

Trivia Trek

Introduction:-

Project Description

Trivia Trek is an interactive quiz application designed to make learning engaging and accessible for users of all age groups. It provides multiple-choice questions across a wide range of topics and allows users to track their performance through scores, timers, and detailed result summaries. Users can review incorrect answers, understand explanations, and continuously improve their knowledge. The application focuses on delivering an educational yet enjoyable experience through a simple and intuitive interface.

Introduction

Trivia Trek introduces a modern approach to learning through interactive quizzes. It is built with a user-centric design that combines simplicity with efficient functionality. The platform provides topic-wise quizzes, instant scoring, time-bound challenges, and performance analytics to enhance the overall learning process. Trivia Trek aims to make knowledge exploration more engaging by transforming traditional learning into an interactive experience.

Core System Overview

Trivia Trek is designed with a responsive and efficient architecture that ensures smooth performance across devices. Each quiz presents a set of multiple-choice questions along with a timer, scoring logic, and clear feedback for every attempt. At the end of each quiz, users receive a detailed summary of their performance, including the total score, correct and incorrect responses, and explanations for the questions they answered incorrectly.

The application emphasizes ease of use, clear navigation, and quick interaction, making it suitable for both quick practice sessions and extended learning.

Scenario-Based Introduction

Consider a student preparing for competitive exams or simply wanting to test their general knowledge. They open Trivia Trek and are greeted with a clean and simple interface displaying various quiz categories.

The student selects a category, such as Science and Technology. A timer starts, and the quiz begins. They answer each question while keeping track of time and score. After completing the quiz, the student receives a detailed results page that includes:

- Total number of questions attempted
- Correct and incorrect responses
- Time taken
- Explanations for incorrect answers
- Suggestions for improvement

This process encourages the student to revisit topics, retake quizzes, and strengthen their understanding through active practice.

Target Audience

Trivia Trek is designed for a broad user base, including:

Students: Individuals preparing for examinations or wanting to strengthen their knowledge across subjects.

Knowledge Enthusiasts: Users who enjoy testing and expanding their general awareness.

Educators: Teachers who want to conduct quick assessments or introduce interactive learning.

General Users: Anyone seeking to learn or engage in quizzes for entertainment and self-improvement.

Project Goals and Objectives

The primary aim of Trivia Trek is to provide an accessible and engaging quiz-based learning platform. Its objectives include:

User-Friendly Interface: Develop a clean and intuitive interface that enables smooth navigation, easy quiz selection, and clear presentation of results.

Effective Quiz Management: Incorporate structured quizzes, timers, scoring systems, and a detailed results summary to support learning and progress tracking.

Learning Enhancement: Provide explanations for incorrect answers to help users understand concepts and improve knowledge retention.

Engaging User Experience: Use interactive elements such as timed questions and performance analytics to maintain user motivation.

Scalable Modern Technology Stack: Build the application using modern web technologies to ensure fast loading, responsive design, and scalability for future features such as leaderboards, user accounts, and advanced analytics.

Key Features:-

➤ **Interactive Quiz System:**

A real-time quiz interface that displays multiple-choice questions, tracks the timer, and updates scores instantly as the user progresses.

➤ **Category-Based Quiz Modules:**

Separate quiz sections for different topics such as General Knowledge, Science, Technology, History, and more, allowing users to choose and attempt quizzes based on their interests.

➤ **Performance Tracking and Review:**

A detailed performance summary is provided at the end of each quiz, including correct answers, incorrect answers, explanations, total score, and time taken.

➤ **Responsive and Cross-Device Compatibility:**

The application is designed to function smoothly on different devices such as mobiles, tablets, and desktops, ensuring a consistent and accessible quiz experience.

PRE-REQUISITES:-

Here are the key prerequisites for developing a frontend application using React.js:

➤ **Node.js and npm:**

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

Install Node.js and npm on your development machine, 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/>

➤ **React.js:**

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

Install React.js, a JavaScript library for building user interfaces.

- Create a new React app:

```
npm create vite@latest
```

Enter and then type project-name and select preferred frameworks and then enter

- Navigate to the project directory:

```
cd project-name
```

```
npm install
```

- Running the React App:

With the React app created, you can now start the development server and see your React application in action.

- Start the development server:

```
npm run dev
```

This command launches the development server, and you can access your React app at <http://localhost:5173> in your web browser.

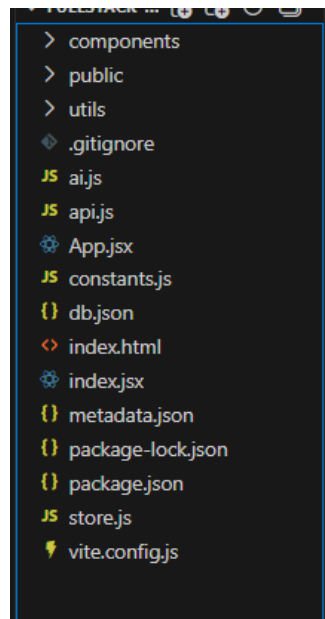
➤ **HTML, CSS, and JavaScript:** Basic knowledge of HTML for creating the structure of your app, CSS for styling, and JavaScript for client-side interactivity is essential.

➤ **Version Control:** Use Git for version control, enabling collaboration and tracking changes throughout the development process. Platforms like GitHub or Bitbucket can host your repository.

- Git: Download and installation instructions can be found at: <https://git-scm.com/downloads>

- **Development Environment:** Choose a code editor or Integrated Development Environment (IDE) that suits your preferences, such as Visual Studio Code, Sublime Text, or WebStorm.
- Visual Studio Code: Download from <https://code.visualstudio.com/download>
 - Sublime Text: Download from <https://www.sublimetext.com/download>
 - WebStorm: Download from <https://www.jetbrains.com/webstorm/download>

Project structure:



The structure of the Trivia Trek application is organized to ensure clarity, modularity, and ease of maintenance. Since the project is developed using **React** along with a **JSON database**, the files and folders are arranged to separate UI components, services, and data-handling layers efficiently. A well-planned directory structure allows developers to collaborate effectively and extend the application with minimal complexity.

src/App.js, App.css, App.jsx (App Component)

These files represent the main App component, which functions as the root of the React application. The App component is responsible for setting up the overall layout, initializing global states, and defining the routing structure. It serves as the entry point from where different pages—such as quiz categories, quiz interface, score summary, and review pages—are rendered.

PROJECT FLOW:-

Project demo:

Before starting to work on this project, let's see the demo.

Demo link: <https://drive.google.com/file/d/1b65uh24M-1Z1aiwgkboMs5bhrDPDoIa/view>

Use the code in: <https://github.com/DevLikhith5?tab=repositories>

Milestone 1: Project Setup and Configuration

Installation of Required Tools and Software

To begin development of the Trivia Trek application, the project folder was prepared and all essential tools and libraries were installed. The application is built using React, along with additional packages required for UI design, routing, and interaction with the JSON database.

Tools and Libraries Used:

- React JS
- React Router
- JSON Server (used as a lightweight database)
- Material UI (for UI components)
- React Icons (for UI icons)

These tools collectively enabled the creation of an interactive quiz interface, styled components, and seamless data retrieval from the JSON database.

Reference Resources:

- <https://react.dev/learn/installation>
- <https://mui.com/material-ui/getting-started/>
- <https://github.com/typicode/json-server>

Milestone 2: Web Development

Setup of the React Application

- A new React project was created using the appropriate setup command.
- The file structure was organized to include components, pages, services, and assets.
- Routing was configured using React Router to allow navigation between quiz categories, quiz screens, instructions, and result pages.

- Required libraries such as Material UI, React Icons, and JSON Server were installed to support UI design and backend data management.

App.js component.

```

1 |
2 | import Layout from './components/Layout';
3 | import LandingPage from './components/LandingPage';
4 | import QuizEngine from './components/QuizEngine';
5 | import ResultsPage from './components/ResultsPage';
6 | import ReviewList from './components/ReviewList';
7 | import LoadingScreen from './components/LoadingScreen';
8 | import AuthPage from './components/AuthPage';
9 | import HistoryView from './components/HistoryView';
10 | import { useGameStore } from './store';
11 | import BrutalistButton from './components/BrutalistButton';
12 | import { RefreshCw } from 'lucide-react';
13 | import { AnimatePresence, motion } from 'framer-motion';
14 |
15 | const App = () => {
16 |   const {
17 |     view,
18 |     questions,
19 |     userAnswers,
20 |     isReviewMode,
21 |     reviewFilter,
22 |     error,
23 |     retryQuiz,
24 |     resetGame,
25 |     setReviewMode,
26 |     exitReview
27 |   } = useGameStore();
28 |
29 |   const renderContent = () => {
30 |     switch (view) {
31 |       case 'AUTH':
32 |         return <AuthPage />;
33 |       case 'LOADING':
34 |         return <LoadingScreen />;
35 |       case 'ERROR':
36 |         return (
37 |           <div className="flex flex-col items-center justify-center min-h-[50vh] text-center p-8">
38 |             <div className="border-4 border-black p-8 bg-brut-red text-white shadow-hard max-w-md">
39 |               <h2 className="text-3xl font-black mb-4">SYSTEM ERROR</h2>
40 |               <p className="font-mono mb-8">{error || "Unknown anomaly detected."}</p>
41 |               <BrutalistButton onClick={resetGame} variant="secondary" className="border-white hover:bg-black hover:text-white">
42 |                 <RefreshCw className="mr-2" /> REBOOT
43 |               </BrutalistButton>
44 |             </div>
45 |           </div>

```

Code Description:-

- ./App.css: The CSS file used for styling the App component and controlling the overall layout of the application.
- Imports such as Layout, LandingPage, QuizPage, ResultPage, ReviewList, AuthPage, HistoryView, and BrutalistButton: These components represent different screens and functional sections of the quiz application.
- useGameStore from ./store: A global state management hook used to store and access quiz data, view states, user answers, and other shared information throughout the application.
- The App function is defined, where various states and functions such as view, questions, userAnswers, reviewMode, reviewFilter, retryQuiz, resetGame, and exitReview are retrieved from useGameStore.
- The renderContent() function is used to determine which component should be displayed on screen based on the current view state (for example AUTH, LOADING, ERROR, QUIZ, RESULTS, REVIEW).
- In the ERROR view, a custom error layout is displayed, including an error message and a Reboot button created using the BrutalistButton component.
- The App component returns the Layout component, and inside it the output of renderContent(), ensuring that the appropriate screen is displayed depending on the current state of the application.

- Layout: This component serves as the main structural wrapper for the application, maintaining consistent UI structure across different pages.
- The App component is exported as the default export of the module, making it available for use in the entry point of the application.

2. Design UI components:

- Create Components.
- Implement layout and styling.
- Add navigation.

3. Implement frontend logic:

- Integration with API endpoints.
- Implement data binding.

Data Provider:-

```

1  // stores.js
2  import { create } from 'zustand';
3
4  import { generateQuizQuestions } from './ai';
5  import { saveQuizResult, fetchUserHistory } from './api';
6
7
8
9  export const useGameStore = create((set, get) => ({
10   view: 'AUTH',
11   config: {
12     topic: 'All',
13     difficulty: 'Medium',
14     questionCount: 5,
15     customTopic: ''
16   },
17   score: 0,
18   questions: [],
19   currentQuestionIndex: 0,
20   userAnswers: [],
21   error: undefined,
22   isReviewMode: false,
23   reviewFilter: 'ALL',
24   user: undefined,
25   history: [],
26
27   setPendingConfig: (newConfig) => set((state) => ({
28     config: { ...state.config, ...newConfig }
29   })),
30
31   setView: (view) => set({ view }),
32
33   startGame: async () => {
34     const { config } = get();
35
36
37     if (config.topic === 'Custom' && (!config.customTopic || config.customTopic.trim() === '')) {
38       alert("PLEASE ENTER A CUSTOM TOPIC TO PROCEED.");
39       return;
40     }
41
42     set({ view: 'LOADING', error: undefined });
43
44     try {
45       const questions = await generateQuizQuestions(

```

Code Description:-

- Imports the create function from the Zustand library to create a global state store for the application.
- Imports helper functions such as generateQuizQuestions from the AI module and saveQuizResult, fetchUserHistory from the API module, which are used for quiz

generation and saving user data.

- Creates the useGameStore state manager using Zustand, which holds all key variables needed for the quiz system, such as:
 - view: Controls which screen (Auth, Loading, Quiz, Result, Review) is displayed.
 - config: Stores quiz settings including topic, difficulty, and question count.
 - score, questions, currentQuestionIndex, userAnswers: Handle the active quiz session.
 - error, isReviewMode, reviewFilter, user, history: Additional states used for UI flow and user data.
- Provides functions such as setPendingConfig to update quiz settings, and setView to change the current screen.
- Defines startGame as an asynchronous function that validates custom topic input, switches the view to Loading, generates quiz questions using generateQuizQuestions, and initializes the quiz state.

DB Component:-

```
1 dbjson > ...
2 {
3   "results": [
4     {
5       "id": "90b9",
6       "userId": "google-1763790979714",
7       "topic": "about iron",
8       "difficulty": "Easy",
9       "score": 5,
10      "totalQuestions": 5,
11      "timestamp": "2025-11-22T05:57:23.668Z"
12    },
13    {
14      "id": "12b4",
15      "userId": "google-1763792504617",
16      "topic": "Generate 10 Java SE 17 OCP-style multiple-choice questions covering modules, \nrecords, sealed classes, lambdas, streams, gener
17      "difficulty": "Medium",
18      "score": 2,
19      "totalQuestions": 5,
20      "timestamp": "2025-11-22T06:23:17.098Z"
21    },
22    {
23      "id": "30e5",
24      "userId": "google-1763792745306",
25      "topic": "questions on golalng",
26      "difficulty": "Medium",
27      "score": 5,
28      "totalQuestions": 5,
29      "timestamp": "2025-11-22T06:27:01.622Z"
30    },
31    {
32      "id": "2921",
33      "userId": "google-1763793524502",
34      "topic": "Geography",
35      "difficulty": "Medium",
36      "score": 11,
37      "totalQuestions": 20,
38      "timestamp": "2025-11-22T06:40:53.224Z"
39    },
40    {
41      "id": "3a36",
42      "userId": "google-1763793524502",
43      "topic": "quantum physics",
44      "difficulty": "Easy",
45      "score": 5,
46      "totalQuestions": 5,
```

Code Description:-

- The db.json file serves as the local JSON database for the application, used through JSON Server to simulate a backend.
- It contains an array named "results", which stores all quiz attempts made by users.
- Each object inside the "results" array represents a single quiz result entry.
- Every entry includes the following fields:
 - id: A unique identifier automatically assigned to each result.
 - userId: Identifies the user who attempted the quiz.
 - topic: The topic used for generating the quiz.
 - difficulty: The difficulty level selected by the user.
 - score: The number of correct answers obtained.
 - totalQuestions: Total number of questions in the quiz.
 - timestamp: Stores the date and time when the quiz was completed.
- This structure allows the application to store past quiz history and retrieve it when needed, such as for showing the user's previous results.
- JSON Server uses this file to provide REST API endpoints (GET, POST, etc.) for interacting with quiz results during development.

Auth Page Component:-

```

1
2 import React, { useState } from 'react';
3 import { useGameStore } from '../store';
4 import { Github, Globe, Lock, UserPlus, LogIn } from 'lucide-react';
5 import { motion } from 'framer-motion';
6
7 const AuthPage = () => {
8   const { login } = useGameStore();
9   const [isLoading, setIsLoading] = useState(false);
10  const [mode, setMode] = useState('signin');
11
12  const handleAuth = (provider) => {
13    setIsLoading(true);
14
15    setTimeout(() => {
16      const mockUser = {
17        id: `${provider}-${Date.now()}`,
18        name: `${provider} === 'google' ? 'Google' : 'GitHub' Agent`,
19        email: `agent@${provider}.com`,
20        provider: provider
21      };
22
23      login(mockUser);
24      setIsLoading(false);
25    }, 1000);
26  };
27
28  return (
29    <div className="flex flex-col items-center justify-center min-h-[80vh] p-4 md:p-8 w-full">
30      <motion.div
31        initial={{ opacity: 0, scale: 0.95 }}
32        animate={{ opacity: 1, scale: 1 }}
33        className="w-full max-w-md flex flex-col"
34      >
35        <div className="flex w-full z-10">
36          <button
37            onClick={() => setMode('signin')}
38            className={`flex-1 border-4 border-black p-4 font-black text-xl uppercase transition-all flex items-center justify-center gap-2`}
39            ${mode === 'signin'
40              ? 'bg-white text-black border-b-0 top-1 pb-5 z-20'
41              : 'bg-gray-200 text-gray-500 hover:bg-gray-300 border-b-4 top-1 z-0'}
42          >
43            <LogIn size={20} />
44          </button>
45        </div>

```

Code Description:-

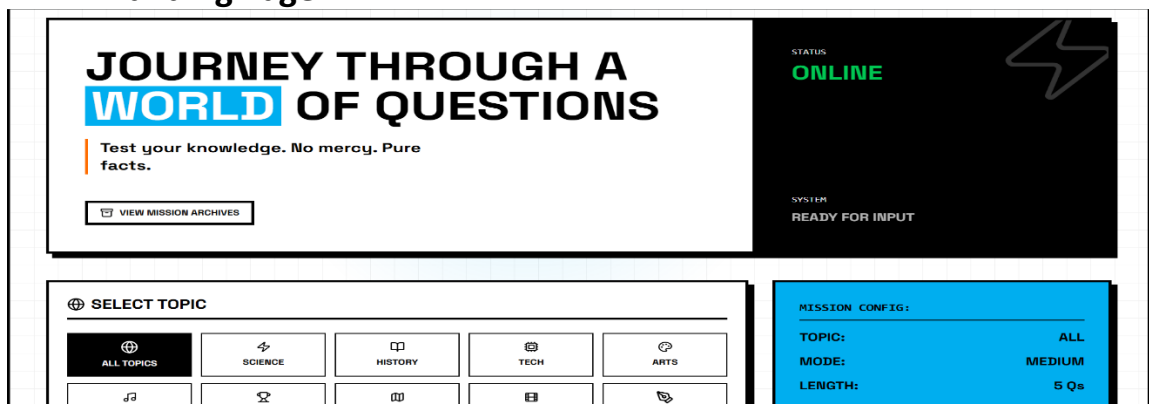
- Imports React, useState, and motion for UI animations, along with useGameStore for accessing global state and login functionality.
- Imports icon components such as Github, Globe, Lock, UserPlus, and Login from lucide-react to enhance the visual layout of the authentication screen.
- Defines the AuthPage functional component, which manages the login interface for the application.
- Retrieves the login function from useGameStore, which is used to update the global user state after authentication.
- Uses two local state variables, isLoading and mode.
 - isLoading controls the animation and delay during the login simulation.
 - mode switches between different authentication modes such as “signin.”
- Defines the handleAuth function, which accepts a provider name (such as Google or GitHub) and simulates a login process by enabling loading and creating a mock user object after a timeout.
- The mock user object contains id, name, email, and provider information based on the selected provider.
- After creating the mock user, the login function updates the global store, and loading is disabled.

Project Execution:

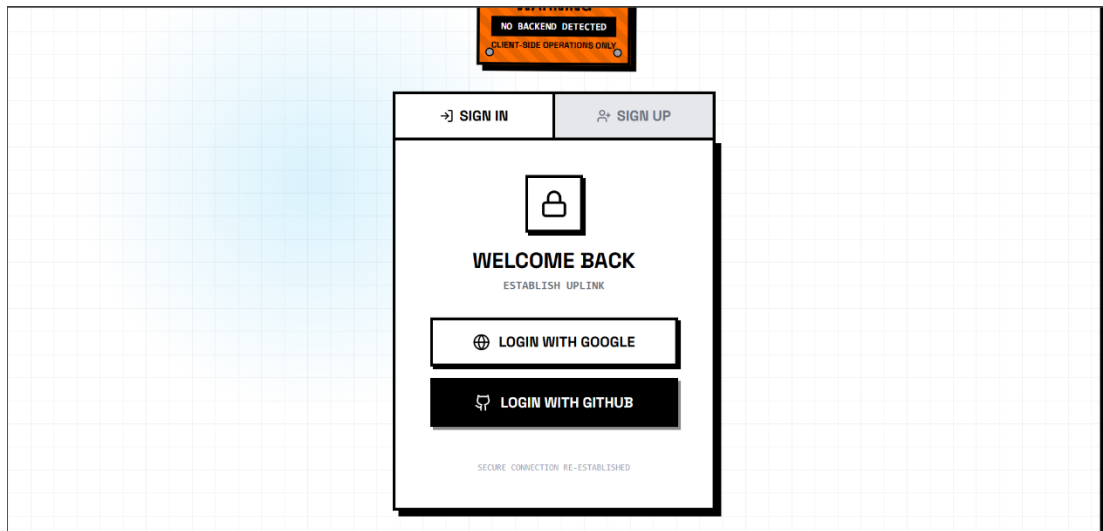
After completing the code, run the react application by using the command “npm start” or “npm run dev” if you are using vite.js

Here are some of the screenshots of the application.

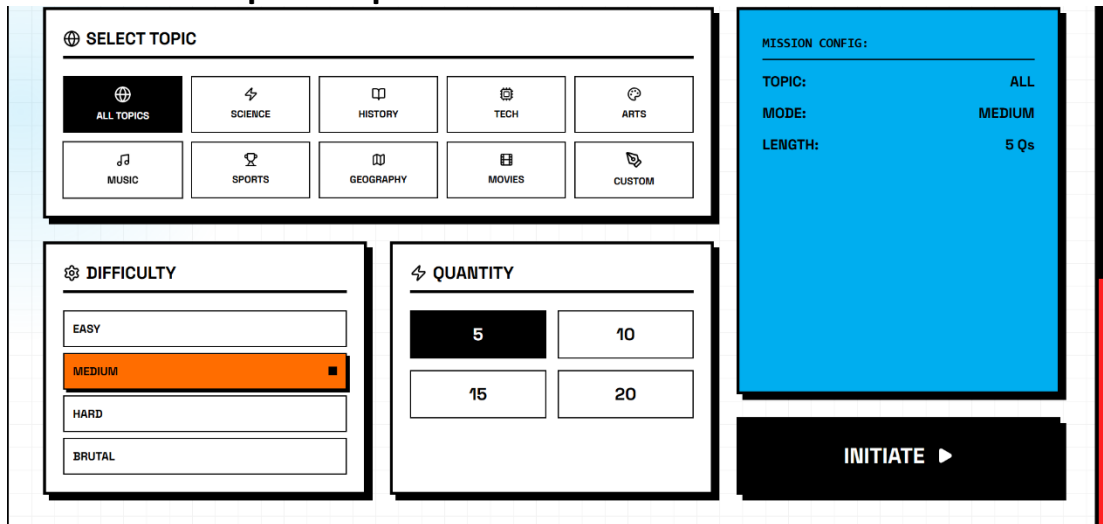
- **Landing Page**



- **Auth Page Component**



- **Select Topic Component**



- **Questions Component**

- **Results Component**

Project Demo link:

<https://drive.google.com/file/d/1HmIIPpQAQOax-VxmouLfrGV1crCH1ZaH/view>