);
}

```

Code Description:-

- Imports **useLocation** from *react-router-dom* to access state passed through route navigation.
- Defines the default React component **Results**, which displays the quiz results page.
- Inside the component, extracts the **state** object from **useLocation()**, and then destructures **questions** and **userAnswers** from the state. These contain all quiz questions and the user's selected answers.
- Defines a helper function **getScore**, which:
 - Initializes a local variable **score** to 0.
 - Iterates through each question using **forEach**.
 - Compares each **userAnswers[idx]** with the question's **correct_answer**.
 - Increments the score for every correct match.
 - Returns the final score.

- Renders the results page inside a main <div> with padding applied through the p-6 class.
- Displays:
 - A large heading “**Quiz Results**” using Tailwind classes.
 - A subheading that shows the **user's score out of the total number of questions** using the getScore() function.
- Maps over the **questions** array to render detailed results for each question:
 - Retrieves the user's answer for the index.
 - Retrieves the correct answer for the question.
 - Determines whether the user's answer is correct by comparing both.
 - Returns JSX likely showing:
 - The question text,
 - The user's selected answer,
 - The correct answer,
 - Visual indicators for correct/incorrect responses.
- Wraps all question result items inside the parent <div> to properly structure the layout.
- Exports the **Results** component as the default export so it can be used by other pages in the application.

Add Category Component:

```
import React, { useState } from "react";
import axios from "axios";
import { CheckCircle, XCircle, Plus, Loader, Sparkles } from "lucide-react";
import Toast from "../../components/Toast";

const AddCategory = () => {
  const [name, setName] = useState("");
  const [message, setMessage] = useState("");
  const [messageType, setMessageType] = useState("");
  const [loading, setLoading] = useState(false);
  const [toast, setToast] = useState({ isOpen: false, message: "", type: "success" });

  const handleSubmit = async (e) => {
    e.preventDefault();
    setMessage("");
    setMessageType("");
  
```

```

if (!name.trim()) {
  setMessage("Category name is required");
  setMessageType("error");
  return;
}

if (name.trim().length < 3) {
  setMessage("Category name must be at least 3 characters long");
  setMessageType("error");
  return;
}

 setLoading(true);

try {
  await axios.post("http://localhost:3000/categories", { name: name.trim() });
  setToast({ isOpen: true, message: `Category "${name.trim()}" successfully added!`, type: "success" });
  setName("");
  setMessage("");
  setMessageType("");
} catch (error) {
  console.error(error);
  setMessage(error.response?.data?.message || "Error adding category. Please try again.");
  setMessageType("error");
} finally {
  setLoading(false);
}
};

```

Code Description:-

- Imports React's useState, axios for API requests, icons from **lucide-react**, and a custom **Toast** component.
- Defines the **AddCategory** component, used to create a new category.
- Manages several state variables:
 - name for the input field
 - message and messageType for validation feedback
 - loading for submit-button state
 - toast for showing success notifications
- The **handleSubmit** function:

- Prevents default form submission.
- Validates the category name (required and minimum length).
- Shows validation messages if invalid.
- Sends a POST request to **http://localhost:3000/categories** with the category name.
- On success, shows a success toast and resets the input.
- On error, displays an error message from the server or a fallback message.
- Stops the loading indicator after the request finishes.
- Exports the AddCategory component for use in the application.

Admin Dashboard Component:

```
const AdminDashboard = () => {
  const [categories, setCategories] = useState([]);
  const [subcategories, setSubcategories] = useState([]);
  const [questionSets, setQuestionSets] = useState([]);
  const [attempts, setAttempts] = useState([]);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState("");

  useEffect(() => {
    let mounted = true;

    const fetchAll = async () => {
      try {
        setLoading(true);
        const [catRes, subRes, questionRes, attemptRes] = await Promise.all([
          axios.get(`${API_URL}/categories`),
          axios.get(`${API_URL}/subcategories`),
          axios.get(`${API_URL}/questions`),
          axios.get(`${API_URL}/quizAttempts`),
        ]);

        if (!mounted) return;
        setCategories(catRes.data || []);
        setSubcategories(subRes.data || []);
        setQuestionSets(questionRes.data || []);
        setAttempts(attemptRes.data || []);
        setError("");
      } catch (err) {
        console.error("Dashboard fetch failed", err);
        if (mounted) {
      
```

```

        setError("Unable to load dashboard data. Is json-server running?");
    }
} finally {
    if (mounted) setLoading(false);
}
};

fetchAll();

return () => {
    mounted = false;
};
}, []);
};

const totalCategories = categories.length;
const totalQuestions = useMemo(
() =>
questionSets.reduce(
    (sum, set) => sum + (Array.isArray(set.questions) ?
set.questions.length : 0),
    0
),
[questionSets]
);
const totalQuizzesTaken = attempts.length;
const totalActiveUsers = useMemo(
() => new Set(attempts.map((att) => att.userEmail || att.userName)).size,
[attempts]
);
const averageDuration = useMemo(() => {
    if (!attempts.length) return 0;
    const total = attempts.reduce((sum, att) => sum + (att.durationSeconds ||
0), 0);
    return Math.round(total / attempts.length);
}, [attempts]);

const recentActivity = useMemo(() => {
    return [...attempts]
        .sort((a, b) => new Date(b.date) - new Date(a.date))
        .slice(0, 5);
}, [attempts]);

const levelDistribution = useMemo(() => {
    const base = { easy: 0, medium: 0, hard: 0 };
    attempts.forEach((att) => {
        const key = att.level?.toLowerCase();
        if (!key) return;
        base[key] = (base[key] || 0) + 1;
    });
}, [attempts]);

```

```
});  
return base;  
, [attempts]);
```

Code Description:

- The **AdminDashboard** component manages and displays admin data such as categories, subcategories, question sets, and quiz attempts.
- Uses useState to store fetched data and handle loading/error states.
- A useEffect runs on mount, fetching all dashboard data from multiple API endpoints using Promise.all.
Successfully fetched data updates the state, and errors trigger an error message.
- Computes key dashboard stats using useMemo:
 - **totalCategories** - number of categories.
 - **TotalQuestions** - total questions across all question sets.
 - **totalQuizzesTaken** - total quiz attempts.
 - **totalActiveUsers** - unique users based on email/name.
 - **averageDuration** - average time users spent on quizzes.
 - **recentActivity** - 5 most recent attempts sorted by date.
 - **levelDistribution** - number of attempts categorized by difficulty (easy, medium, hard).

Manage Questions Component:

```
const handleDeleteClick = (id) => {  
  setDeleteTargetId(id);  
  setShowDeleteModal(true);  
};  
  
const deleteQuestionSet = async () => {  
  if (!deleteTargetId) return;  
  try {  
    await axios.delete(`.${API_URL}/questions/${deleteTargetId}`);  
    setQuestionSets((prev) => prev.filter((set) => set.id !==  
deleteTargetId));  
  } catch (error) {  
    console.error(error.message);  
  }  
};
```

```

        setToast({ isOpen: true, message: "Question set successfully deleted.", type: "success" });
        setShowDeleteModal(false);
        setDeleteTargetId(null);
    } catch (err) {
        console.error("Failed to delete question set", err);
        setToast({ isOpen: true, message: "Failed to delete question set. Please try again.", type: "error" });
    }
};

const categoryLookup = useMemo(() => {
    const map = {};
    categories.forEach((cat) => (map[cat.id] = cat.name));
    return map;
}, [categories]);

const subcategoryLookup = useMemo(() => {
    const map = {};
    subcategories.forEach((sub) => (map[sub.id] = sub.name));
    return map;
}, [subcategories]);

```

Code Description

- **handleDeleteClick**
 - Sets the ID of the question set to delete.
 - Opens the delete confirmation modal.
- **deleteQuestionSet**
 - Sends a DELETE request to remove the selected question set from the server.
 - On success, removes it from local state and shows a success toast.
 - On failure, logs the error and shows an error toast.
 - Closes the modal and resets the selected ID after completion.
- **categoryLookup** (memoized)
 - Creates a fast lookup map of category IDs to category names for display purposes.
- **subcategoryLookup** (memoized)
 - Creates a lookup map of subcategory IDs to subcategory names

Manage Categories Component:

```
const loadCategories = async () => {
  try {
    setLoading(true);
    const res = await axios.get(` ${API_URL}/categories`);
    setCategories(res.data || []);
    setError("");
  } catch (err) {
    console.error("Failed to load categories", err);
    setError("Unable to load categories. Please check your JSON server.");
  } finally {
    setLoading(false);
  }
};

useEffect(() => {
  loadCategories();
}, []);
```

```
const saveCategory = async () => {
  if (!nameInput.trim()) {
    setStatus("Name cannot be empty.");
    return;
  }
  try {
    await axios.patch(` ${API_URL}/categories/${editingId}` , {
      name: nameInput.trim(),
    });
    setCategories((prev) =>
      prev.map((cat) =>
        cat.id === editingId ? { ...cat, name: nameInput.trim() } : cat
      )
    );
    setToast({ isOpen: true, message: `Category "${nameInput.trim()}" successfully updated.` , type: "success" });
    cancelEdit();
  } catch (err) {
    console.error("Update failed", err);
    setStatus("Failed to update. Try again.");
  }
};
```

Code Description

- **loadCategories**

- Fetches all categories from the API.
- Sets a loading state while retrieving data.
- Updates the category list on success and clears any existing errors.
- Shows an error message if the request fails.
- A useEffect runs once on component mount to load categories by calling loadCategories().
- **saveCategory**
 - Validates that the updated category name is not empty.
 - Sends a PATCH request to update the selected category.
 - Updates the category list in state without refetching from the server.
 - Shows a success toast and exits edit mode after a successful update.
 - Displays an error message if the update fails.

Category Component Component:

```
useEffect(() => {
  const loadData = async () => {
    try {
      setLoading(true);
      const [remote, localCats, localSubs] = await Promise.allSettled([
        axios.get("https://opentdb.com/api_category.php"),
        axios.get("http://localhost:3000/categories"),
        axios.get("http://localhost:3000/subcategories"),
      ]);

      if (remote.status === "fulfilled") {
        setCategories(remote.value.data?.trivia_categories || []);
      }

      if (localCats.status === "fulfilled") {
        setLocalCategories(localCats.value.data || []);
      }

      if (localSubs.status === "fulfilled") {
        setLocalSubcategories(localSubs.value.data || []);
      }
    } catch (err) {
      console.error("Failed to load categories:", err);
    } finally {
      setLoading(false);
    }
  }
});
```

```
    };
    loadData();
}, []);
```

Code Description:

- A useEffect runs once on component mount to load category and subcategory data from both a remote API and a local JSON server.
- Inside the effect, **loadData**:
 - Starts a loading state.
 - Uses Promise.allSettled to fetch:
 - Remote trivia categories from the OpenTDB API,
 - Local categories,
 - Local subcategories.
 - Saves each result to its corresponding state only if the request was successful.
- Catches and logs any errors during the fetch process.
- Ends by turning off the loading state.

Difficulty Component:

```
import React, { useEffect, useMemo, useState } from "react";
import axios from "axios";
import { useLocation, useNavigate, useParams } from "react-router-dom";
import { motion } from "framer-motion";
import { ArrowLeft, Zap, Target, Award } from "lucide-react";
import { useProtectedNavigation } from "../hooks/useProtectedNavigation";

const defaultLevels = ["easy", "medium", "hard"];

const Difficulty = () => {
  const navigate = useNavigate();
  const location = useLocation();
  const { id } = useParams();
  const { navigateTo, Modal: ProtectedModal } = useProtectedNavigation();

  const isCustom = id?.startsWith("local-");
  const subCategoryId = isCustom ? id.replace("local-", "") : null;

  const [levels, setLevels] = useState(isCustom ? [] : defaultLevels);
```

```

const [loading, setLoading] = useState(isCustom);
const [error, setError] = useState("");
const [subInfo, setSubInfo] = useState(
  location.state?.customMeta || null
);

useEffect(() => {
  let active = true;

  if (!isCustom) {
    setLevels(defaultLevels);
    setLoading(false);
    setError("");
    return undefined;
  }
  const fetchCustom = async () => {
    setLoading(true);
    setError("");
    try {
      const [subRes, questionsRes] = await Promise.all([
        axios.get(
          `http://localhost:3000/subcategories?id=${subCategoryId}`
        ),
        axios.get(
          `http://localhost:3000/questions?subCategoryId=${subCategoryId}`
        ),
      ]);
    
```

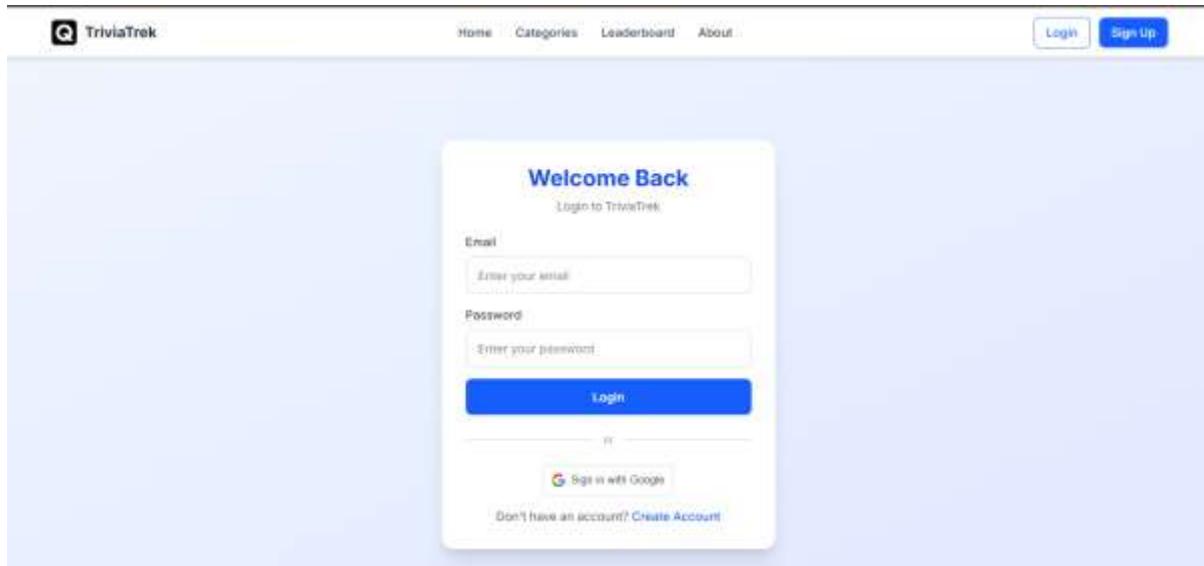
Code Description:

- Imports React hooks, axios, router utilities, framer-motion, and icons, as well as a custom useProtectedNavigation hook.
- Defines the **Difficulty** component, which displays available difficulty levels for a chosen quiz category or subcategory.
- Retrieves routing data using:
 - useNavigate for navigation,
 - useLocation for receiving state,
 - useParams to read the URL id.
- Determines if the quiz is custom by checking if the id starts with "local-".
Extracts the subcategory ID for custom quizzes.
- Initializes component state:
 - levels - defaults to predefined levels (easy, medium, hard) unless custom.
 - loading - true for custom subcategories until their data loads.

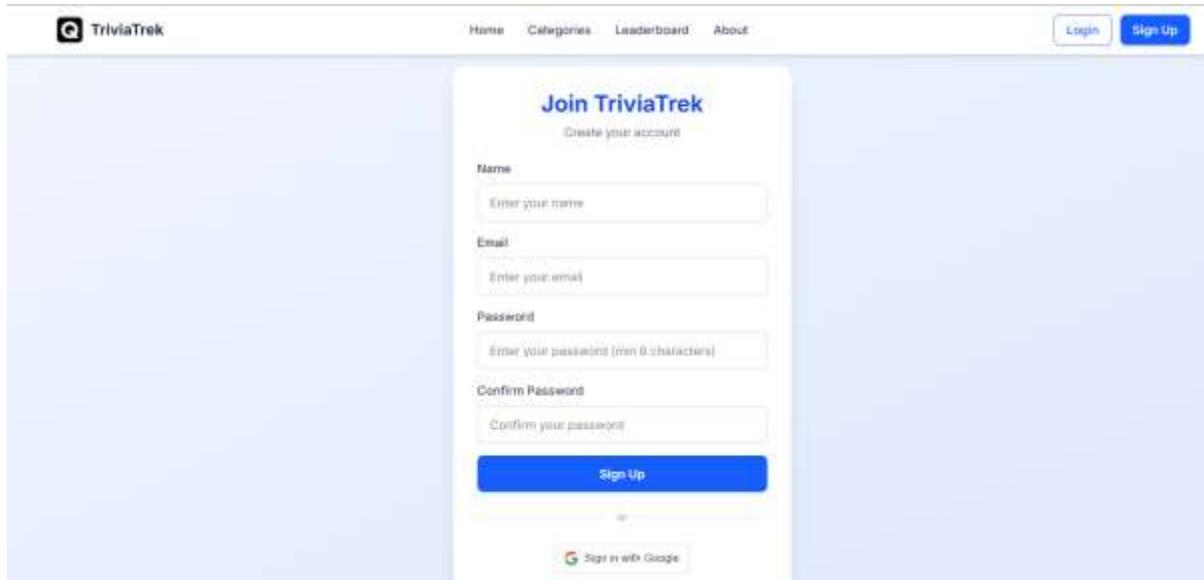
- error - stores loading errors.
- subInfo - subcategory metadata from router state if available.
- A useEffect runs when the component mounts:
 - If not a custom quiz, it sets default levels and stops loading.
 - If it *is* custom, it fetches:
 - Subcategory details,
 - Questions belonging to that subcategory.
 - Uses Promise.all to load both in parallel.
 - Updates state with the results and handles errors and loading states.
 - A cleanup flag (active) prevents state updates after unmounting.

OUTPUT SCREENSHOTS

Login Page:

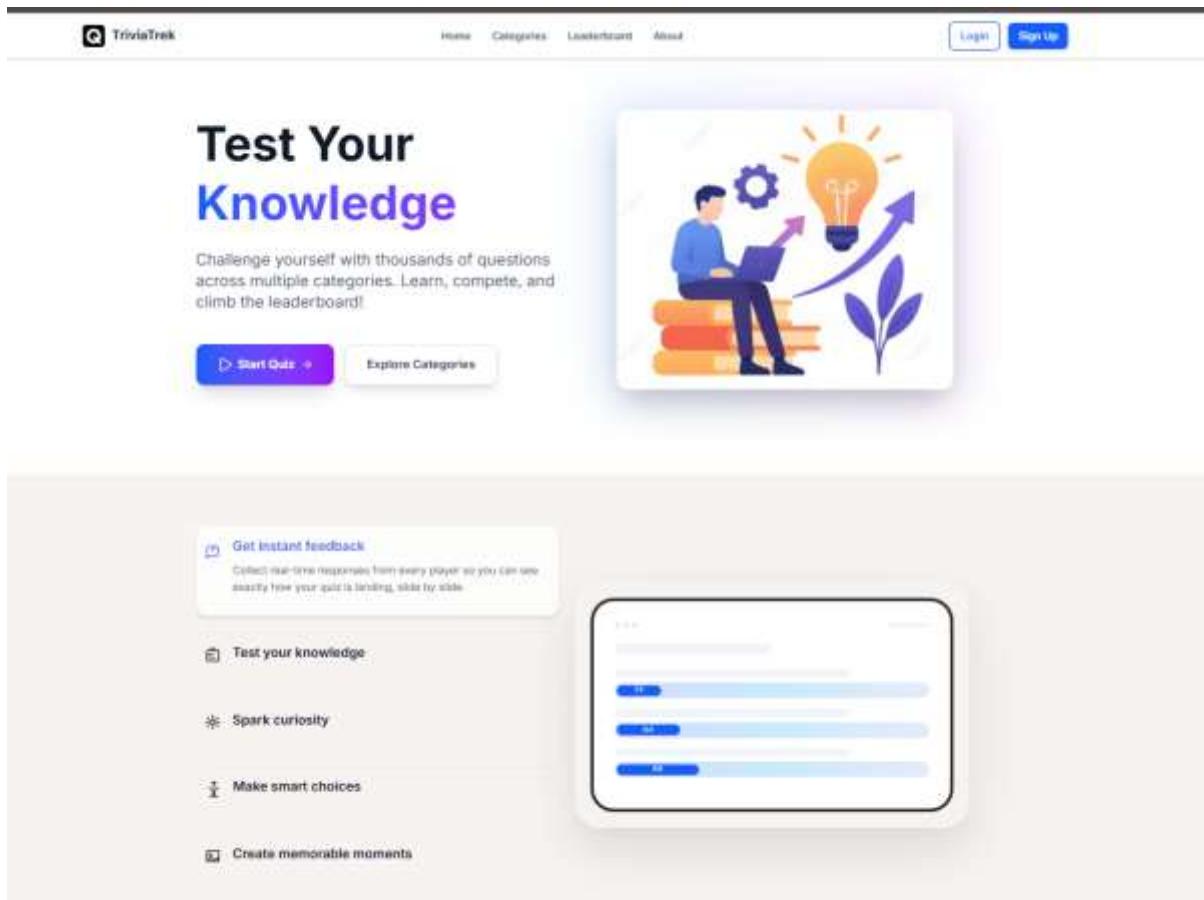


SignUp page:



The screenshot shows the SignUp page for the TriviaTrek website. At the top, there is a navigation bar with links for Home, Categories, Leaderboard, and About. On the right side of the navigation bar are two buttons: 'Login' and 'Sign Up'. The main content area is titled 'Join TriviaTrek' and has a sub-instruction 'Create your account'. It contains four input fields: 'Name' (placeholder: 'Enter your name'), 'Email' (placeholder: 'Enter your email'), 'Password' (placeholder: 'Enter your password (min 8 characters)'), and 'Confirm Password' (placeholder: 'Confirm your password'). Below these fields is a large blue 'Sign Up' button. At the bottom of the form is a small link for 'Sign in with Google'.

Home Page:



The screenshot shows the Home page for the TriviaTrek website. At the top, there is a navigation bar with links for Home, Categories, Leaderboard, and About. On the right side of the navigation bar are two buttons: 'Login' and 'Sign Up'. The main content area features a large, bold title 'Test Your Knowledge' in black and blue text. Below the title is a sub-instruction: 'Challenge yourself with thousands of questions across multiple categories. Learn, compete, and climb the leaderboard!'. To the right of the text is a colorful illustration of a person sitting at a laptop, with a lightbulb above them symbolizing ideas. Below the illustration are two buttons: 'Start Quiz' (purple) and 'Explore Categories' (white). The bottom section of the page contains several promotional cards with icons and text: 'Get instant feedback' (Collect real-time responses from every player so you can see exactly how your quiz is landing, slide by slide), 'Test your knowledge' (with a list icon), 'Spark curiosity' (with a star icon), 'Make smart choices' (with a document icon), and 'Create memorable moments' (with a camera icon). To the right of these cards is a large, rounded rectangular box containing three horizontal progress bars, each with a blue progress indicator.

Categories Page:



Home Categories Leaderboard About

[Login](#) [Sign Up](#)

Explore

Q Search categories...

General Knowledge



General Knowledge
Tap to view quizzes

Entertainment

[View All](#)



Entertainment: Books
Tap to view quizzes



Entertainment: Film
Tap to view quizzes



Entertainment: Music
Tap to view quizzes

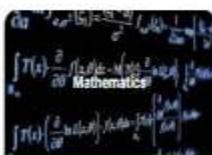
Science



Science & Nature
Tap to view quizzes



Science: Computers
Tap to view quizzes



Science: Mathematics
Tap to view quizzes

LeaderBoard Page:

The screenshot shows the TriviaTrek Leaderboard page. At the top, there is a navigation bar with links for Home, Categories, Leaderboard, and About, along with Login and Sign Up buttons. Below the navigation is a large title "Leaderboard" with a trophy icon. A sub-header says "Compete with players worldwide and see how you rank!"

Below the title are two filter sections: "Filter by Difficulty" (set to All Levels) and "Filter by Category" (set to All Categories). Three player cards are displayed prominently:

- Leo Carter** (L icon) - Best Accuracy: 90.0% (9 attempts)
- admin_user** (A icon) - Best Accuracy: 100.0% (27 attempts)
- Aisha Khan** (A icon) - Best Accuracy: 70.0% (1 attempts)

Below these cards is a section titled "Full Rankings" which states "8 players ranked". The table has columns: RANK, PLAYERS, BEST ACCURACY, BEST SCORE, AVG ACCURACY, ATTEMPTS, and AVG TIME.

| RANK | PLAYERS | BEST ACCURACY | BEST SCORE | AVG ACCURACY | ATTEMPTS | AVG TIME |
|------|----------------------------------|---------------|------------|--------------|----------|----------|
| 1 | admin_user admin@example.com | 100.0% | 4/7 | 8.0% | 27 | 0 156s |
| 2 | Leo Carter leo@example.com | 90.0% | 9/10 | 90.0% | 1 | 0 390s |
| 3 | Aisha Khan aisha@example.com | 70.0% | 7/10 | 70.0% | 1 | 0 430s |
| 4 | Priya Verma priya@example.com | 60.0% | 6/10 | 60.0% | 1 | 0 520s |
| 5 | john john@example.com | 40.0% | 4/10 | 35.0% | 2 | 0 28s |

About Page:

The screenshot shows the 'Key Features' section of the TriviaTrek website. At the top, there is a navigation bar with links for Home, Categories, Leaderboard, and About, along with Login and Sign Up buttons. Below the navigation is a decorative yellow starburst graphic. The main title 'Key Features' is centered in a large, bold, black font. A subtitle 'Everything you need for an amazing quiz experience' follows. The features are presented in a grid of six items, each with an icon and a brief description:

- Diverse Categories**: Explore thousands of questions across multiple categories including Science, History, Entertainment, and more.
- Multiple Difficulty Levels**: Challenge yourself with Easy, Medium, or Hard difficulty levels tailored to your skill level.
- Track Your Progress**: Monitor your performance, view detailed analytics, and see how you improve over time.
- Compete Globally**: Join the leaderboard and compete with players from around the world to see who ranks highest.
- Real-Time Quizzes**: Take timed quizzes with instant feedback and detailed explanations for each question.
- Learn & Improve**: Review your mistakes, understand correct answers, and expand your knowledge with every quiz.

Profile Page:

The screenshot shows the TriviaTrek profile page for a user named "admin_user". At the top, there's a navigation bar with links for Home, Categories, Leaderboard, and About. A blue header bar indicates the user is "admin_user". Below the header, the user's profile card displays a blue circular icon with a white letter "A", the username "admin_user", the email "admin@example.com", and the role "Administrator". There's also a "Edit Profile" button. The main section features four performance metrics in cards: "Total Attempts" (27), "Average Accuracy" (8.0%), "Best Accuracy" (100.0%), and "Avg. Time" (0m). Below these are two larger sections: "Performance Overview" and "Recent Activity". The "Performance Overview" card shows a progress bar for "Best Score" (8 points), "Total Questions Answered" (100), and "Total Time Spent" (7 minutes). The "Recent Activity" card lists three recent quizzes taken by the user: "DSA + Arrays" (Medium, 0/1, 0.0%), "18" (Medium, 1/10, 10.0%), and another Medium quiz (0/1, 0.0%).

Admin Panel DashBoard:

The screenshot shows the Quiz Command Center Admin Panel Dashboard. On the left, a sidebar menu includes options like Dashboard, Add Category, Add Sub-Category, Add Questions, Manage Categories, Manage Sub-Categories, Manage Questions, and Analytics. It also shows the current user as "A- admin_user" and links to "Go to User View" and "Logout". The main dashboard has a header "Quiz Command Center" with a subtitle "Stay on top of everything happening inside your trivia universe — categories, question banks, and quiz attempts; update in real time from 'db.json'". Below the header are several summary cards: "Total Categories" (2), "Quizzes Available" (4), "Quizzes Taken" (38), and "Recent Activity" (listing 6 recent user interactions). The "Level Distribution" chart shows the proportion of Easy, Medium, and Hard quizzes taken. At the bottom, there are links for "Question Coverage" and "Top Performers".

Add Category:

✳️ Add New Category

Create a new category to organize your quizzes and content

Category Name

e.g., Science, History, Sports, Technology

Enter a descriptive name for your category (minimum 3 characters)

+ Add Category

Clear

💡 Tip: Categories help organize your quizzes. Use clear, descriptive names that users will easily understand. Once created, you can add subcategories and questions to each category.

Add SubCategory :

✳️ Add New Subcategory

Create a subcategory under an existing category to organize your quizzes

Subcategory Name

e.g., World History, Quantum Physics, Modern Art

✳️ Live Preview

Image Preview

Parent Category

Select a Category

Image URL

<https://example.com/image.png>

Subcategory Name

No category selected

Card Background Color



#F0F9FF

Choose a color for the subcategory card background

+ Add Subcategory

Clear

Add Questions:

⭐ Add Questions

Create quiz questions by selecting a category, subcategory, and difficulty level

| Category | Subcategory | Difficulty Level | Number of Questions |
|------------------|-------------|------------------|---------------------|
| backend-beginner | HTML | Easy | 1 |

Question 1

Question Text

Enter your question here...

Option A Option B

Option C Option D

Correct Answer

Select Correct Answer

+ Submit 1 Question

Manage Category

| MANAGE CATEGORIES | | Search category... |
|--|------------------|---|
| Curate Your Library Update or remove custom categories stored in `lib.json`. | | |
| CATEGORY ID e05e | backend-beginner | Edit Delete |
| CATEGORY ID 6a55 | DSA | Edit Delete |

Manage SubCategories:

MANAGE SUBCATEGORIES

Organize Nested Topics

Update metadata (name, category, imagery, color) for every custom subcategory.

Search by name or category...

The screenshot shows a list of subcategories under a parent category. The first item is 'HTML' with an icon of a monitor displaying code, a category of 'Category: —', and edit/delete buttons. To its right, there's a preview box with ID: 81dc and Color: #F0F8FF. The second item is 'CSS' with an icon of a smartphone displaying code, a category of 'Category: —', and edit/delete buttons. To its right, there's a preview box with ID: 6fb6e and Color: #318BBD. The third item is 'sql-beginner' with an icon of a smartphone displaying code, a category of 'Category: backend-beginner', and edit/delete buttons. To its right, there's a preview box with ID: e02b and Color: #6F28A4.

| Name | Category | ID | Color |
|--------------|----------------------------|-------|---------|
| HTML | Category: — | 81dc | #F0F8FF |
| CSS | Category: — | 6fb6e | #318BBD |
| sql-beginner | Category: backend-beginner | e02b | #6F28A4 |

Manage Questions:

MANAGE QUESTION BANKS

Fine-tune Every Quiz

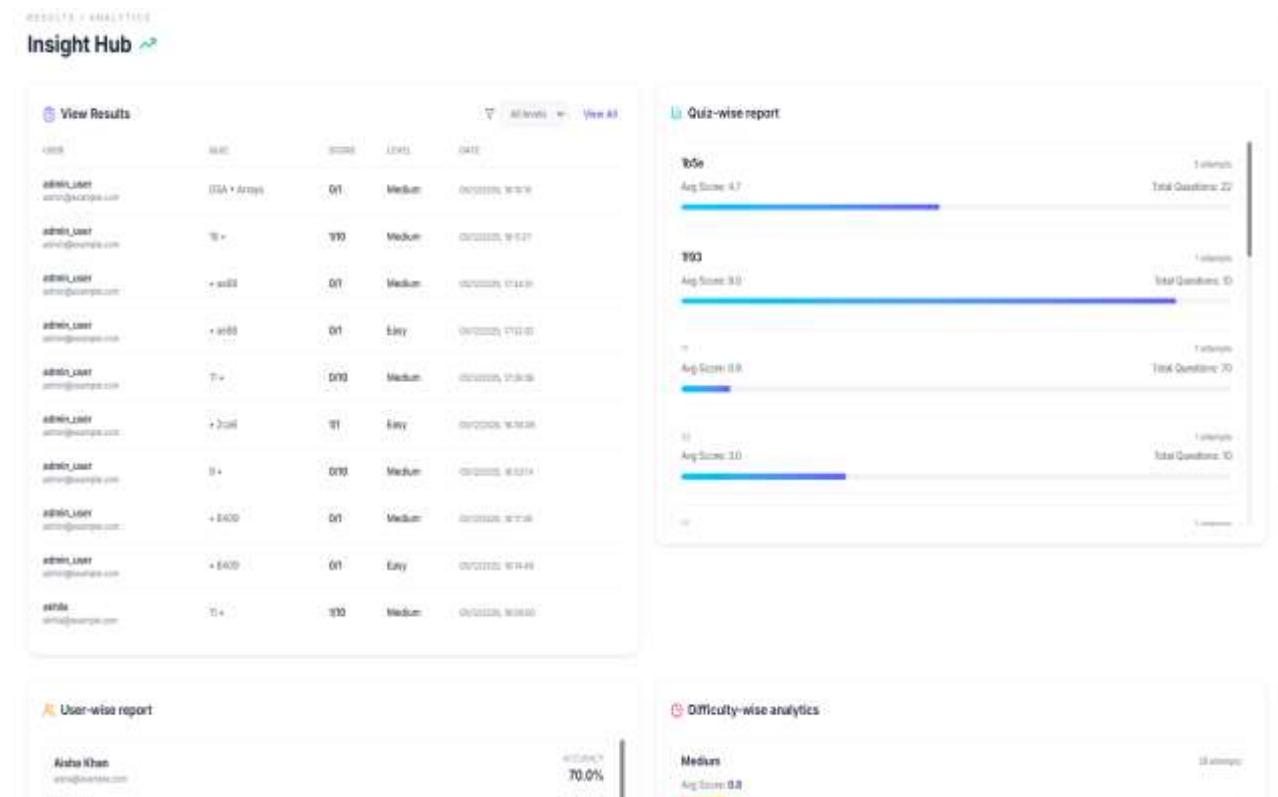
Edit question sets powering your custom quizzes. All changes sync to '10pm' via the JLDN server.

All Categories All Subcategories All Levels

The screenshot shows a list of question sets. The first item is 'HTML' with an icon of a document, a level of 'MEDIUM' (green), and 1 MCQs. The second item is 'CSS' with an icon of a document, a level of 'EASY' (green), and 1 MCQs. The third item is 'backend-beginner • sql-beginner' with an icon of a document, a level of 'EASY' (green), and 1 MCQs. The fourth item is 'DSA • Arrays' with an icon of a document, a level of 'MEDIUM' (green), and 1 MCQs. Each item has an 'Edit' button (green) and a 'Delete' button (red).

| Question Set | Level | Count |
|---------------------------------|--------|--------|
| HTML | MEDIUM | 1 MCQs |
| CSS | EASY | 1 MCQs |
| backend-beginner • sql-beginner | EASY | 1 MCQs |
| DSA • Arrays | MEDIUM | 1 MCQs |

Analytics Page:



Project Demo Link:

https://drive.google.com/file/d/1D36E7Z7OiWC7-SNd4nucUx4Dt_tCKVo/view?usp=sharing

Project Code Explanation Link:

<https://drive.google.com/file/d/1wfCipTu4FPg6O5xZymQ2TQWxF1HYPRL8/view?usp=sharing>