**Explicativo sobre la Aplicación Educativa de Algoritmos**

| **Fecha:** | Noviembre 14 del 2024 |
| --- | --- |
| **Docente:** | Alexander Toscano |
| **Presentado por:** | Jesús Barrios  Jorge Mogollón  Joel Serrano |
|  |  |

| **Explicación sobre como se uso componentes para la maquina tragamonedas**  La máquina tragamonedas está construida utilizando componentes en Vue.js, lo que permite que la aplicación sea modular, fácil de mantener y escalable. A continuación, te explicaré cómo se usaron los componentes en la máquina tragamonedas:  **Componentización de la Aplicación**  Una de las principales ventajas de utilizar Vue.js es la capacidad de dividir la interfaz de usuario en componentes independientes. En la máquina tragamonedas, hemos dividido la interfaz en tres componentes principales:   * SlotScreen.vue: Muestra los carretes que giran. * LeverAndCredits.vue: Contiene la palanca para iniciar el giro y muestra los créditos restantes. * PrizeTable.vue: Muestra una tabla con los posibles premios. * Estos componentes están integrados en un componente principal llamado SlotMachine.vue, que los organiza y gestiona la interacción entre ellos.   **Ventajas de Usar Componentes**   * Modularidad:Cada componente tiene su propio archivo .vue, lo que significa que su estructura (HTML), comportamiento (JavaScript), y estilo (CSS) están encapsulados. Esto facilita el mantenimiento y la actualización de cada parte de la aplicación sin afectar a las demás. * Reusabilidad:Los componentes pueden ser reutilizados en diferentes partes de la aplicación o incluso en diferentes proyectos. Si en el futuro necesitas una tabla de premios o una pantalla de carretes, puedes reutilizar estos componentes con pequeñas modificaciones. * Separación de Preocupaciones:Al dividir la aplicación en componentes, cada uno tiene una única responsabilidad. Por ejemplo, SlotScreen.vue se encarga únicamente de manejar la visualización y el comportamiento de los carretes, mientras que LeverAndCredits.vue gestiona los créditos y la palanca.   **Cómo se Usaron los Componentes en la Máquina Tragamonedas**  **a) SlotScreen.vue**  Este componente se encarga de mostrar los carretes de la máquina tragamonedas.   * Encapsulación de Estado:El estado de los carretes (reels y displayedReels) está encapsulado dentro del componente. Esto significa que SlotScreen.vue es completamente responsable de cómo se muestran y manejan los carretes. * Comunicación entre Componentes: El componente escucha un evento (spin-reels) emitido desde LeverAndCredits.vue para iniciar el giro de los carretes. Esto es posible gracias a la emisión de eventos en Vue, que permite la comunicación entre componentes hermanos a través del componente principal.   **b) LeverAndCredits.vue**  Este componente maneja los créditos y la palanca.   * Manejo de Eventos:Cuando el usuario hace clic en la palanca, se emite un evento (spin-reels) que es escuchado por SlotMachine.vue. Este evento luego es manejado por SlotScreen.vue para iniciar el giro de los carretes. * Encapsulación de Lógica: La lógica para disminuir los créditos y verificar si el usuario tiene suficientes créditos está encapsulada dentro de este componente, lo que lo hace responsable de todo lo relacionado con los créditos.   **c) PrizeTable.vue**  Este componente muestra los posibles premios.   * Encapsulación de Datos: Los premios y sus imágenes están encapsulados dentro del componente PrizeTable.vue, lo que significa que si necesitas actualizar los premios o las imágenes, solo necesitas modificar este componente.   **Flexibilidad en el Diseño**  El uso de componentes permite ajustar fácilmente el diseño y el estilo sin interferir con el comportamiento de los otros componentes. Por ejemplo, si deseas cambiar el diseño de la tabla de premios, puedes hacerlo directamente en PrizeTable.vue sin afectar el resto de la aplicación.  **Integración en SlotMachine.vue**  El componente SlotMachine.vue actúa como el contenedor principal que organiza y coordina los otros componentes.  **Integración de Componentes**   * SlotMachine.vue importa y utiliza SlotScreen.vue, LeverAndCredits.vue, y PrizeTable.vue dentro de su template. Esto crea la estructura visual de la máquina tragamonedas.   **Comunicación entre Componentes**   * SlotMachine.vue actúa como intermediario en la comunicación entre LeverAndCredits.vue y SlotScreen.vue. Recibe el evento spin-reels de LeverAndCredits.vue y lo maneja llamando al método spinReels() en SlotScreen.vue.   **Página Principal (index.vue)**  Finalmente, la página index.vue es donde se integra el componente SlotMachine.vue.  **Encapsulación de la Lógica Principal**  La página index.vue simplemente incluye el componente SlotMachine.vue, encapsulando así toda la lógica y presentación de la máquina tragamonedas en un solo componente.  **Conclusión**  El uso de componentes en Vue.js ha permitido que la máquina tragamonedas esté organizada en partes pequeñas y manejables. Cada componente se encarga de una parte específica de la aplicación, lo que facilita el desarrollo, la depuración y el mantenimiento. Además, la comunicación entre componentes a través de eventos hace que la interacción sea fluida y modular, permitiendo la separación clara de responsabilidades y la posibilidad de reutilizar código en el futuro. |
| --- |

| **Proyecto de Visualización  de Respuestas en JSON** Este proyecto permite mostrar una lista de respuestas con información de su nivel de calidad. Usa Nuxt 3 como framework principal, Pinia para el manejo del estado y una API simulada para la carga de datos en formato JSON. |
| --- |
| **Estructura del Proyecto**  **Components/Respuestas.vue:** Componente de visualización de respuestas.  **Server/api/consultas/index.get.ts:** Archivo de la API que devuelve datos de respuestas en JSON.  **stores/respuestas.ts:** Store de Pinia que maneja el estado y la obtención de respuestas.  **Pages/index.vue:** Página principal que integra el componente Respuestas. |
| *La siguiente imagen representa la página principal de la aplicación y utiliza el componente Respuestas para mostrar las respuestas.* **Explicación de la Página Principal**  Uso de <Respuestas />:  Esta página incluye el componente Respuestas, que se encarga de mostrar la lista de respuestas.El componente Respuestas se importa y se inserta en el template para que se muestre en la página.  **Función**  La página index.vue sirve como punto de entrada a la aplicación, donde se visualizan las respuestas cargadas.Este componente es responsable de mostrar la lista de respuestas obtenidas del store respuestas.  **Explicación del Componente**  <template>:  Muestra un encabezado "Respuestas" para indicar el propósito del componente.Si hay respuestas disponibles (getRespuestas.length > 0), recorre la lista y muestra cada respuesta con su título (title), descripción (description) y nivel de calidad (level).Si no hay respuestas, muestra un mensaje indicando que no hay datos disponibles.  **Script**  **Store:** Importa y usa el store useRespuestasStore para acceder a las respuestas. **Datos:** Llama a la acción fetchRespuestas para cargar los datos desde la API al montarse el componente. **getRespuestas**: Es un getter del store que permite acceder a la lista de respuestas cargadas.  Este componente está diseñado para ser reutilizable y se puede incorporar en cualquier página que requiera mostrar las respuestas.  **Server/api/consultas/index.get.ts**  Este archivo contiene la lógica del endpoint de la API. Se trata de un archivo de servidor en la carpeta server y se encarga de devolver un JSON con datos simulados de respuestas.    **Explicación del Endpoint**  **Propósito**: Este archivo simula una API que devuelve una lista de respuestas en formato JSON. Cada respuesta incluye:  **id:** Identificador único de cada respuesta. **title:** Título de la respuesta. **level:** Indica el nivel o calidad de la respuesta (por ejemplo, "Muy buena", "Buena", "Mala"). **description:** Breve descripción de la respuesta.  **Funcionamiento**  Al acceder a /api/consultas, este endpoint devuelve el array de respuestas.El método defineEventHandler permite manejar las solicitudes a este endpoint y devolver la lista de respuestas simulada.Este archivo facilita el desarrollo sin necesidad de una base de datos real, ya que simula los datos.    **Stores/respuestas.ts**  Este archivo define el store use Respuestas Store utilizando Pinia para manejar el estado de las respuestas de manera centralizada.    **Explicación del Store**  **Interfaz Respuesta**: Define la estructura de los datos de una respuesta, asegurando que cada respuesta tenga id, title, level y description.  **Estado (state)**  **respuestas:** Array que almacena la lista de respuestas obtenida de la API.  **Getters**  **getRespuestas:** Devuelve la lista de respuestas del estado, permitiendo el acceso al array respuestas.  **Acciones**  **fetchRespuestas:** Acción que obtiene datos de /api/consultas usando $fetch. **Control de Errores:** Muestra un mensaje en consola si ocurre algún error al intentar obtener los datos.  El store permite la gestión centralizada de las respuestas y asegura que se puedan reutilizar en múltiples componentes si es necesario.  **Resumen para la Entrega**  Propósito General: Esta aplicación permite cargar, almacenar y visualizar una lista de respuestas con su nivel de calidad usando Nuxt 3, Pinia y una API simulada.  **Componentes Clave**   * **Respuestas.vue:** Muestra la lista de respuestas en la página principal. * **index.get.ts (API):** Proporciona una lista de respuestas en formato JSON desde un endpoint local. * **respuestas.ts (Store):** Centraliza el estado de respuestas y facilita su acceso desde distintos componentes. * **index.vue:** Página principal que carga y muestra el componente Respuestas.   Este documento te permitirá explicar el funcionamiento de la aplicación y sus componentes en detalle. Con esta estructura, tienes una base sólida para construir aplicaciones de datos en Nuxt y Pinia*.* |
|  |
| **Link del repositorio**  *https://github.com/misterx1999/Proyecto-de-Visualizaci-n-de-Respuestas-en-JSON.git* |

| Documentación de la Interacción con la API en Edupro |
| --- |
| **Componentes Relacionados con la API**  **DocenteRespuestas.vue**  Este componente se utiliza para que los docentes puedan visualizar las respuestas enviadas por los estudiantes y calificarlas.  **Codigo de DocentesRespuestas.vue**    **Funcionalidad**  Obtiene las respuestas de la API mediante el store respuestas.ts.  Permite a los docentes valorar respuestas enviadas por los estudiantes.  **Pages Relacionadas con la API**  **docentes.vue** Esta página utiliza el componente DocenteRespuestas.vue para mostrar las respuestas y permitir la valoración.  **Código de Docentes.vue**    **Funcionalidad**   * Redirige automáticamente a esta página si el usuario tiene el rol de docente. * Carga el componente **DocenteRespuestas** para mostrar las respuestas.   **algoritmos.vue**  Esta página permite a los estudiantes enviar sus respuestas, las cuales se almacenan en la API.      **Funcionalidad**   * Los estudiantes envían sus respuestas desde el componente **ActividadAlgoritmo.vue.** * Las respuestas se envían al store y posteriormente a la API.   **Stores Relacionados con la API**  **respuestas.ts**   * Este archivo maneja la lógica de almacenamiento, obtención y valoración de las respuestas.     **Funcionalidad**   * **fetchRespuestas:** Obtiene respuestas desde la API. * **addRespuesta:** Agrega una nueva respuesta enviada por un estudiante. * **valorarRespuesta:** Permite valorar las respuestas obtenidas.   **Servidor (Server)**  **index.get.ts**   * Este archivo devuelve un array de respuestas simuladas desde la API.     **Funcionalidad**   * Proporciona un endpoint para obtener respuestas simuladas.   **index.post.ts**   * Este archivo simula la actualización o adición de respuestas en la API.     **Funcionalidad**   * Permite simular la actualización de una respuesta. * Utiliza el cuerpo de la solicitud para actualizar los datos. |
|  |
|  |