Advanced Web Design

Project Title: Trivia Game using Vue 3

Student: Talia Kastrati

Index: 191527

Project Description

Trivia Game is an interactive application that offers users a selection of 24 distinct categories.
Users are able to choose between four difficulty levels: easy, medium, difficult, or a random mix. Points are awarded based on their answers.
Users must complete the quiz within a time limit of 3 minutes. After each question, they receive instant feedback on whether their response was correct or not.
Additionally, the application provides users with their overall score and the number of questions they answered correctly.
For each correctly answered question in the easy category, the user earns 10 points, while they receive 20 points for questions in the medium category and 30 points for those in the difficult category.
Incorrectly answered questions do not result in any negative points.

Project Details

I developed the application using Vue 3 in conjunction with Vite as the underlying technology stack.

To access the categories and questions for the game, I integrated the Open Trivia Database API, available at https://opentdb.com/. This API serves as the primary source for retrieving the quiz content.

The project structure is the following:

1. The ‘views’ folder

"The 'views' folder stores the five distinct pages that form the user interface of the application. When a user initiates the app, the initial redirection directs them to the 'HomePage.vue' file. This landing page provides an overview of available categories for the user's selection.

Once a category is chosen, the user is sent to the 'GameDifficulty.vue' page, where they can specify their preferred level of game difficulty. Subsequently, they are automatically routed to the 'QuestionPage.vue,' where the quiz logic and question presentation are stored.

Upon completing the quiz, users are then directed either to the 'NewGame.vue' or to the 'TimesUpPage.vue' in the event that their time has expired."

2. The **'components'** folder

"In the **'components'** folder, a collection of reusable Vue.js components are stored. These components help with structuring the user interface and maintaining a cohesive design throughout the app.

One such component is 'BaseTitle.vue,' which encapsulates the title/logo of the app. It features slots for inserting logos and titles dynamically. The 'BaseTitle' component is utilized within the 'MainLayout.vue,'

that serves as the foundation for the application's structure. 'MainLayout' is subsequently included in 'App.vue,' acting as a unifying layout framework for all views.

3. The 'composables' folder

"In the **'composables'** folder, a collection of JavaScript functions, referred to as 'composables' are stored.

The useAPI.js composable serves as a central module responsible for handling communication with the Open Trivia Database API. This composable encapsulates functions and data related to retrieving trivia categories and questions, which are subsequently displayed in the views of the application.

Here's a description of the main elements and functionality within the useAPI.js composable:

Axios Configuration: The composable initializes an Axios instance configured to communicate with the Open Trivia Database API, specifying the base URL as 'https://opentdb.com/'.

Categories: A ref variable named categories is defined to store the list of trivia categories. This variable will be populated with category data retrieved from the API.

getCategories Function: This asynchronous function, getCategories, is responsible for fetching trivia categories from the API. It first checks if the categories array is empty; if so, it makes a GET request to 'api_category.php' and populates categories with the response data.

getQuestion Function: Another asynchronous function, getQuestion, retrieves a trivia question based on a specified category. It accepts a categoryId as a parameter and sends a GET request to 'api.php' with the required parameters, including the category ID. It then returns a single trivia question from the response.

getQuestionDifficulty Function: Similar to getQuestion, this function, getQuestionDifficulty, retrieves a trivia question based on a category and difficulty level. It accepts categoryId and diff (difficulty) as parameters, sends a GET request to 'api.php' with the specified parameters, and returns a single trivia question with the requested difficulty level.

The router.js file

This file contains the configuration for the Vue Router in the Vue 3 application. Vue Router is responsible for managing the application's client-side navigation, allowing to define routes and map them to specific components.

> **createRouter** and **createWebHistory** are imported from **vue-router**. These functions are used to create the router instance and specify the router's mode.
>
> **HomePage** and various other components are imported using the **@** alias to specify the path to the components. These components represent different views in the application.

routes is an array of route objects. Each route object defines a URL path, a name for the route, and the component that should be rendered when the route is matched.

The first route object { path: '/', name: 'Home', component: HomePage } defines the root route '/', which corresponds to the HomePage component. It also has the name 'Home' for programmatic navigation.

The other route objects define various routes used in the application, such as routes for displaying trivia questions, selecting game difficulty, ending the game, and handling times-up scenarios. These routes include dynamic segments, such as :id and :difficulty, which allow for variable values in the URL.

For some of these routes, the component is not directly imported but instead loaded dynamically using a function () => import(...).The imported components are only fetched when the corresponding route is accessed.

The router is created using createRouter with the specified configuration options:

- history: This is set to createWebHistory() to use the HTML5 History API for routing. It provides clean and user-friendly URLs.
- routes: The array of route objects defined earlier is provided as the route configuration.

The router instance is exported as the default export of this module, making it available for use throughout the Vue application.

The logic and functionalities of the QuestionPage.vue view

Figure 1: The QuestionPage.vue view

Importing useAPI: This composable contains the logic for fetching and managing trivia categories.

Destructuring useAPI(): destructure the categories ref variable and the getCategories function from the result of calling useAPI().

onMounted Hook: Using the onMounted lifecycle hook, which runs code after the component has been mounted (i.e., when it's ready for interaction). Inside this hook, the getCategories function is invoked asynchronously with await. This function sends a request to the Open Trivia Database API to fetch the trivia categories.

Category Rendering in the Template: In the template section, is the div with the class categories. Here the divs are conditionally rendered based on whether the categories array has content (i.e., its length is greater than 0). This ensures that the categories are displayed only after they have been fetched from the API.

RouterLink Iteration: Inside the div with the categories class, a v-for loop is used to iterate over each category in the categories array. For each category, a RouterLink component is created that represents a link to the category's game difficulty page. The category.id is passed as a parameter in the URL, which is used to identify the selected category when navigating to the game difficulty page.

Category Name: Within each RouterLink, the category's name is displayedusing {{ category.name }}. This text is dynamically populated with the names of the fetched trivia categories.

The logic and functionalities of the GameDifficulty.vue view

Figure 2: The GameDifficulty.vue view

In the template section, I'm defining the structure of this page:

- I'm using the BaseTitle component to display the game's title and logo at the top of the page.
- There's a <p> element with the class p_difficulty that displays a message asking the user to select a game difficulty.
- Below that, I have a with the class difficulties, which will contain the links to different difficulty levels.

- Inside the difficulties , I'm using the router-link component from Vue Router to create links to different difficulty levels. The :to attribute of each router-link is constructed dynamically using the categoryId extracted from the route's parameters. This ensures that the links lead to the correct category and difficulty level pages.
- Each router-link has a specific class (easy, medium, hard, random) that defines its appearance and hover effect based on the selected difficulty level. For example, the easy link has a yellow border and background color, and it turns white when hovered.

The logic and functionalities of the QuestionPage.vue view

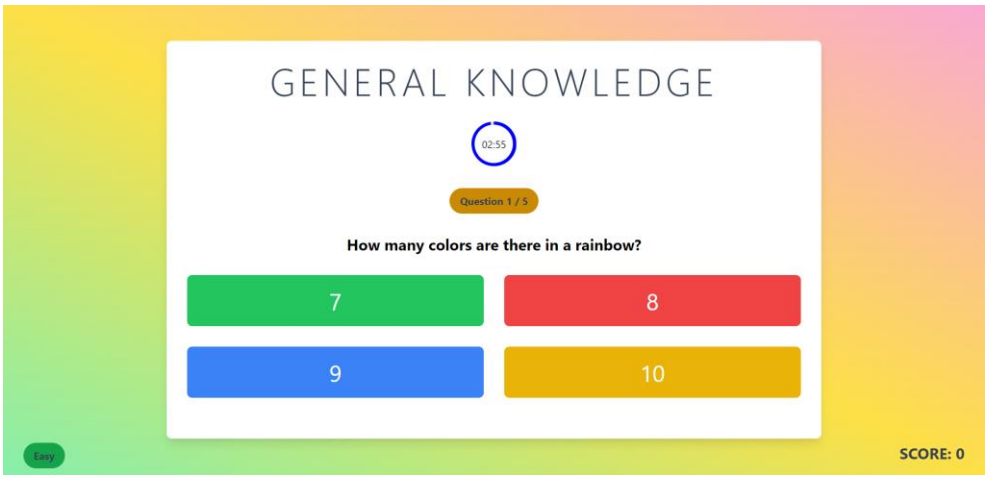

Figure 3: The QuestionPage.vue view

Import Statements:Importing several Vue-related and custom modules and components. Some of the important imports include useRoute, useRouter from Vue Router, custom composable functions like useAPI, useColor, useScore, useCounter, and various custom components like MainScore, NotificationAnswers, BaseTitle, and DifficultyChip.

Initialization: Initialize various reactive variables using the ref function. These variables are used to manage different aspects of the game, including the question, answers, notifications, and more.

Game Constants: Definition of several constants that are used throughout the game, such as the maximum number of questions (maxQuestions), countdown timer settings, and variables to track time-related data like minutes, seconds, setTime, and timestamps.

Event Handlers:

-handleAnswer: This function handles user answers and the associated logic. It takes the points earned as an argument. Inside this function, we update the score, track answered questions, show notifications, and move to the next question or the end of the game based on conditions.

-loadNextQuestion: This function loads the next question based on the selected difficulty level and category. It retrieves question data from the API, formats the answers, and shuffles them for display.

-countDownTimer: This function calculates and updates the countdown timer's appearance based on the remaining time. It also checks if the timer reaches certain thresholds and triggers specific actions (e.g., changing the background color, redirecting to specific pages).

Lifecycle Hooks:

-onMounted: This hook runs when the component is mounted. It initiates the countdown timer, loads the first question, and sets up initial game state.

-Template Markup: The template section contains the HTML structure of the game interface. It includes placeholders for displaying the current question, answers, countdown timer, question number, and various styling classes.

-Conditional Rendering: Elements like the countdown timer, question, answers, loading message, and notification are conditionally rendered based on the game state.

-Styling: The scoped style section contains CSS styles specific to this component. It defines the appearance of various UI elements, including the countdown timer, question, answers, and notifications. Y

-Dynamic Styling: Dynamic styles are used to update the appearance of the countdown timer, such as changing the background color as time runs out.

-Component Interaction: Interaction with other components, such as MainScore, BaseTitle, and NotificationAnswers, by including them in the template and passing data as props or by using Vue's reactivity system.