Desarrollo FrontEnd

Interfaz de Usuario

HTML: https://www.w3schools.com/html/
CSS: https://www.w3schools.com/css/

Desarrollo FrontEnd

JavaScript: https://www.w3schools.com/js/
React

Interfaz de Usuario

- Medio con que el usuario puede comunicarse con el software
- El flujo comienza en el área de diseño antes de llegar al área de informática
- HTML + CSS

Interfaz de Usuario

Elementos más importantes de la portada de un website:

- Dejar claro el propósito del sitio
- Ayudar a los usuarios a encontrar lo que necesitan
- Usar diseño visual para mejorar y no para definir la interacción del Sitio Web

HTML

- Lenguaje de marcado que define el contenido de las páginas web

- Se compone en base a etiquetas

HTML

Etiquetas principales:

- head: cómo los clientes deben interpretar la página
- body: lo que se verá en la página

HTML

```
<!DOCTYPE html>
<html>
    <head>
        <title>Título de la página web</title>
        ...
    </head>
    <body>
        Cuerpo de la página web
    </body>
</html>
```

HTML: head

Etiqueta en cabecera	Función	¿Es obligatoria?
<title>	Da un título al documento HTML	Sí
<base>	Define ruta de acceso	No
<link>	Define archivos vinculados	No
<meta>	Define metadatos como descripción y palabras clave	No
<script>	Delimita scripts incluídos	No
<style>	Delimita definición de estilos	No

HTML: body

- división dentro del contenido
- <a> para enlaces. Atributos: target, href
- para poner el texto en negrita
- para listas ordenadas, <ul> para listas sin orden, <li> para los elementos de las listas
- <h1>...<h6> para títulos dentro del contenido
- <img> para añadir imágenes al documento
- <p> para párrafos
- para estilos de una parte del texto

HTML: ejemplo

```
<body>

<div style="font-size: 30px">The div element</div>

<div style="background-color: lightblue; text-align: center">
  <h2>This is a heading in a div element</h2>
  <p>This is some text in a div element.</p>
</div>

</body>
```

HTML: ejemplo

The div element

This is a heading in a div element

This is some text in a div element.

CSS

- Hojas de estilo en cascada
- Define y crea la presentación de un HTML
- Reemplaza al atributo “style” de los HTML

CSS: propiedades

- font-family: define la familia tipográfica
- font-size: define el tamaño de la fuente
- color: define el color de la tipografía
- width: define el ancho de un elemento
- max-width/min-width: definen el ancho mínimo o máximo de un elemento
- height: define el alto de un elemento
- max-height/min-height: definen el alto mínimo o máximo de un elemento

CSS: propiedades

- padding: define la distancia desde el borde de un elemento hasta su contenido
- margin: define la distancia entre un elemento y otro
- border: define el borde de un elemento (color, estilo y grosor)
- background: define los fondos de un objeto (imagen o color)
- display: define el tipo de caja (none, block, flex, etc)

CSS: ejemplo

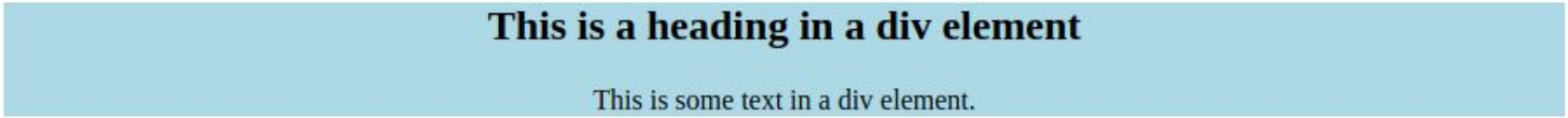
```
<body>

<div class="title">The div element</div>

<div class="myDiv">
  <h2>This is a heading in a div element</h2>
  <p>This is some text in a div element.</p>
</div>

</body>
```

CSS: ejemplo

```
.myDiv {
  background-color: lightblue;
  text-align: center;
}
.title {
    font-size: 30px;
}
```

CSS: ejemplo

The div element



JavaScript

- Se ejecuta en el navegador
- Es un lenguaje de tipado dinámico
- Se complementa con HTML y CSS, agregándoles funcionalidad
- Es un lenguaje de programación interpretado

JavaScript: declaración de variables

- var: ámbito de función
- let: ámbito de bloque
- const: ámbito de bloque. No se pueden reasignar

JavaScript: var

```
var i = "global";

function foo() {

    i = "local";

    console.log(i); // local

}

foo();

console.log(i); // local
```

JavaScript: var

```
var i = "global";

function foo() {

    var i = "local";

    console.log(i); // local

}

foo();

console.log(i); // global
```

JavaScript: let

```
let i = 0;
function foo() {
        i = 1;
        let j = 2;
        if (true) {
                console.log(i) // 1
                console.log(j) // 2
        }
}
foo();
```

JavaScript: let

```
function foo() {
        let i = 0;
        if (true) {
                let i = 1;
                console.log(i) // 1
        }
console.log(i) // 0
}
foo();
```

JavaScript: const

```
const i = 0;
i = 1; // TypeError: Assignment to constant variable
```

JavaScript: DOM Selector

- let elementByClass = document.querySelector(".miClase")
- let elementById = document.querySelector("#id")
- let elementListByClass = document.querySelectorAll(".miClase")

JavaScript: eventos

- Como atributos dentro del HTML tag:

Clickeable block

- Con DOM Selector:

Clickeable block

document.querySelector(“#clickeable-block”).addEventListener(“click”, function() { console.log(“click”) })

JavaScript: callbacks

Funciones que se pasan a otras funciones como argumentos

- function showAlert(){ alert('Alerta'); }

  button.addEventListener('click', showAlert);

- function showAlert(alert_type){ alert(alert_type); }

  button.addEventListener('click', function() {

  showAlert(“Alerta”)});

Próxima clase

- **React**
- Vanilla (sin framework)
- Angular

Frameworks más populares

- **React**
- Angular
- Vue
- Vanilla (sin framework)

Angular

- Orientado principalmente al desarrollo de SPA (Single Page Application) utilizando MVC
- Uso de TypeScript integrado
- Enlace bidireccional de datos (2 way binding)
- Incorpora el uso de directivas (de atributo, estructurales, componentes)

Vue

- Simplifica el funcionamiento de Angular, llegando a tener una curva de conocimiento muy leve
- Su modelo define el dato como centro de todo
- Componentes con código reutilizable
- Comunicación entre componentes por medio de eventos
- Virtual DOM

React

- Fácil de aprender con conocimientos de html, css y javascript
- Desarrollo orientado a componentes
- Virtual DOM
- JSX

```
const name = 'Juan Perez';

const element = <h1>Hello, {name}</h1>;
```

React: Virtual DOM

- Actualiza solamente lo que es necesario
- Buena performance

React: componentes

- Contienen código reutilizable
- Aceptan parámetros (props)
- Renderizan una pequeña porción de código
- El renderizado puede ser condicional
- Devolver null si no se quiere renderizar un componente

React: componentes de clase

- Se definen como clases que heredan de React.Component
- Tienen estado (state)
- Las props son utilizadas como atributos de la clase: this.props.{{prop}}
- Se utiliza el método render para indicar qué mostrar en pantalla

React: componentes de función

- Más sencillos que los de clase
- Utilizables (no obligatorios) cuando el componente es stateless. Sin embargo, pueden tener estado
- En lugar de usar el método render, el html de retorno se devuelve directamente con return
- Las props no son tratadas como atributos sino como parámetro de la función: props.{{prop}}

React: componentes de función

```
export default function Profile() {
  return (
    <img
      src="https://i.imgur.com/MK3eW3Am.jpg"
      alt="Katherine Johnson"
    />
  )
}
```

React: props

- Parámetros que se pasan a los componentes
- Pueden ser callbacks
- Read only
- Los componentes de clase siempre deben invocar al constructor base con props: super(props)

React: props

```
export default function Profile() {
  return (
    <Avatar
      person={{ name: 'Lin Lanying', imageId: '1bX5QH6' }}
      size={100}
    />
  );
}
```

```
function Avatar({ person, size }) {
  return (
    <img
      className="avatar"
      src={getImageUrl(person)}
      alt={person.name}
      width={size}
      height={size}
    />
  );
}
```

React: state (componentes de clase)

- Estado que tiene cada componente
- Es un objeto cuyos valores iniciales deben setearse en el constructor del componente
- Para actualizar el estado se utiliza setState (IMPORTANTE)

React: state (componentes de función)

```
import { useState } from 'react';
import { sculptureList } from './data.js';

export default function Gallery() {
  const [index, setIndex] = useState(0);

  function handleClick() {
    setIndex(index + 1);
  }

  let sculpture = sculptureList[index];
  return (
    <>
      <button onClick={handleClick}>
        Siguiente
      </button>
      <h2>
        <i>{sculpture.name} </i>
        por {sculpture.artist}
      </h2>
      <h3>
        ({index + 1} de {sculptureList.length})
      </h3>
      <img
        src={sculpture.url}
        alt={sculpture.alt}
      />
      <p>
        {sculpture.description}
      </p>
    </>
  );
}
```

React: ciclo de vida (componentes de clase)

- componentDidMount: se ejecuta después de que el componente se haya renderizado en el DOM
- componentWillUnmount: se utiliza para liberar recursos al borrarse el DOM producido por el componente
- componentDidUpdate: se ejecuta cuando se modifica el estado de un componente

React: ciclo de vida (componentes de función)

```
import { useEffect } from 'react';
import { createConnection } from './chat.js';

function ChatRoom({ roomId }) {
  const [serverUrl, setServerUrl] = useState('https://localhost:1234');

  useEffect(() => {
    const connection = createConnection(serverUrl, roomId);
    connection.connect();
    return () => {
      connection.disconnect();
    };
  }, [serverUrl, roomId]);
  // ...
}
```

React: hooks

- Permiten usar el estado y otras características de React sin escribir una clase
- Solo utilizables dentro de componentes funcionales
- Solo utilizables en el nivel superior (NO dentro de loops, condiciones o funciones anidadas)
- Pueden ser custom

React: hooks predefinidos

- useState para inicializar estado
- useEffect tiene el mismo propósito que los tres métodos de ciclo de vida de un componente

React: hooks custom

```
function useOnlineStatus() {
  const [isOnline, setIsOnline] = useState(true);
  useEffect(() => {
    function handleOnline() {
      setIsOnline(true);
    }
    function handleOffline() {
      setIsOnline(false);
    }
    window.addEventListener('online', handleOnline);
    window.addEventListener('offline', handleOffline);
    return () => {
      window.removeEventListener('online', handleOnline);
      window.removeEventListener('offline', handleOffline);
    };
  }, []);
  return isOnline;
}
```

```
import { useOnlineStatus } from './useOnlineStatus.js'

function StatusBar() {
  const isOnline = useOnlineStatus();
  return <h1>{isOnline ? '✅ Conectado' : '❌ Descone
}

function SaveButton() {
  const isOnline = useOnlineStatus();

  function handleSaveClick() {
    console.log('✅ Progreso guardado');
  }

  return (
    <button disabled={!isOnline} onClick={handleSaveCl
      {isOnline ? 'Guardar progreso' : 'Reconectando..
    </button>
  );
}

export default function App() {
  return (
    <>
      <SaveButton />
      <StatusBar />
    </>
  );
}
```

React: rutas

- npm install react-router-dom
- Se utiliza para renderizar un componente dependiendo del path al cual se accede
- Se puede pasar de un componente a otro con this.history.push(“/newpath”)

React: rutas

```jsx
import ReactDOM from "react-dom/client";
import { BrowserRouter, Routes, Route } from "react-router-dom";
import Layout from "./pages/Layout";
import Home from "./pages/Home";
import Blogs from "./pages/Blogs";
import Contact from "./pages/Contact";
import NoPage from "./pages/NoPage";

export default function App() {
  return (
    <BrowserRouter>
      <Routes>
        <Route path="/" element={<Layout />}>
          <Route index element={<Home />} />
          <Route path="blogs" element={<Blogs />} />
          <Route path="contact" element={<Contact />} />
          <Route path="*" element={<NoPage />} />
        </Route>
      </Routes>
    </BrowserRouter>
  );
}

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<App />);
```

React vs Angular

React	**Angular**
Virtual DOM Actualiza solo lo necesario.	Regular DOM. Actualiza toda la estructura hasta llegar a lo que hay que modificar
Con tener conocimientos de JS es suficiente	Se necesita familiarizarse con una nueva sintaxis
One-Way binding Cambio en estado -> Cambio en UI	Two-Way Binding: Cambio en estado <-> Cambio en UI

Hands On:
Creamos un proyecto desde cero

React: instalación

- node (v >= 14)
- npm
- nvm (recomendable)

React: instalación

- create-react-app para armar automáticamente la estructura de un proyecto React
- npm install {{libreria}} para instalar librerias externas
- package-json tiene todas las dependencias externas y los scripts que se pueden ejecutar
- npm run start para correr la aplicación

Preguntas?

Próxima clase: ejemplo práctico

Links útiles

Tutorial de React: https://es.react.dev/learn

Material UI: https://mui.com/getting-started/usage/

¡Muchas gracias!