**תיק מתכנת**

**Client side:**

1. Component Documentation

**Component: `ReturnToMainMenuButton`**

JSX: ReturnToMainMenuButton.jsx

Purpose: This component renders a button that allows users to navigate back to the main menu. It is styled using a CSS module.

Key Functionality:

* The button uses React Router's `Link` component to navigate to the home route.
* The button is styled through an imported CSS module to ensure consistent appearance across the application.

CSS: ReturnToMainMenuButton.module.css

Purpose: Provides styling for the `ReturnToMainMenuButton` component.

Key Styles:

* Defines padding, background color, text color, border-radius, and hover effects for the button to ensure it stands out as an interactive element.

**Component: `QuizDisplay`**

JSX: QuizDisplay.jsx

Purpose: This component displays a quiz question along with multiple-choice options. Users can select an answer and submit it.

Key Functionality:

* The component takes in props for the question, options, selected answer, and event handlers for answer selection and form submission.
* It renders the quiz question and maps through the options to display each one as a clickable list item.
* The selected answer is highlighted, and the submit button is disabled until an answer is selected.

CSS: QuizDisplay.module.css

Purpose: Provides styling for the `QuizDisplay` component.

Key Styles:

* Styles the container, question text, answer options, and submit button.
* Includes responsive design elements and hover effects to enhance user interaction.

**Component: `Login`**

JSX: Login.jsx

Purpose: Handles user authentication by allowing users to enter their name and log in.

Key Functionality:

* Uses a custom hook (`useLogin`) to manage the state of the login form, including the entered name.
* On form submission, the entered name is passed to a function to handle the login process.

JS: useLogin.js

Purpose: Custom hook to manage the login state and logic.

Key Functionality:

* Manages the state for the user's name and handles changes and form submission.
* Calls the `onLogin` function with the user's name upon successful form submission.

CSS: Login.module.css

Purpose: Provides styling for the `Login` component.

Key Styles:

* Styles the login container, form, input fields, and submit button.
* Ensures that the design is responsive and visually appealing in both light and dark modes.

**Component: `MainMenu`**

JSX: MainMenu.jsx

Purpose: Acts as the central navigation point in the application, providing options to navigate to different sections like playing, generating, or creating quizzes.

Key Functionality:

* Uses a custom hook (`useMainMenu`) to manage navigation based on the button clicked.
* Dynamically displays a welcome message with the user's name and presents buttons for various actions.

JS: useMainMenu.js

* Purpose: Custom hook to manage navigation from the main menu.

Key Functionality:

* Provides a function (`handleNavigation`) that programmatically navigates to different routes within the application using React Router's `useNavigate`.

CSS: MainMenu.module.css

Purpose: Provides styling for the `MainMenu` component.

Key Styles:

* Styles the main menu container, options form, buttons, and headings.
* Includes responsive design elements and hover effects for interactive buttons.

**Component: `Header`**

JSX: Header.jsx

Purpose: Renders the application's header, containing a main menu button on the left and a theme toggle button on the right.

Key Functionality:

* The header is fixed at the top of the page and includes a button to toggle between light and dark modes.
* The theme toggle button text dynamically changes based on the current theme.

CSS: Header.module.css

Purpose: Provides styling for the `Header` component.

Key Styles:

* Styles the header container, buttons, and ensures the layout is responsive.
* - Includes hover effects for buttons and media queries for better usability on different screen sizes.

**Component: `QuizSummary`**

JSX: QuizSummary.jsx

Purpose: Displays the quiz results, including the user's score and a summary of their answers.

Key Functionality:

* Uses a custom hook (`useQuizSummary`) to retrieve quiz data and handle navigation back to the home page.
* Renders a detailed summary of each question, showing whether the user's answer was correct or incorrect.

JS: useQuizSummary.js

Purpose: Custom hook to manage the quiz summary logic, including navigation and data retrieval.

Key Functionality:

* Retrieves quiz questions, user answers, and the score from the location state.
* Provides a function to navigate back to the home page.

CSS: QuizSummary.module.css

Purpose: Provides styling for the `QuizSummary` component.

Key Styles:

* Styles the summary container, score text, question items, and navigation buttons.
* Includes specific styles for correct and incorrect answers to visually differentiate them.

**Component: `EnterQuizId`**

JSX: EnterQuizId.jsx

Purpose: Allows users to enter a quiz ID to start a quiz, handling validation and error handling.

Key Functionality:

* Uses a custom hook (`useFetchQuiz`) to fetch the quiz data based on the entered ID.
* If the quiz is found, the user is navigated to the quiz play screen; otherwise, an error message is shown.

JS: useFetchQuiz.js

Purpose: Custom hook to fetch quiz data based on the provided quiz ID.

Key Functionality:

* Sends a GET request to the backend API to retrieve quiz data by ID.
* Manages loading and error states, providing feedback to the user if the quiz is not found or if an error occurs.

CSS: EnterQuizId.module.css

Purpose: Provides styling for the `EnterQuizId` component.

Key Styles:

* Styles the container, form, input fields, and submit button.
* Includes responsive design and transitions for interactive elements.

**Component: `PlayQuiz`**

JSX: PlayQuiz.jsx

Purpose: Enables users to play a quiz fetched from the backend, displaying each question one at a time and calculating the final score.

Key Functionality:

* Uses a custom hook (`useQuiz`) to fetch quiz questions and manage state for playing the quiz.
* Handles user interactions, such as selecting answers and moving to the next question, while calculating the score in real-time.

JS: useQuiz.js

Purpose: Custom hook to fetch quiz data and manage the state for playing the quiz.

Key Functionality:

* Fetches quiz data from the backend and shuffles the answer options to ensure randomness.
* Manages user interactions, score calculation, and navigation to the quiz summary screen after the quiz is completed.

CSS: PlayQuiz.module.css

Purpose: Provides styling for the `PlayQuiz` component.

Key Styles:

* Styles the quiz container, question text, answer options, and navigation buttons.
* Includes responsive design and hover effects for a better user experience.

**Component: `CreateQuiz`**

JSX: CreateQuiz.jsx

Purpose: Allows users to create a custom quiz by selecting difficulty, category, and the number of questions. The selected questions are saved to the backend, and a Quiz ID is generated.

Key Functionality:

* Uses a custom hook (`useCreateQuiz`) to manage the state of quiz creation, including fetching categories and questions, and saving the quiz.
* The component provides a step-by-step interface for creating the quiz and generates a unique Quiz ID for sharing.

JS: useCreateQuiz.js

Purpose: Custom hook to handle the state and logic for creating a quiz.

Key Functionality:

* Manages the state for quiz difficulty, category, number of questions, and selected questions.
* Fetches categories and questions from The Trivia API, and saves the created quiz to the backend.
* Provides functions to handle question selection, form submission, and copying the generated Quiz ID.

CSS: CreateQuiz.module.css

Purpose: Provides styling for the `CreateQuiz` component.

Key Styles:

* Styles the form container, input fields, buttons, and question list.
* Includes responsive design and hover effects, as well as loading indicators for better user feedback.

**Component: `GenerateQuiz`**

JSX: GenerateQuiz.jsx

Purpose: Allows users to generate a quiz by selecting difficulty, category, and the number of questions. The generated quiz is saved on the backend, and a shareable Quiz ID is provided.

Key Functionality:

* Uses a custom hook (`useGenerateQuiz`) to manage the state of quiz generation, including fetching categories and questions, and saving the quiz.
* Provides a simple interface for generating the quiz and copying the Quiz ID.

JS: useGenerateQuiz.js

Purpose: Custom hook to handle the state and logic for generating a quiz.

Key Functionality:

* Manages the state for quiz difficulty, category, number of questions, and the generated quiz.
* Fetches categories from The Trivia API and generates a quiz based on user selections.
* Provides functions for submitting the form, handling errors, and copying the Quiz ID.

CSS: GenerateQuiz.module.css

Purpose: Provides styling for the `GenerateQuiz` component.

Key Styles:

* Styles the form container, input fields, buttons, and generated quiz display.
* Includes responsive design, loading indicators, and animations to enhance user experience.

**Server-Side Components**

**File: `server.js`**

Purpose:

* The main entry point for the server-side application, responsible for setting up the Express server, configuring middleware, and establishing a connection to the MongoDB database.

Key Functionality:

Express Server Setup:

* Creates an Express application and configures it to listen on a specified port (default: 5000).
* Configures middleware to handle JSON requests and manage Cross-Origin Resource Sharing (CORS) to allow requests from the frontend.

MongoDB Connection:

* Connects to a MongoDB database using the connection string from environment variables (`MONGO\_URI`).
* Logs the connection status to the console.

Routes:

* Sets up the main API routes by importing and using the `quizRoutes` module to handle quiz-related operations.

Default Route:

* Defines a basic route (`/`) to ensure the server is running.

**File: `quizRoutes.js`**

Purpose:

* Defines and handles the API routes related to quizzes, including creating new quizzes, retrieving quizzes by ID, and optionally retrieving all quizzes.

Key Functionality:

* Creating a New Quiz (`POST /api/quizzes/create`):

1. Generates a unique quiz ID using the `uuid` library.
2. Validates the request to ensure that the `questions` array is provided.
3. Saves the quiz to the MongoDB database using the `Quiz` model.
4. Responds with the generated quiz ID or an error if the creation fails.

* Retrieving a Quiz by ID (`GET /api/quizzes/:quizId`):

1. Fetches a quiz from the database based on the provided `quizId`.
2. Responds with the quiz data if found, or an error if not found.

File: `Quiz.js`

Purpose:

* Defines the Mongoose schema and model for quizzes, specifying the structure of quiz documents stored in the MongoDB database.

Key Functionality:

* Mongoose Schema:

1. `quizId`: A unique identifier for each quiz, stored as a string and required.
2. `questions`: An array of objects, each representing a question with the following fields:
3. `question`: The text of the question.
4. `correctAnswer`: The correct answer for the question.
5. `incorrectAnswers`: An array of incorrect answers.
6. `createdAt`: A timestamp indicating when the quiz was created, with a default value of the current date and time.

* Mongoose Model:

1. Exports the `Quiz` model, which allows the application to interact with the `quizzes` collection in MongoDB.

2. APIs, Database, and Environment Setup

**APIs Used:**

* Backend API:

Purpose: The server-side API handles all quiz-related operations, including creating and retrieving quizzes.

* Endpoint Examples:

1. `POST /api/quizzes/create`: Creates a new quiz and stores it in the database.
2. `GET /api/quizzes/:quizId`: Retrieves a quiz by its unique ID.
3. `GET /api/quizzes`: Optionally retrieves all quizzes stored in the database.

* The Trivia API:

Purpose: Fetching quiz categories and questions for creating or generating quizzes.

Endpoint Examples:

1. `https://the-trivia-api.com/api/categories` - Fetches available categories.
2. `https://the-trivia-api.com/api/questions` - Fetches quiz questions based on selected parameters.

* Backend API:

Purpose: Interacting with the backend for storing and retrieving quizzes.

Endpoint Examples:

1. `/api/quizzes/create` - Saves a new quiz and generates a Quiz ID.
2. `/api/quizzes/{quizId}` - Retrieves a quiz based on the Quiz ID.

Database:

* MongoDB:

Purpose: Storing quiz data, including questions, answers, and user scores.

Link to DB: MongoDB is hosted on a cloud service like MongoDB Atlas, accessible via the backend API. The exact connection string would be set in the environment variables (e.g., `MONGO\_URI`).

Environment Setup:

* Vite:

Purpose: A fast build tool and development server. Used for managing environment variables and optimizing the development experience.

Special Environment Variables:

1. `VITE\_API\_URL`: The base URL for the backend API.

Installation:

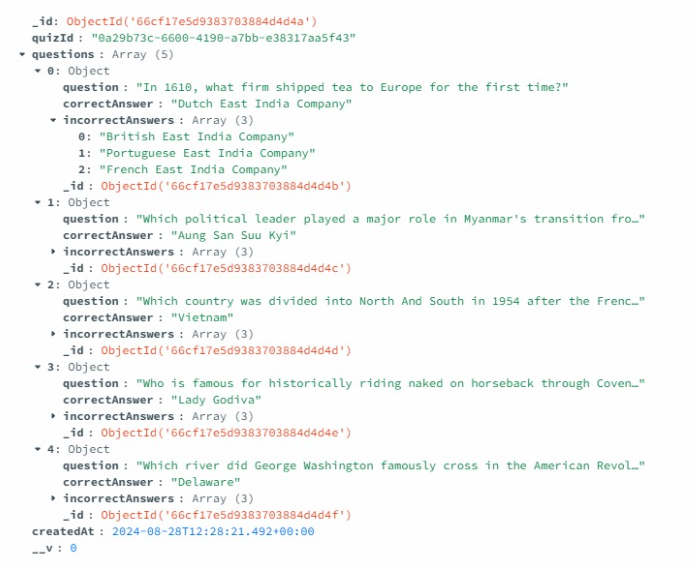
1. Installed via npm as part of the development dependencies (`npm install vite`).

* Node.js and Express:

Purpose: Used to build the backend API that interacts with the MongoDB database.

Installation:

1. Installed via npm (`npm install express mongoose`).

Example of quiz in the DB: