Logotipo

El contenido generado por IA puede ser incorrecto.

Santiago Yosa González

**Ingeniero Sistemas de Información**



Logotipo

El contenido generado por IA puede ser incorrecto.}

Pacto Pedagógico

Presentación del curso

Exploración comparativa entre sitios tradicionales y SPA

Instalación guiada de Node.js y NPM

Creación del proyecto inicial con React y Vite.

**Unidad I**

**Estructuración de Contenidos**

**Situación inicial de introducción**

**Analogía del Restaurante:** Imagina dos tipos de restaurantes para explicar la evolución de la web:

* **El Restaurante Clásico (MPA - Multi Page Application):** Cada vez que quieres un vaso de agua, tienes que levantarte, salir del restaurante, volver a entrar y sentarte en una mesa nueva que ya tiene el agua. Es ineficiente y lento. (Así funcionan los sitios tradicionales: cada clic recarga toda la página y trae todo el HTML nuevo desde el servidor).
* **El Servicio a la Mesa (SPA - Single Page Application):** Te sientas una sola vez. El mesero (JavaScript) te trae el agua, luego la comida, luego el postre, sin que tú te muevas de tu silla. La experiencia es fluida e instantánea. (Así funciona React: cargamos el sitio una vez y solo actualizamos lo que el usuario necesita ver).

**Arquitectura Web: Tradicional vs. SPA**

**MPA (Multi Page Application):**

El ciclo de vida es controlado por el servidor. El navegador hace una petición GET, el servidor renderiza el HTML completo y lo envía. Si el usuario hace clic en "Contacto", el navegador destruye la página actual y repite el ciclo completo.

* *Desventaja:* "Flash" de pantalla blanca, mayor consumo de ancho de banda repetitivo.

**SPA (Single Page Application):**

El servidor envía un solo archivo HTML (casi vacío) y un gran archivo de JavaScript (el "bundle"). El navegador ejecuta ese JS, y es este quien construye la interfaz visualmente (DOM). Cuando el usuario navega, JavaScript intercepta la acción y actualiza solo esa sección de la pantalla, sin recargar.

* *Concepto Clave:* **Client-Side Rendering (CSR)**. El navegador hace el trabajo pesado de "pintar", no el servidor.

**El Motor y la Caja de Herramientas: Node.js y NPM**

**¿Por qué instalar Node.js si React es para el navegador?**

Porque las herramientas modernas (como Vite o React) necesitan compilar, optimizar y empaquetar archivos antes de subirlos a la web. El navegador no entiende archivos .jsx o .vue directamente; Node.js nos ayuda a traducirlos a HTML, CSS y JS estándar.

**Node.js (El Entorno):** JavaScript fue diseñado para vivir *dentro* del navegador (encerrado en una caja de arena por seguridad). Node.js libera a JavaScript del navegador, permitiéndole acceder al sistema de archivos, redes y procesador de tu computadora.

* *En desarrollo Frontend:* No usamos Node para crear un servidor web de producción (todavía), lo usamos como una **herramienta de construcción**. Necesitamos Node para compilar, minificar y optimizar nuestro código antes de subirlo a internet.

**NPM (Node Package Manager):** Es el registro de software más grande del mundo.

* *Dependencias:* Son códigos que otros escribieron (librerías, frameworks) para que no reinventemos la rueda.
* *package.json:* Es el "DNI" o "Manifiesto" de tu proyecto. Dice cómo se llama, qué versión es y qué librerías necesita para funcionar.

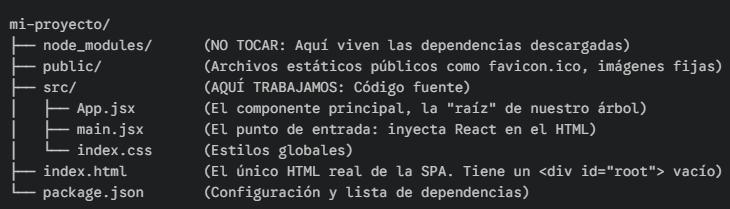
**React y Vite: El Framework y el Empaquetador**

**React:** Es una librería (aunque la tratamos como framework) basada en **Componentes**. En lugar de escribir una página web gigante, escribes pequeñas piezas (un botón, un formulario, un encabezado) y las ensamblas como LEGO.

* ***Virtual DOM****:* React mantiene una copia virtual de la estructura de la página. Cuando algo cambia, compara la copia virtual con la real y solo actualiza lo estrictamente necesario. Esto lo hace extremadamente rápido.

**Vite (pronunciado "veet", rápido en francés):** Antiguamente usábamos *Webpack*, que era lento y complejo para configurar. Vite es la herramienta de construcción moderna.

* ***HMR (Hot Module Replacement):*** Permite que, al guardar un archivo, el cambio se refleje en el navegador instantáneamente sin recargar toda la página, manteniendo el estado de la aplicación (si escribiste en un input, el texto sigue ahí).



**Preguntas Frecuentes**

**Si React es JavaScript, ¿por qué el navegador no lo entiende directamente y necesito compilarlo?**

Los navegadores entienden JavaScript estándar (ES6+), pero React utiliza una sintaxis especial llamada **JSX** (JavaScript XML) que mezcla HTML dentro de JS. Los navegadores no saben qué hacer con <MiComponente />. Herramientas como Vite (usando ESBuild bajo el capó) transforman ese JSX en JavaScript puro que el navegador sí entiende.

**¿Qué es la carpeta node\_modules y por qué es tan pesada?**

Es donde se descargan físicamente todas las librerías listadas en tu package.json y, a su vez, las librerías que esas librerías necesitan (dependencias transitivas). **Nunca** se debe subir a GitHub ni modificar manualmente. Si la borras, siempre puedes regenerarla con el comando npm install.

**Anatomía del Código: package.json**

Texto

El contenido generado por IA puede ser incorrecto.

**Ejemplo 1: HTML + JS Tradicional (Imperativo) *Le ordenas al navegador paso a paso qué cambiar.***

<button id="btn">Click me</button>

<p id="texto">Hola</p>

<script>

// JS manipula directamente el DOM

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

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

btn.addEventListener('click', () => {

texto.innerText = "Adios"; // Cambio directo

texto.style.color = "red";

});

</script>

**Ejemplo 2: React (Declarativo) *Declaras cómo debe verse la interfaz según un estado, React se encarga del "cómo".***

// JSX

import { useState } from 'react';

function App() {

// Estado: la fuente de la verdad

const [mensaje, setMensaje] = useState("Hola");

return (

<div>

<p>{mensaje}</p>

{/\* Solo cambiamos el dato, React actualiza la vista \*/}

<button onClick={() => setMensaje("Adios")}>

Click me

</button>

</div>

);

}

**Verificar si tenemos el motor (Node.js)**

Vite **no funciona** sin Node.js. Es como intentar arrancar un auto sin motor.

Abre la terminal (Símbolo del sistema / PowerShell).

Escribe: node -v

**Escenario A :**

Aparece algo como v18.17.0 o superior.

**Escenario B :**

Aparece *"node no se reconoce como un comando interno..."* o una versión muy vieja (ej. v12.x).

**Instalación de Node.js (Solo si falló el paso anterior):**

Ir a [nodejs.org](https://nodejs.org).

Descargar la versión **LTS (Long Term Support)**. *Es vital que sea la LTS (actualmente la v23 o v24) y no la "Current".*

Ejecutar el instalador.

**¡IMPORTANTE!:** En el asistente de instalación, asegúrense de que esté marcada la opción **"Add to PATH"** (generalmente viene marcada por defecto). Esto permite que la terminal encuentre a Node.

**La "Instalación" de Vite**

Aquí hay una aclaración conceptual clave: **No instalamos Vite en el computador para siempre** (como instalamos Office). Lo que hacemos es invocar a un "constructor" que descarga la última versión de Vite solo para crear la carpeta.

1. En la terminal, naveguen a la carpeta donde guardarán sus trabajos: cd Documentos (o la ruta que prefieran).
2. Ejecuten el comando mágico: npm create vite@latest

**¿Qué va a pasar aquí? (El momento de confusión común)** Si es la primera vez que usan esto en esa máquina, la terminal les preguntará:

*Need to install the following packages: create-vite@5.0.0 Ok to proceed? (y)*

Debes simplemente presionar la tecla **"y"** (yes) y luego **Enter**. *Explicación:* "Node está pidiendo permiso para descargar la herramienta temporalmente".

Set-ExecutionPolicy -Scope Process -ExecutionPolicy Bypass

Si npm create vite@latest falla, podemos intentar con: npx create-vite@latest

**Cheat Sheet – La Chuleta**

**¿Tengo Node?** -> node -v (Debe ser v22+)

**Crear Proyecto** -> npm create vite@latest

* *Nombre:* mi-app
* *Framework:* React
* *Variante:* JavaScript

**Entrar a la carpeta** -> cd mi-app

**Instalar las piezas** -> npm install

**Arrancar** -> npm run dev

**Práctica acompañada - Creación del entorno (Paso a Paso)**

**Verificación de Node.js:** Abre tu terminal (CMD, PowerShell o Terminal de VSCode) y escribe: node -v ***Resultado esperado****:* Deberías ver una versión.

**Creación del andamiaje (Scaffolding):** Navega a tu carpeta de proyectos y ejecuta: npm create vite@latest

El sistema te hará preguntas. Responde así (usa las flechas del teclado):

* *Project name:* clase-01-intro
* *Select a framework:* React
* *Select a variant:* JavaScript

**Instalación de dependencias:** Fíjate que Vite te da instrucciones al terminar. Entra a la carpeta: cd clase-01-intro

Ahora, mira la carpeta, no existe node\_modules. Ejecuta: npm install *Observación:* Verás una barra de progreso. NPM está leyendo el package.json y trayendo las librerías necesarias.

**Encendido de motores:** Ejecuta el entorno de desarrollo: npm run dev

*Resultado:* Verás una URL local (ej. http://localhost:5173).

**Acción:** Copia esa URL en tu navegador (Chrome/Edge/Firefox).

**Práctica 2: Limpieza del área de trabajo**

Vite nos da un ejemplo, pero nosotros queremos un lienzo en blanco para trabajar.

**Instrucciones:**

1. Abre la carpeta clase-01-intro en VS Code.
2. Busca la carpeta src.
3. Abre App.jsx. Este es tu componente principal.
4. Borra **todo** el contenido dentro del return ( ... ), dejando solo un div vacío o un fragmento.
5. Borra los import que ya no uses (como el logo de react o App.css).
6. Escribe un <h1> que diga "¡Hola React desde Cero!".
7. Guarda el archivo y mira el navegador sin recargarlo. ¿Cambió?
8. Abre main.jsx e intenta entender cómo se conecta tu App con el HTML (index.html).

**Ejercicio independiente - Actividad: Mi Tarjeta de Presentación Digital**

**Instrucciones:**

1. Utilizando el proyecto limpio de la práctica anterior.
2. Crea una estructura HTML dentro del return de App.jsx que represente una tarjeta de presentación.
3. Debe contener obligatoriamente:
   * Un h1 con tu nombre completo.
   * Un h2 o h3 con tu rol (ej. "Estudiante de Desarrollo Web").
   * Un párrafo p breve describiendo tu expectativa de este curso.
   * Una lista desordenada ul con 3 de tus hobbies.
   * (Opcional/Reto) Intenta buscar una imagen en internet y usar la etiqueta <img>. *Pista: En React, las etiquetas vacías deben cerrarse siempre, ej: <img src="..." />.*

**Criterios de Logro:**

* La aplicación corre sin errores en la consola (terminal sin rojo, consola del navegador limpia).
* El contenido es visible en el navegador.
* El código tiene una estructura de identación legible.
* Se eliminó el código basura del ejemplo inicial de Vite.

**¿Debería saber algo más?**

Aquí unos tips de profundidad:

1. **Extensiones de VS Code:** Instala **"ES7+ React/Redux/React-Native snippets"**. Te permitirá escribir rafce(**R**eact **A**rrow **F**unction **C**omponent **E**xport) y presionar TAB para crear la estructura de un componente automáticamente.
2. **Semantic Versioning (SemVer):** En el package.json, viste símbolos como ^ o ~ antes de los números de versión.
   * ^18.2.0: Significa "Instala la versión más reciente compatible con la 18" (puede actualizar la segunda cifra, pero no la primera). Esto es vital para evitar que una actualización automática rompa tu proyecto en el futuro.
3. **Git:** Aunque hoy no lo vimos a fondo, cada vez que creas un proyecto con Vite, este inicializa un repositorio git local. Es buena práctica hacer un "commit" inicial después de confirmar que la instalación funciona.

Logotipo

El contenido generado por IA puede ser incorrecto.