|  |  |
| --- | --- |
| **NOMBRE ESTUDIANTE:** | Issac de la Cadena  Carlos Bayas  Kevin Donoso  Andrea Oña  Daniel Oña |
| **FECHA:** | 08-12-2024 |
|  |  |
|  |  |
| **TEMA:** | Desarrollo del producto versión 2 |
|  |  |

**Introducción**

En este informe se documentan las actividades realizadas para garantizar la calidad del código de las interfaces web proporcionadas. Estas actividades incluyeron la refactorización del código HTML, la mejora de la semántica, estandarización de buenas prácticas de nombramiento y indentación, y la propuesta de casos de prueba adicionales para evaluar la funcionalidad y la calidad del sistema.

**Objetivos del Código**

* Obtener datos del backend y mostrarlos en tablas.
* Validación de credenciales y manejo de redirecciones tras el inicio de sesión.
* Implementar un flujo lógico para juegos interactivos, registro de estadísticas y navegación entre pantallas.
* Listar pacientes, agregar nuevos registros y navegar a la información detallada de un paciente.

**Nombrado Claro**

# Refactorización del Código HTML

**Archivo: Juego de Memoria Semántica**

**Descripción de la tarea:** El código original del juego de memoria semántica fue revisado para:

1. Añadir comentarios descriptivos en el código.
2. Estandarizar la indentación y la estructura.
3. Usar mejores prácticas en nombres de clases e identificadores.

**Código Refactorizado:**

<!DOCTYPE html>

<html lang="es">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Juego de Memoria Semántica</title>

    <!-- Enlace a la hoja de estilos -->

    <link rel="stylesheet" href="../static/styles.css">

</head>

<body>

    <!-- Contenedor principal del juego -->

    <div id="pantalla-juego" class="pantalla">

        <header>

            <!-- Botón para iniciar un nuevo juego -->

            <button id="boton-nuevo-juego">Nuevo Juego</button>

            <h1>Memoria Semántica</h1>

            <!-- Botón para salir del juego -->

            <button id="boton-salir">Salir</button>

        </header>

        <!-- Instrucciones para el usuario -->

        <p class="instrucciones">INSTRUCCIONES: Arrastra cada palabra a su categoría</p>

        <!-- Contenedor de palabras -->

        <section class="contenedor-palabras">

            <div class="palabra" draggable="true" data-categoria="Aseo">Jabón</div>

            <div class="palabra" draggable="true" data-categoria="Cocina">Cuchillo</div>

            <div class="palabra" draggable="true" data-categoria="Muebles">Librero</div>

            <!-- Más palabras aquí -->

        </section>

        <!-- Contenedor de categorías -->

        <section class="contenedor-categorias">

            <div class="categoria" data-categoria="Cocina">Cocina</div>

            <div class="categoria" data-categoria="Aseo">Aseo</div>

            <div class="categoria" data-categoria="Muebles">Muebles</div>

        </section>

    </div>

    <!-- Pantalla final del juego -->

    <div id="pantalla-final" class="pantalla oculto">

        <h2>¡Bien Hecho!</h2>

        <p>Juego completado: 8/8</p>

        <button id="boton-reiniciar">Jugar de Nuevo</button>

        <button id="boton-salir2">Salir</button>

    </div>

    <!-- Contenedor para notificaciones emergentes -->

    <div id="notificacion" class="notificacion"></div>

    <!-- Enlace al archivo de JavaScript -->

    <script src="../static/script.js"></script>

</body>

</html>

**1.** **Nombrado Claro**

* Cambiar nombres ambiguos a nombres más descriptivos:
  + Variables como p o palabras pueden reemplazarse con nombres significativos como palabraElemento o elementoPalabra.
  + tiempo podría ser temporizadorJuego.

**2. Indentación**

* Ajustar la indentación para que el código sea más legible:
  + Asegurarse de que cada bloque de código esté correctamente alineado.
  + **Antes:**

palabras.forEach((p)=>{

    p.addEventListener('dragstart',e=>{ // código });

    });

**Después:**

palabras.forEach((palabraElemento) => {

    palabraElemento.addEventListener('dragstart', (evento) => {

        // Código

    });

});

**3. Comentarios**

* Añadir comentarios claros y útiles para explicar bloques complejos:
  + Ejemplo:  
    **Antes:**

const palabras = document.querySelectorAll('.palabra');

**Después:**

// Seleccionar todas las palabras arrastrables en el juego

const palabras = document.querySelectorAll('.palabra');

**4. Buenas prácticas de ES6**

* Reemplazar var con let o const según corresponda:
  + Usar const para variables que no cambian.
  + Usar let para variables cuyo valor puede cambiar.
  + Ejemplo:  
    **Antes:**

var tiempo = 60;

**Después:**

let tiempo = 60;

* Usar funciones de flecha para simplificar el código cuando sea apropiado:
  + **Antes:**

palabras.forEach(function(p) {

    p.addEventListener('dragstart', function(e) {

        // Código

    });

});

* + **Después:**

palabras.forEach((palabraElemento) => {

    palabraElemento.addEventListener('dragstart', (evento) => {

        // Código

    });

});

**5. Separación de Responsabilidades**

* Dividir funciones grandes en funciones más pequeñas con responsabilidades únicas:
  + **Antes:**

palabras.forEach((p) => {

    p.addEventListener('dragstart', (e) => {

        // Lógica para dragstart

    });

    p.addEventListener('dragend', (e) => {

        // Lógica para dragend

    });

});

* + **Después:**

const configurarEventosPalabra = (palabraElemento) => {

    palabraElemento.addEventListener('dragstart', manejarInicioArrastre);

    palabraElemento.addEventListener('dragend', manejarFinArrastre);

};

palabras.forEach(configurarEventosPalabra);

* Definir funciones específicas para cada tipo de evento:

const manejarInicioArrastre = (evento) => {

    // Lógica para el inicio del arrastre

};

const manejarFinArrastre = (evento) => {

    // Lógica para el fin del arrastre

};

**6. Validación de Inputs**

* Verificar los valores antes de realizar operaciones con ellos:
  + **Antes:**

temporizador.textContent = `${tiempo}`;

* + **Después:**

if (!isNaN(tiempo) && tiempo >= 0) {

    temporizador.textContent = `${tiempo}`;

} else {

    console.error("El valor de 'tiempo' no es válido");

}

**7. Uso de Constantes**

* Definir valores constantes en un lugar único:
  + **Antes:**

const tiempo = 60;

* + **Después:**

const TIEMPO\_INICIAL = 60;

let tiempo = TIEMPO\_INICIAL;

**8. Manejo de Errores**

* Añadir manejo de errores para evitar interrupciones en la ejecución:

try {

    const palabras = document.querySelectorAll('.palabra');

    palabras.forEach(configurarEventosPalabra);

} catch (error) {

    console.error("Error al configurar los eventos de las palabras:", error);

}

**Código Refactorizado Final**

// Constantes y variables

const TIEMPO\_INICIAL = 60;

let tiempo = TIEMPO\_INICIAL;

// Elementos del DOM

const palabras = document.querySelectorAll('.palabra');

const temporizador = document.getElementById('temporizador');

// Funciones de manejo de eventos

const manejarInicioArrastre = (evento) => {

    console.log("Arrastre iniciado", evento);

    // Lógica adicional...

};

const manejarFinArrastre = (evento) => {

    console.log("Arrastre finalizado", evento);

    // Lógica adicional...

};

const configurarEventosPalabra = (palabraElemento) => {

    palabraElemento.addEventListener('dragstart', manejarInicioArrastre);

    palabraElemento.addEventListener('dragend', manejarFinArrastre);

};

// Configuración inicial

try {

    palabras.forEach(configurarEventosPalabra);

    temporizador.textContent = `${tiempo}`;

} catch (error) {

    console.error("Error durante la configuración inicial:", error);

}

# Casos de Prueba Funcionales y de Calidad

Se diseñaron y ejecutaron casos de prueba adicionales para validar la funcionalidad y la calidad de las interfaces. A continuación, se detallan algunos ejemplos:

**Casos de Prueba Funcionales**

1. **Casos de Prueba con Resultados**

|  |  |  |
| --- | --- | --- |
| **ID:** P-001 | **Título:** Validación de la entrada del nombre al iniciar el juego | |
| **Descripción:**  Se espera probar el correcto funcionamiento de la aplicación al dejar el campo del nombre en blanco, con el objetivo de ver cómo responde ante un campo vacío. | | |
| **Procedimiento de prueba:**   1. Ejecutar la aplicación para visualizar la ventana inicial. 2. Dar clic en el campo “Tu nombre”. 3. Dejar el campo vacío. 4. Dar click en el botón “Iniciar Juego”. | | |
| **Resultados esperados:**   El juego no debería iniciar y se presentaría una ventana indicando un mensaje de error que solicite el ingreso del nombre. | | |
| **EJECUCIONES DEL CASO DE PRUEBA** | | |
| **Fecha:**  7-noviembre- 2024  **Responsable:**   Daniel Oña | **Resultados obtenidos:**  El resultado obtenido al ejecutar el procedimiento de prueba fue el despliegue de un mensaje de error solicitando al usuario que por favor ingrese su nombre. | **Aprobado ( X )** |
| **Capturas:** | | |
| **Fecha:**  **Responsable:** | **Resultados obtenidos:** | **Reprobado  (  )**  **Aprobado (  )** |
| **Capturas:** | | |

|  |  |  |
| --- | --- | --- |
| **ID:** P-002 | **Título:** Asignación correcta de palabras a su categoría | |
| **Descripción:** Verificar que el usuario pueda arrastrar las palabras a las categorías correctas y recibir retroalimentación positiva. | | |
| **Procedimiento de prueba:**   1. Ingresar un nombre válido en el campo “Tu nombre” 2. Hacer click en botón “Iniciar Juego” 3. Arrastrar cada palabra a su categoría correcta. | | |
| **Resultados esperados:**  Cada palabra debería asignarse correctamente en su categoría sin mensajes de error e indicando al usuario que ha clasificado la palabra de forma adecuada. | | |
| **EJECUCIONES DEL CASO DE PRUEBA** | | |
| **Fecha:**  7-noviembre- 2024  **Responsable:**   Daniel Oña | **Resultados obtenidos:**  El resultado obtenido al ejecutar el procedimiento de prueba fue el despliegue del mensaje “¡Correcto!”, indicando al usuario que su intento ha sido exitoso. | **Aprobado ( X )** |
| **Capturas:** | | |
| **Fecha:**  **Responsable:** | **Resultados obtenidos:** | **Reprobado  (  )**  **Aprobado (  )** |
| **Capturas:** | | |

|  |  |  |
| --- | --- | --- |
| **ID:** P-003 | **Título:** Correcto reinicio del juego | |
| **Descripción:** Verificar el funcionamiento del botón “Nuevo juego” para restablecer el estado inicial de la aplicación. | | |
| **Procedimiento de prueba:**   1. Ingresar un nombre válido en el campo “Tu nombre” 2. Hacer click en botón “Iniciar Juego” 3. Asignar una de las palabras a su categoría. 4. Presionar el botón "Nuevo Juego". | | |
| **Resultados esperados:**   El juego debería volver a su estado inicial, solicitando de nuevo el nombre al usuario y todas las palabras vuelven a su posición sin asignaciones previas. | | |
| **EJECUCIONES DEL CASO DE PRUEBA** | | |
| **Fecha:**  7-noviembre- 2024  **Responsable:**   Daniel Oña | **Resultados obtenidos:**  El resultado obtenido al realizar el procedimiento de prueba fue la presentación de la página inicial de la aplicación, solicitando al usuario ingresar su nombre de nuevo para iniciar el juego. | **Aprobado ( X )** |
| **Capturas:** | | |
| **Fecha:**  **Responsable:** | **Resultados obtenidos:** | **Reprobado  (  )**  **Aprobado (  )** |
| **Capturas:** | | |

|  |  |  |
| --- | --- | --- |
| **ID:** P-004 | **Título:** Presentación de resultados | |
| **Descripción:** Verificar como al finalizar una sesión de juego se presentan los resultados obtenidos por el paciente. | | |
| **Procedimiento de prueba:**   1. Ingresar un nombre válido en el campo “Tu nombre” 2. Hacer click en botón “Iniciar Juego” 3. Fallar a propósito cuatro veces 4. Asignar cada una de las palabras a su categoría correspondiente. | | |
| **Resultados esperados:**  El juego debería presentar al usuario que el juego ha terminado, junto a los resultados correspondientes al intento. | | |
| **EJECUCIONES DEL CASO DE PRUEBA** | | |
| **Fecha:**  7-noviembre- 2024  **Responsable:**   Daniel Oña | **Resultados obtenidos:**  El resultado obtenido al realizar el procedimiento de prueba fue el despliegue de los resultados, presentando el nombre del usuario, pero no el tiempo que demoró ni los errores que cometió. | **Reprobado (X)** |
| **Capturas:** | | |
| **Fecha:**  **Responsable:** | **Resultados obtenidos:** | **Reprobado  (  )**  **Aprobado (  )** |
| **Capturas:** | | |

# Conclusiones

1. **Legibilidad y Mantenibilidad:**
   * La refactorización del código mejoró significativamente la legibilidad mediante el uso de nombres descriptivos, indentación correcta y comentarios explicativos.
   * Dividir el código en funciones pequeñas y específicas permite que el mantenimiento sea más sencillo y reduce la probabilidad de errores.
2. **Buenas Prácticas:**
   * La implementación de buenas prácticas de programación, como el uso de constantes, manejo de errores, y funciones de flecha, hace que el código sea más profesional y alineado con estándares modernos.
   * El cambio de var a let y const reduce problemas potenciales de alcance de variables y mejora la predictibilidad del código.
3. **Eficiencia:**
   * Al simplificar y separar las responsabilidades, el código se ejecuta de manera más eficiente y es más fácil de depurar.
   * El uso de un controlador centralizado para manejar eventos evita redundancias y hace el comportamiento del sistema más consistente.
4. **Escalabilidad:**
   * La estructura modular del código facilita la adición de nuevas funcionalidades en el futuro, como otros eventos de interacción o nuevas características para el juego.
   * La validación de entradas y manejo de errores asegura que el código pueda manejar diferentes escenarios sin interrumpir la experiencia del usuario.

# Recomendaciones

1. **Automatización del Formato:**
   * Utilizar herramientas como **Prettier** o **ESLint** para garantizar un formato consistente en el código y detectar errores comunes.
2. **Casos de Prueba:**
   * Diseñar e implementar más casos de prueba funcionales y de calidad, enfocándose en:
     + Navegación por teclado.
     + Mensajes de error claros y útiles.
     + Evaluación de rendimiento con interacciones simultáneas.
3. **Documentación Adicional:**
   * Crear una documentación clara y detallada del código que incluya:
     + Explicaciones de las funciones clave.
     + Diagramas de flujo para mostrar la lógica del juego.
     + Ejemplos prácticos de cómo extender la funcionalidad.
4. **Optimización de Recursos:**
   * Analizar el uso de recursos del navegador y optimizar el código para garantizar que funcione de manera eficiente incluso en dispositivos de baja gama.
5. **Pruebas de Usabilidad:**
   * Realizar pruebas con usuarios reales para identificar problemas de interacción y mejorar aspectos como la gamificación, navegación intuitiva y retroalimentación visual.
6. **Escalabilidad:**
   * Si el proyecto crece, considerar la posibilidad de implementar una arquitectura basada en componentes o bibliotecas modernas como **React** o **Vue.js**, que faciliten la gestión del estado y el desarrollo modular.
7. **Manejo de Versiones:**
   * Utilizar un sistema de control de versiones como **Git** para rastrear cambios en el código y facilitar la colaboración en equipo.