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.}

**Arquitectura web:**

Conceptos y características, tipos, mapas de navegación, procedimiento para diseñar, técnicas de integración de contenidos web, características y licenciamiento de la plataforma.

**Unidad I**

**Estructuración de Contenidos**

**Situación inicial - El Arquitecto vs. El Albañil**

Imagina que te contratan para construir un hospital. ¿Qué es lo primero que haces? Muchos dirían: "Comprar ladrillos y cemento". **Error.** Si haces eso, acabarás con baños sin puertas o quirófanos sin electricidad.

Lo primero que se hace es un **Plano Arquitectónico**. Se definen los flujos: ¿Por dónde entran las ambulancias? ¿Dónde esperan los pacientes? (Navegación). Se definen los materiales permitidos y certificados (Licenciamiento y Dependencias).

En el desarrollo de software pasa lo mismo:

* **Vite/React** son tus materiales y herramientas (ladrillos y grúas).
* **La Arquitectura de la Información** es el plano.
* **La estructura de carpetas** es cómo organizas esos materiales en la obra para no perder tiempo buscando un martillo.

Hoy aprenderemos a leer el plano (Arquitectura) y a conocer nuestra caja de herramientas (Estructura de archivos de Vite) para que, cuando construyamos, la aplicación no se derrumbe.

**Arquitectura Web y Mapas de Navegación**

Antes de tocar código, debemos entender la estructura lógica.

1. **Arquitectura de la Información (AI):**

Es la práctica de organizar y etiquetar el contenido para que el usuario encuentre lo que busca. En una SPA (Single Page Application), esto es vital porque no tenemos "páginas físicas" diferentes, sino "vistas" simuladas.

* **Jerárquica (Árbol):** La más común. Home -> Categoría -> Detalle. (Ej. Amazon).
* **Lineal:** Secuencial. Paso 1 -> Paso 2 -> Final. (Ej. Un Checkout de pago o un asistente de instalación).
* **En Red (Malla):** Todo conecta con todo. (Ej. Wikipedia).

1. **El Mapa de Navegación (Sitemap):**

Es la representación gráfica de la arquitectura.

* *Concepto Clave:* En React, el mapa de navegación dicta tu **Árbol de Componentes**. Si en tu mapa tienes un "Encabezado" que aparece en todas partes, en tu código tendrás un componente <Header /> en el nivel superior.

**Anatomía de un Proyecto Vite + React**

Vamos a "hacer la autopsia" al proyecto que creamos con npm create vite@latest. Cada archivo tiene una razón de ser ingenieril. Al abrir la carpeta del proyecto, vemos una estructura diseñada para la escalabilidad. Vamos a dividirla en tres zonas: **Configuración (Raíz)**, **Código Fuente (src)** y **Archivos Públicos**.

**La Zona Estática (Carpeta public)**

* **Contenido:** Archivos estáticos puros.
* *Diferencia con assets:* Los archivos aquí **NO** se procesan ni se minifican. Se copian tal cual al servidor final. Se usa para el favicon, robots.txt o imágenes que no cambian nunca y que se referencian directamente en el HTML (no vía import).

**La Zona de Configuración (La Raíz)**

Estos archivos controlan *cómo* se construye y gestiona el proyecto, no el código que ve el usuario final.

* **package.json:** (La CC del proyecto). Define el nombre, la versión y, lo más importante, las **dependencias** (React, React DOM) y los **scripts** (dev, build). Es el mapa de ruta que sigue Node.js.
* **package-lock.json:** (El recibo de compra exacto). Asegura que si instalas este proyecto en otro computador, se descarguen las versiones *exactas* de las librerías para evitar errores de compatibilidad.
* **vite.config.js:** (El Jefe de Obra). Es el archivo de configuración del empaquetador. Aquí se definen los "plugins" (como el plugin de React para que Vite entienda JSX) y se pueden configurar puertos, alias o proxies.
* **eslint.config.js:** (El Inspector de Calidad ).
  + *Explicación:* En versiones antiguas se usaba .eslintrc. Ahora, Vite usa el nuevo "Flat Config" de ESLint.
  + *Función:* Es un "policía" que vive en tu editor. Analiza tu código en tiempo real y te grita (subraya en rojo) si cometes errores de sintaxis, si dejas variables sin usar o si violas las reglas de buenas prácticas de React.
* **.gitignore:** (La lista de exclusión). Le dice a Git: "Ignora estos archivos, no los subas a la nube".
  + *Vital:* Aquí **debe** estar node\_modules. Si subes node\_modules a GitHub, es como intentar subir el edificio entero en lugar de solo los planos.
* **index.html:** (La Puerta de Entrada).
  + A diferencia de herramientas antiguas (CRA) donde estaba escondido en public, en Vite vive en la **raíz**.
  + Contiene el <div id="root"></div> y, crucialmente, la etiqueta <script type="module" src="/src/main.jsx"> que conecta todo.

**La Zona de Código (Carpeta src)**

Aquí es donde pasaremos el 99% del tiempo programando. Es el "Taller".

* **main.jsx:** (El Motor de Arranque).
  + Es el punto de entrada de JavaScript. Importa React y busca el elemento root del HTML para inyectar la aplicación.
  + *Nota:* Usa la extensión .jsx explícitamente, obligatorio en Vite para archivos que contienen sintaxis HTML+JS.
* **App.jsx:** (El Componente Principal).
  + Es el contenedor padre de toda la interfaz. Actualmente contiene el código de ejemplo de los contadores y logos que vimos al iniciar.
* **App.css y index.css:** (La Pintura).
  + index.css: Estilos globales (reseteos, fuentes base).
  + App.css: Estilos específicos del componente App.
* **assets/:** (El Almacén de Recursos).
  + Contiene react.svg.
  + *Diferencia Clave:* Lo que guardas aquí se puede **importar** dentro de los archivos JS (ej: import reactLogo from './assets/react.svg'). Vite procesa estos archivos, los optimiza y les agrega "hashes" para mejorar el caché del navegador.

**Integración de Contenidos y Licenciamiento**

1. **Import / Export (Módulos ES6):**

La técnica de integración en JS moderno.

* export: Hace que una función o variable sea pública para otros archivos.
* export default: La exportación principal del archivo (solo una por archivo).
* import: Trae esa funcionalidad a otro archivo.

**2. Licenciamiento:**

Al usar librerías de NPM (Node Package Manager), estamos usando código de otros.

* **MIT / ISC / Apache:** Permisivas. Puedes usarlas gratis, incluso para vender tu software. (React usa MIT).
* **GPL (Copyleft):** "Virales". Si usas una librería GPL, tu software *también* debe ser libre y abierto. Cuidado con esto en proyectos comerciales.

**El Semáforo de las Licencias**

Para saber si una librería es segura de usar, miren el tipo de licencia. Vamos a clasificarlas en un semáforo:

* **VERDE (Permisivas - "Haz lo que quieras"):**
  + **Nombres comunes:** MIT, ISC, Apache 2.0, BSD.
  + **¿Qué significa?:** Puedes usar la librería gratis, modificarla, usarla en trabajos de la universidad, en proyectos personales o **incluso en software comercial cerrado** para vender. Lo único que piden es que no borres el nombre del autor original en el código fuente.
  + *Ejemplo:* **React** usa licencia MIT. (Por eso Facebook permite que cualquiera haga apps con React y las venda).
* **AMARILLO (Copyleft Débil - "Comparte los cambios"):**
  + **Nombres comunes:** LGPL, MPL (Mozilla).
  + **¿Qué significa?:** Puedes usarla en tu software cerrado, PERO si modificas el código *interno* de la librería, debes compartir esos cambios con la comunidad.
* **ROJO (Copyleft Fuerte / Virales - "Todo o nada"):**
  + **Nombres comunes:** GPL (v2 o v3), AGPL.
  + **¿Qué significa?:** ¡Cuidado! Si usas una librería GPL en tu proyecto, **tu proyecto entero se vuelve GPL automáticamente**.
  + **La consecuencia:** Estás legalmente obligado a liberar tu código fuente para que cualquiera lo vea y lo copie. Si estás haciendo un software privado para un cliente o una empresa, **prohibido usar GPL** (a menos que el cliente quiera que sea Open Source).

**Preguntas Frecuentes**

**¿Puedo borrar el archivo index.css y App.css?**

Sí. Esos archivos son estilos de ejemplo. De hecho, en un proyecto profesional, solemos borrarlos para empezar con nuestra propia arquitectura de estilos (o instalar Tailwind/SASS). No son obligatorios para que React funcione, son solo "maquillaje".

**¿Por qué hay una carpeta public y otra assets dentro de src?**

Esta es una distinción de optimización de Vite:

* src/assets: Para imágenes/archivos que usas *dentro* del código (importados). Vite los procesa, los renombra para caché y los optimiza.
* public: Para archivos estáticos que no cambian y no se importan (como el favicon.ico o un robots.txt). Se copian tal cual al servidor final.

**Ejemplos implementados y demostrativos**

Del Mapa de Navegación al Código. Imagina que diseñamos un sitio simple para una veterinaria. *Mapa:*

1. Inicio (Home)
2. Servicios
3. Contacto

*Implementación en Estructura de Carpetas (Best Practice):* No tiramos todos los archivos en src. Creamos organización:

src/

├── components/ (Piezas reutilizables: Botones, Navbar)

│ ├── Navbar.jsx

│ └── Footer.jsx

├── pages/ (Vistas completas del mapa)

│ ├── Home.jsx

│ ├── Servicios.jsx

│ └── Contacto.jsx

├── App.jsx (El orquestador)

└── main.jsx

**Importación de Dependencias (Cómo se ve en código)**

Si queremos usar un icono, no dibujamos el SVG a mano, instalamos una dependencia (ej. react-icons).

*En terminal:* npm install react-icons *En código (Home.jsx):*

// 1. Integración: Importamos la dependencia

import { FaDog } from "react-icons/fa";

function Home() {

return (

<div>

<h1>Bienvenido a la Veterinaria</h1>

{/\* 2. Uso del componente importado \*/}

<FaDog style={{ color: 'brown', fontSize: '50px' }} />

</div>

);

}

export default Home;

**Práctica acompañada - La Limpieza y Estructuración**

**Instrucciones:**

1. Abre tu proyecto en VS Code.
2. **Detén el servidor** (Ctrl + C en la terminal) si está corriendo.
3. **Limpieza de archivos:**
   * Elimina src/App.css.
   * Elimina src/index.css (o borra todo su contenido para dejarlo en blanco).
   * Elimina la carpeta src/assets (crearemos la nuestra luego).
4. **Limpieza de código:**
   * Entra a main.jsx: Elimina la línea import './index.css'.
   * Entra a App.jsx: Elimina TODO el contenido y déjalo así:

function App() {

return (

<div>

<h1>Estructura Inicial Lista</h1>

</div>

)

}

export default App

1. **Creación de Arquitectura:**

* Dentro de src, crea una carpeta llamada components.
* Dentro de src, crea una carpeta llamada pages.

1. **Verificación:** Ejecuta npm run dev. Si ves el texto "Estructura Inicial Lista" en fondo blanco y letra negra (times new roman por defecto)

¡Felicidades! Has "reseteado" los estilos y tienes el control total.

**Práctica 2: Creación de Componentes e Integración**

Vamos a crear nuestro primer componente visual (un Encabezado) y lo vamos a integrar en la App principal.

**Instrucciones:**

1. Dentro de la carpeta src/components, crea un archivo llamado Header.jsx.
2. Escribe el siguiente código (o usa rafce si ya instalaste la extensión):

function Header() {

return (

<header style={{ backgroundColor: '#333', color: 'white', padding: '1rem' }}>

<h2>Mi Sitio Web Increíble</h2>

<nav>

<a href="#" style={{color: 'white', marginRight: '10px'}}>Inicio</a>

<a href="#" style={{color: 'white'}}>Contacto</a>

</nav>

</header>

);

}

export default Header;

1. Ahora, ve a src/App.jsx.
2. Importa tu nuevo componente al principio del archivo: import Header from './components/Header';
3. Úsalo dentro del return de la App:

function App() {

return (

<div>

<Header />

<main style={{ padding: '20px' }}>

<h1>Contenido Principal</h1>

<p>Aquí irá el resto de mi aplicación.</p>

</main>

</div>

)

}

**Ejercicio independiente**

**Escenario:** Un cliente (una panadería local) quiere una Single Page Application. Necesitan mostrar:

1. Quiénes somos.
2. Catálogo de Panes.
3. Formulario de Pedidos.

**Instrucciones:**

1. **En papel (o Paint):** Dibuja el **Mapa de Navegación** jerárquico.
2. **En VS Code:** Crea la estructura de carpetas necesaria en tu proyecto (components, pages).
3. Crea los archivos vacíos (placeholders) para cada página dentro de la carpeta pages: QuienesSomos.jsx, Catalogo.jsx, Pedidos.jsx.
4. Dentro de cada archivo, pon simplemente un título <h1> con el nombre de la página.
5. **El Entregable:** Modifica App.jsx para que importe los 3 componentes y los muestre uno debajo del otro (aún no sabemos hacer routing, así que los mostraremos todos juntos a modo de "scroll vertical").

**Criterios de Logro:**

* Estructura de carpetas lógica (src/pages).
* Nombres de archivos en **PascalCase** (PrimeraLetraMayuscula.jsx) - *Estándar de la industria*.
* El archivo App.jsx contiene los 3 imports correctos.
* No hay errores en consola.

**Ejercicio independiente**

**La importancia del export default vs export nombrado:** Export default se usa cuando el archivo *es* el componente (ej. Header.jsx). El export nombrado { funcionAuxiliar } se usa para librerías de utilidades donde un archivo tiene muchas funciones pequeñas.

**JSX no es HTML:** Recuérdales constantemente que aunque parece HTML, es JavaScript.

* class se convierte en className (porque class es una palabra reservada en JS).
* for se convierte en htmlFor.
* Las etiquetas siempre deben cerrarse (<img />, <br />). Si escriben <img> como en HTML antiguo, Vite lanzará un error y la app se romperá.

**El ecosistema de dependencias:** Menciónales que revisar el sitio **npmjs.com** es fundamental antes de instalar algo. Deben mirar:

* "Weekly Downloads" (¿Es popular?).
* "Last Publish" (¿Está mantenido o abandonado?).
* Esto es parte de la "seguridad y licenciamiento" de la plataforma.

Logotipo

El contenido generado por IA puede ser incorrecto.