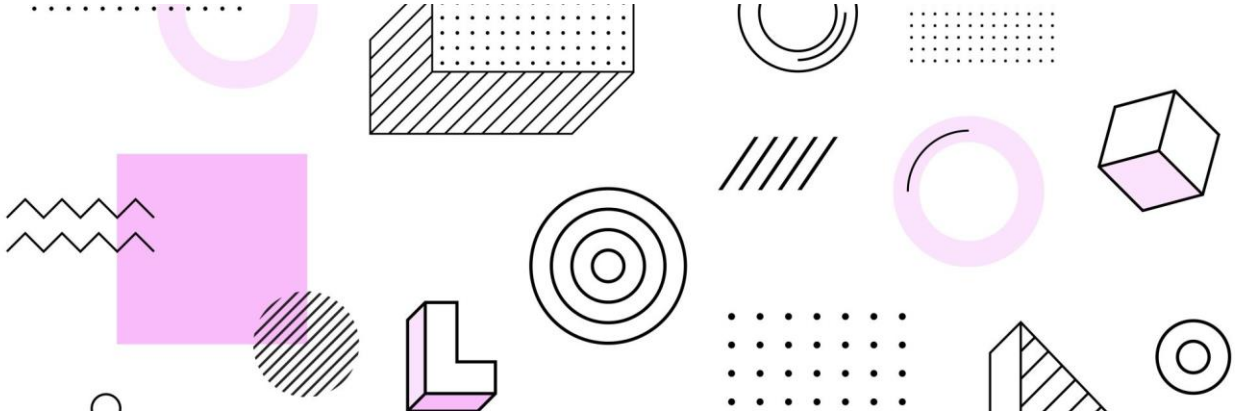

Taller de Nivelación Curso Desarrollo Web Frontend

El presente taller tiene el propósito de evaluar la comprensión y aplicación de conceptos fundamentales del desarrollo web Frontend desde los conceptos de lógica de programación, HTML, CSS, JavaScript, React Js, los diferentes tipos de Hooks, su uso adecuado y cómo utilizar React Router DOM v6 para la navegación en una aplicación React. Se busca nivelar y asegurar la apropiación de los conceptos vistos hasta el momento en el curso de desarrollo web frontend, la implementación de ReactJS y sus tecnologías asociadas.

MÓDULO SOBRE LÓGICA, LÓGICA DE PROGRAMACIÓN Y PROGRAMACIÓN CON JAVASCRIPT

Preguntas teóricas

1. ¿Qué es la lógica en el contexto de la programación? Y explicar por qué es importante en el desarrollo web Frontend.
2. Definir el concepto de “algoritmo” y proporcionar un ejemplo sencillo de un algoritmo relacionado con la lógica de programación.
3. ¿Qué son estructuras de control en la programación?, ¿Cuáles son los tipos de estructuras de control y las estructuras más comunes de cada tipo?, Describir al menos dos tipos de estructura de control, explicar por qué son importantes y proporcionar ejemplos de cada uno de cómo se utilizan en el desarrollo web Frontend.
4. Describir cómo se declaraban variables y constantes en JavaScript antes de la introducción de ECMAScript 6 (ES6). Explicar cómo ES6 mejoró la declaración de variables y constantes, y mencionar los problemas que esta mejora resuelve en el desarrollo web Frontend.
5. ¿Cómo se declaran las funciones en JavaScript y cuál es la diferencia entre una declaración de función, una expresión de función y una función de flecha (arrow function)? Proporcionar ejemplos de cada una.
6. ¿Por qué es necesario el uso de funciones en el desarrollo web Frontend? Enumerar al menos tres razones fundamentales y proporcionar ejemplos de situaciones en las que las funciones son esenciales. Además, mencionar la ventaja de las funciones flecha en el contexto de estas razones.
7. ¿Cuál es la diferencia entre parámetro y argumento?
8. Definir el concepto de Callback y proporcionar un ejemplo práctico.

9. ¿Qué es el hoisting en JavaScript y cómo afecta a las variables y funciones? Proporcionar ejemplos de hoisting en declaraciones de variables y funciones.
10. Definir brevemente el concepto de objeto en JavaScript y cuál es la visión general sobre este concepto. Indicar, también cómo se declaran estas estructuras de datos.
11. ¿Qué son propiedades?, y ¿Cuál es la diferencia entre una propiedad y un método en un objeto?
12. Explicar las dos formas de acceder a una propiedad de objetos e indicar las situaciones en que conviene usar una manera sobre la otra.
13. ¿Existe alguna forma de recorrer las propiedades de un objeto? En caso negativo, explicar por qué no es posible y en caso positivo proporcionar un ejemplo. Mencionar una situación en la cual sea muy útil recorrer las propiedades de un objeto.
14. ¿Por qué son útiles los objetos en la programación web y qué tipos de datos pueden almacenar?
15. ¿Qué es un array en JavaScript y por qué son esenciales?
16. ¿Cómo acceder a un elemento dentro de un array? Explicar con un ejemplo.
17. Mencionar al menos tres funciones de arrays y describir su utilidad en la programación web.
18. Proporcionar un ejemplo de cómo se utiliza una función de array para transformar y filtrar datos en un array.

Preguntas prácticas

1. Escribir un programa con JavaScript que resuelva el siguiente problema: Dada una lista (o array) de números enteros, encontrar el número más grande de la lista y mostrarlo en consola. No se debe usar la función `Math.max()`, ni `.forEach()`.
2. Escribir una función en JavaScript que tome dos números como argumentos y devuelva su suma. Luego, escribir la misma función utilizando una función flecha (arrow function) y comparar ambas declaraciones. Llamar a ambas funciones con valores de ejemplo y mostrar los resultados en la consola del navegador.
3. Escribir una función en JavaScript que reciba un array de números como argumento y utilizar la función de array para calcular la suma de todos los números pares en el array. Luego, llamar a la función con un array de ejemplo y mostrar el resultado en la consola del navegador.

MÓDULO SOBRE HTML, CSS Y RESPONSIVE DESIGN

Preguntas teóricas

1. ¿Qué significa HTML y cuál es su función en el desarrollo web?
2. ¿Cuál es la estructura básica de un documento HTML? Describir las etiquetas esenciales.
3. ¿Qué es CSS y cuál es su propósito en el desarrollo web?
4. ¿Qué son selectores CSS, cuáles son los principales tipos de selectores y porqué es importante entender la especificidad en el contexto de las hojas de estilo en cascada (CSS)? Describir al menos tres tipos de selectores CSS y cómo la especificidad afecta a la aplicación de estilos en un proyecto de desarrollo web Frontend. Proporcionar ejemplos de situaciones en las que se utiliza la especificidad de selectores para resolver conflictos de estilos.

5. Explicar las diferencias entre los estilos en línea (inline), estilos internos (embedded) y estilos externos (external) en CSS. Indicar cuál de los tres estilos es el recomendado usar y por qué.
6. ¿Qué es flexbox y cómo se utiliza para el diseño de páginas web?
7. Explicar cómo se emplean las propiedades flexbox y explicar la función de las principales propiedades en la creación de diseños flexibles.
8. ¿Qué es CSS Grid Layout y en qué se diferencia de flexbox?
9. Proporcionar un ejemplo de cómo crear una cuadrícula sencilla con CSS Grid.
10. ¿Qué significa el diseño responsivo en el contexto del desarrollo web?
11. Enumerar al menos tres técnicas o estrategias utilizadas para lograr el diseño responsivo en una página web.

Preguntas prácticas

1. Crear un documento HTML llamado index.html que incluya encabezados, párrafos, enlaces y una lista no ordenada. Aplicar la práctica recomendada de estilo CSS para cambiar el color y la fuente de los elementos.
2. Diseñar una sección de la página web en index.html que contenga tres elementos (por ejemplo, tarjetas). Utilizar flexbox para alinearlos horizontalmente y distribuir el espacio de manera uniforme.
3. Crear una cuadrícula de imágenes (o galería de imágenes) utilizando CSS Grid. Asegurarse de que las imágenes se ajusten automáticamente a diferentes tamaños de pantalla.
4. Modificar la página web para que sea responsiva. Asegurarse de que los elementos se reorganicen y se ajusten correctamente cuando el tamaño de pantalla del navegador cambie.
5. Agregar media queries a la página para personalizar el diseño en dispositivos móviles. Lograr que los estilos cambien cuando la pantalla sea más estrecha.

MÓDULO SOBRE DOM E INTERACCIÓN CON EL DOM

Preguntas teóricas

1. ¿Qué es el DOM (Modelo de Objeto de Documento) en el contexto de la programación web?
2. ¿Cuál es la diferencia entre el DOM y el HTML en una página web?
3. ¿Porqué es importante entender y manipular el DOM en el desarrollo web Frontend?
4. ¿Qué son los eventos del DOM y cuáñ es su función en una página web?
5. Proporcionar ejemplos de eventos prácticos y comunes, como “click”, “submit” y “load o DOMContentLoad”.
6. ¿Por qué es importante manejar eventos en la interacción usuario-web y cómo se añaden controladores de eventos a los elementos del DOM?
7. Describir al menos tres métodos para seleccionar elementos del DOM en JavaScript.
8. ¿Cómo se crea un nuevo elemento HTML y se agrega al DOM utilizando JavaScript?
9. ¿Cuál es la importancia de la manipulación del DOM en la creación de aplicaciones web dinámicas e interactivas?
10. Explicar brevemente los conceptos “event bubbling” y “event delegation” en el contexto de eventos del DOM.

11. ¿Por qué son relevantes los conceptos “event bubbling” y “event delegation” en la gestión de eventos en páginas web con múltiples elementos interactivos?

Preguntas prácticas

1. Escribir una función en JavaScript que tome un botón en el DOM y, al hacer click en él, cambie el color de fondo de un elemento específico en la página. Luego, agregar el botón y el elemento objetivo en el DOM y asociar la función al evento “click”.
2. Crear una lista no ordenada de elementos (por ejemplo, elementos de lista) en el DOM. Implementar la delegación de eventos (event delegation) para que, al hacer clic en cualquier elemento de la lista, se muestre un mensaje en la consola que indique qué elemento de la lista se ha hecho clic.
3. Agregar un formulario a tu página web con un campo de entrada y un botón "Enviar". Implementar una función que sea llamada al enviar el formulario y que valide el campo de entrada (por ejemplo, si está vacío), muestre un mensaje de error accesible si es necesario y en caso de que el formulario esté correctamente diligenciado muestre en consola un objeto con el dato que ha ingresado el usuario en el campo de entrada y un alert con el siguiente mensaje: “Formulario correctamente diligenciado”.

MÓDULO SOBRE COMUNICACIÓN CON EL SERVIDOR (STORAGE, PROMESAS, ASINCRONÍAS Y PETICIONES HTTPS)

Preguntas teóricas

1. Definir brevemente el concepto de localStorage y sessionStorage.
2. Describir las diferencias claves entre localStorage y sessionStorage.
3. ¿Por qué son útiles para almacenar datos en el navegador y cuál es su límite de capacidad?
4. ¿Qué son las promesas en JavaScript y para qué se utilizan en el desarrollo web?
5. Explica el concepto de asincronía en programación y cómo las promesas ayudan a manejar operaciones asincrónicas.
6. Proporciona un ejemplo de cómo se utiliza una promesa para realizar una operación asincrónica, como una solicitud de red.
7. ¿Qué es JSON Server y cómo se utiliza en el desarrollo web?
8. ¿Por qué es útil simular una API REST falsa con JSON Server en el desarrollo frontend?
9. ¿Cuáles son las diferencias claves entre los métodos del objeto promesa `.then().catch()` y las palabras claves `async/await`?
10. Proporciona un ejemplo de cómo configurar una API falsa con JSON Server y realizar solicitudes (GET, POST, PUT, PATCH y DELETE) a través de ella.
11. Describe las diferencias entre Fetch y Axios como métodos para realizar solicitudes HTTP en JavaScript.
12. ¿Por qué es importante considerar las peticiones HTTP en aplicaciones web modernas?
13. Proporciona ejemplos de cómo se utilizan Fetch y Axios para realizar solicitudes GET y POST.
14. Explica la importancia del manejo de errores al trabajar con promesas en el desarrollo web.
15. Describe cómo se manejan los errores en las promesas, incluyendo el uso de catch.
16. ¿Cuáles son las diferencias claves entre los métodos del objeto promesa `.then().catch()` y la estructura `try/catch`?

17. Proporciona un ejemplo de cómo se puede manejar un error en una promesa al realizar una solicitud de red.

Preguntas prácticas

1. En una sección de la página web construida en los módulos anteriores, permitir a un usuario almacenar y recuperar datos utilizando `localStorage` y `sessionStorage`. Demostrar cómo se puede guardar y recuperar datos de estas áreas de almacenamiento del navegador.
2. Escribir una función que utilice una promesa para simular una operación asincrónica, como, por ejemplo, una solicitud de datos. Luego, mostrar los resultados de la promesa en una sección en la página web.
3. Crear una API falsa con JSON Server y realizar una solicitud GET y POST con Fetch o Axios y mostrar los resultados en una sección de la página web.
4. Crear un repositorio remoto en GitHub con la estructura de carpetas y archivos de la página web creada para darle respuesta a las preguntas anteriores.
5. La respuesta a las preguntas teóricas debe estar resueltas en un archivo README.md en el repositorio creado en el ítem anterior.
6. Desplegar la página web en GitHub pages.

MÓDULO SOBRE REACT JS

Preguntas teóricas

1. Explicar brevemente el concepto de ReactJS y sus principales características.
2. Definir qué es un componente en ReactJS y mencionar los tipos de componentes que existen.
3. ¿Qué es el Virtual DOM y cuál es su función en ReactJS?
4. ¿Qué es JSX en ReactJS y porqué es importante?
5. ¿Qué es un Hook en ReactJS y cuál es su propósito?
6. Mencionar al menos tres tipos de Hooks en ReactJS y explicar brevemente para qué se utilizan.
7. ¿Cuáles son las reglas de uso para los Hooks en ReactJS?
8. Explicar qué es React Router DOM versión 6, para qué se utiliza y cuáles son sus principales componentes y Hooks.
9. Explicar cómo se realiza la navegación entre diferentes páginas utilizando React Router DOM.
10. ¿Cómo se definen rutas en React Router DOM?
11. Describir cómo crear un proyecto ReactJS con Vite
12. Describir cómo desplegar un JSON server en cualquier Hosting free o servicio en la nube gratuito de su elección
13. Describir cómo desplegar una aplicación en cualquier Hosting de su elección.

Preguntas prácticas

1. Crear un proyecto React con vite llamado **taller-refuerzo-<su nombre>-<su primer apellido>** y eliminar los archivos y elementos que suministra la template que no son requeridos para el desarrollo de los siguientes ítems.

2. Organizar el nuevo proyecto y crear la estructura de carpetas y archivos sugerida en clase para mantener buenas prácticas de desarrollo.
3. Crear un componente funcional en React llamado '**MiComponente**' que renderice un párrafo con el texto "*Hola, soy un componente funcional de React"*
4. Crear la ruta '**/micomponente**' que renderice el componente '**MiComponente'**
5. Utilizar el Hook '`useState`**'** para manejar un estado '**contador**' y mostrarlo en un componente llamado '**MiContador**'. Se debe incluir los botones para incrementar y decrementar el contador, así como la ruta '**/micontador'** que renderice este componente.
6. Levantar un json server con la data que se encuentra en el archivo db.json y despliegarlo en un servicio en la nube gratuito de su elección.
7. Utilizar el Hook **'**`useEffect`**'** para cargar datos del API simulado en el punto anterior, cuando un componente llamado '**MisProductos**' se monte. Este componente debe mostrar en cards cada dato que obtenga del API. Asimismo, se debe crear la ruta **'/misproductos'** para este componente.
8. Permitr filtrado múltiple por **categoría** y **rating** de productos en el componente **MisProductos**.
9. Crear una ruta dinámica que permita mostrar los detalles de un producto seleccionado. Por ejemplo, '**/miproducto/1**' debería mostrar los detalles del producto con id 1 en un componente llamado '**DetallesProducto**'.
10. Crear navbar que permita la navegación a todas las páginas construidas.
11. Desplegar la aplicación React en GitHub Pages

MÓDULO SOBRE GESTION DE ESTADOS Y DATOS CON REACT CONTEXT Y USEREDUCER

Preguntas teóricas

1. ¿Qué es `React Context` y para qué se utiliza en el desarrollo web con React?
2. Describir cómo se crea un contexto en React y cómo se proporciona y consume información a través de él.
3. ¿Cuál es la ventaja de utilizar `React Context` en lugar de pasar `props` a través de múltiples componentes?
4. Explicar el propósito de `useReducer` en React y cómo se diferencia de `useState`.
5. Describe los argumentos que toma la función `useReducer`.
6. ¿Por qué es útil utilizar useReducer para gestionar el estado en aplicaciones más complejas?
7. ¿Cómo se puede utilizar `React Context` junto con `useReducer` para gestionar el estado global en una aplicación de React?
8. ¿Por qué es importante tener un sistema de gestión de estado global en aplicaciones React más grandes?
9. Menciona al menos tres ventajas de utilizar una combinación de `React Context` y `useReducer` en comparación con el manejo de estado local en componentes.
10. ¿En qué situaciones podría ser beneficioso migrar de un enfoque de manejo de estado local a un enfoque de estado global utilizando `React Context` y `useReducer`?

Preguntas prácticas

1. Implementar un contador utilizando `useReducer` en lugar de `useState`. Crear acciones para aumentar y disminuir el contador y mostrar el valor en un componente llamado **'MiContadorConUseReducer'**. Este componente debe ser renderizado en la ruta **'/miContadorUseReducer'** en el aplicativo web desarrollado en el módulo anterior (sobre React Js).
2. En la aplicación web construida en el módulo anterior (sobre React Js), implementar protección de rutas y autenticación de usuarios, creando un contexto que comparta la información del usuario logueado y un estado con `useReducer()` que almacene la información del usuario logueado. Las rutas definidas en el módulo y paso anterior deben permanecer públicas, las rutas o páginas protegidas serán las construidas en el siguiente ítem.
3. Desarrollar un componente llamado **'MiToDoList'** que utilice un contexto y `useReducer` para gestionar el estado de las tareas. Este componente debe permitir agregar, eliminar tareas, marcar tareas como completadas y filtrar tareas por status (realizada, pendiente). Asimismo, 'MiToDoList' debe ser renderizado en la ruta **'/miTodoList'** y tal como fue indicado en el ítem anterior esta ruta debe estar protegida.

   Nota: para desarrollar los puntos dos y tres del presente módulo, se debe crear y desplegar un JSON Server con los endpoints con el listado de usuarios y tareas.

Productos entregables:

1. Repositorio de JSON server en GitHub
2. URL de despliegue del JSON server
3. Repositorio del componente práctico en GitHub
4. URL de despliegue de la aplicación en GitHub Pages
5. Respuestas del componente teórico en un archivo README.md dentro del repositorio creado para el ítem 3.

Primera entrega: 17 de noviembre del 2023

- Primeros 4 módulos

Segunda entrega: 27 de noviembre del 2023

- Último módulo

Reglas y Restricciones para resolver el taller de nivelación:

1. Cada respuesta a las preguntas teóricas no debe superar las 200 palabras. Si es necesario extender el límite, se debe justificar por qué luego de dar la respuesta. En lo posible, ser claros y concisos en la respuesta.
2. Prohibido realizar copia. Todas las respuestas deben ser originales y escritas desde el entendimiento de cada uno. Si es necesario copiar, se debe realizar cita del recurso. Esta restricción aplica también para las preguntas prácticas.
3. En lo posible no demorar más de 30 minutos en darle respuesta a las preguntas conceptuales.