Desarrollo web en entorno cliente

UD – 5

React

Contenidos

ÍNDICE

1. INTRODUCCIÓN

React es una biblioteca de JavaScript desarrollada por **Facebook** y lanzada en **2013**. Es ampliamente utilizada para construir interfaces de usuario dinámicas e interactivas. Su enfoque declarativo y basado en componentes ha hecho de React una herramienta líder en el desarrollo web moderno.

Esta biblioteca se centra en la creación de la capa de vista de una aplicación, proporcionando un sistema eficiente y modular para construir componentes reutilizables. Una de sus características **clave es el Virtual DOM**, que permite actualizaciones rápidas y eficientes al minimizar las manipulaciones directas del DOM. Además, React utiliza un sistema unidireccional de flujo de datos, lo que facilita la gestión del estado y la depuración en aplicaciones complejas.

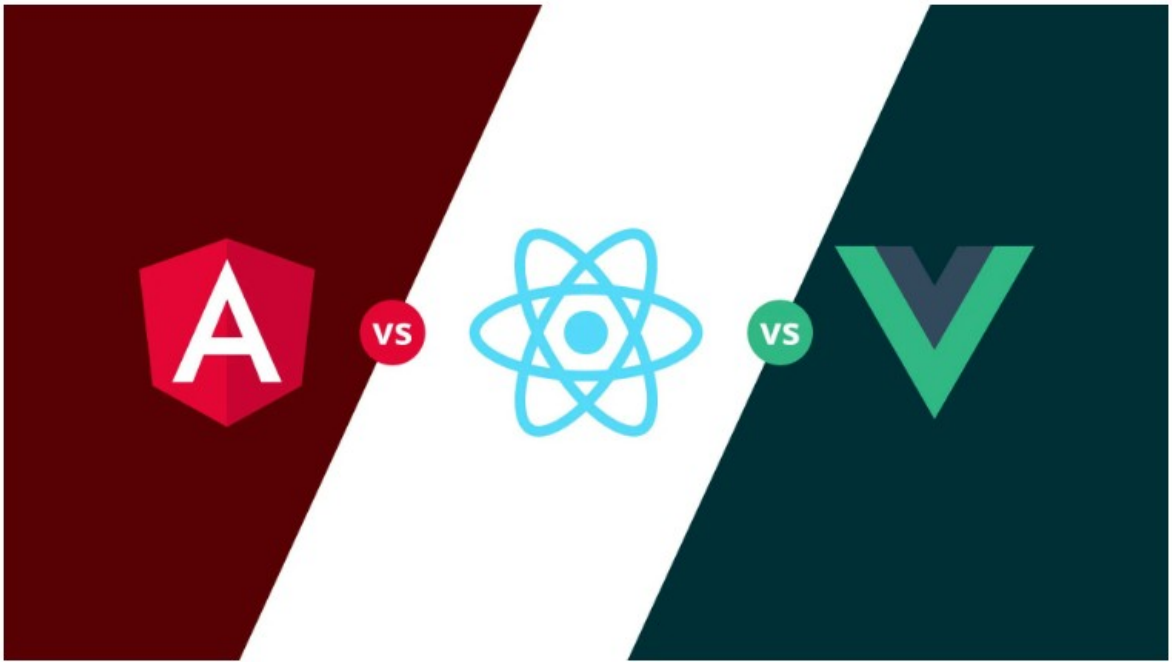
React es altamente flexible y puede ser integrado con otras bibliotecas o frameworks. Es ideal tanto para aplicaciones pequeñas como para proyectos grandes y escalables. Su ecosistema robusto incluye herramientas como **React Router** para manejo de rutas y bibliotecas de estado como Redux o Context API, que amplían su funcionalidad.

En resumen, React destaca por su rendimiento, su enfoque en la reutilización de componentes y su capacidad para manejar interfaces complejas de manera intuitiva. Esto la convierte en una solución popular entre desarrolladores de todos los niveles.

¿Por qué elegir React frente a otras opciones como Angular o Vue?

- **Simplicidad**: Aunque Angular ofrece una solución completa para el desarrollo de aplicaciones, su curva de aprendizaje puede ser empinada. React, al ser una biblioteca en lugar de un framework completo, **resulta más fácil de aprender** y utilizar.
- **Flexibilidad**: A diferencia de Vue, React **no impone una arquitectura rígida**, lo que permite a los desarrolladores elegir las herramientas y patrones que mejor se adapten a sus necesidades.
- **Rendimiento**: Al igual que Vue, React utiliza el concepto de **Virtual DOM**, ofreciendo un rendimiento eficiente en aplicaciones que requieren actualizaciones frecuentes y dinámicas.

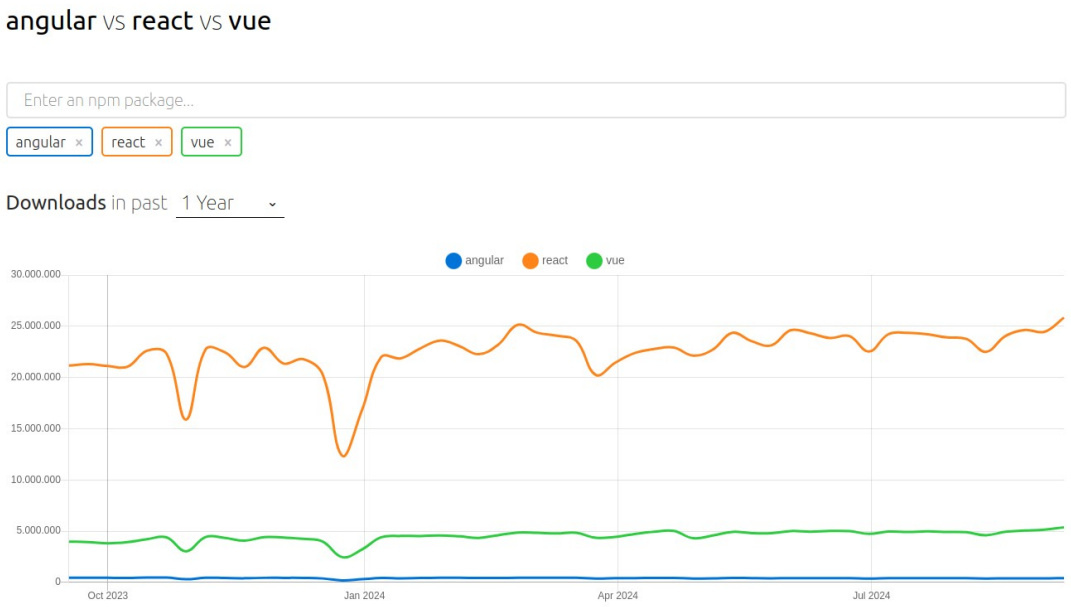
React es una elección sólida para el desarrollo web moderno, combinando facilidad de uso, escalabilidad y un ecosistema vibrante.



2. ¿USO EN LA ACTUALIDAD?

Para comprobar qué framework o librería es la más utilizada, podemos consultar “npm trends”:

https://npmtrends.com/angular-vs-react-vs-vue



3. CONFIGURACIÓN INICIAL

En esta sección, presentaremos cómo implementar una aplicación **React** de una sola página en su máquina local. El proyecto creado utilizará una configuración de compilación basada en Vite y nos permitirá usar componentes de archivo único (SFC) de Reat.

3.1 Dependencias

Para utilizar **React**, es necesario tener instalado nodejs y npm.

https://nodejs.org/en

3.2 NodeJs vs npm

- **Node.js** es un entorno de ejecución de JavaScript del lado del servidor que permite a los desarrolladores utilizar JavaScript para construir aplicaciones web y servidores. A diferencia de JavaScript en el navegador, que se ejecuta del lado del cliente, Node.js permite ejecutar código JavaScript en el servidor. Fue creado por **Ryan Dahl** y lanzado por primera vez en **2009**.

- **NVM** (*Node version manager*) o administrador de versiones de node es una forma de manejar diferentes versiones de node, puedes instalar o desinstalar versiones especificas de node si asi lo quieres.

- **NPM** (*Node Package Manager*) es el sistema de gestión de paquetes para Node.js. Es una herramienta que facilita la instalación, gestión y compartición de bibliotecas y herramientas de JavaScript desarrolladas por la comunidad. Al utilizar NPM, los desarrolladores pueden incluir fácilmente dependencias en sus proyectos, gestionar versiones y distribuir sus propias bibliotecas.

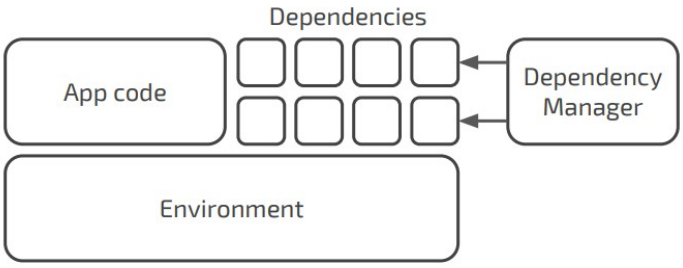
- **NPX** es una herramienta que se incluye con npm. Te permite ejecutar paquetes Node. js que están instalados local o globalmente en tu proyecto.

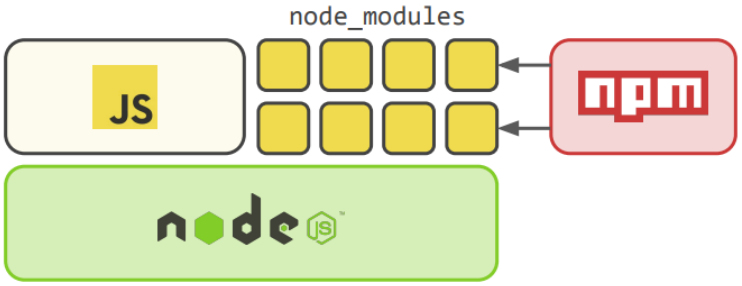
```
agustin@agustin-Inspiron
Webinar I$ node -v
v22.9.0
```

```
agustin@agustin-Inspiron-
Webinar I$ npm -v
10.8.3
```

Instalar Node mediante → **Opción 2 - Instalar Node.js 18 LTS usando NVM**
https://techviewleo.com/how-to-install-node-js-18-lts-on-ubuntu/

3.3 Estructura de un proyecto frontend con React





3.4 Crear primera aplicación

Para crear un proyecto vamos a utilizar vite:

npm create vite@latest

```
bas$ npm create vite@latest

> npx
> create-vite

✔ Project name: … hola-mundo
✔ Select a framework: › React
✔ Select a variant: › JavaScript + SWC

Scaffolding project in /home/agustin/Documentos/Docencia/2024-2025/DWEC/React/Pr
uebas/hola-mundo...

Done. Now run:

  cd hola-mundo
  npm install
  npm run dev
```

Seguimos las instrucciones hasta poder levantar el proyecto en el navegador. Si ves la siguiente página, es que todo ha funcionado correctamente

Para más información:

https://es.vitejs.dev/guide/


```
bas/hola-mundo$ npm run dev

> hola-mundo@0.0.0 dev
> vite

Port 5173 is in use, trying another one...

  VITE v5.4.7  ready in 298 ms

  ➜  Local:   http://localhost:5174/
  ➜  Network: use --host to expose
  ➜  press h + enter to show help
```

Otra alternativa para crear un proyecto sería con **npx** y utilizando el paquete **“create-react-app”**:

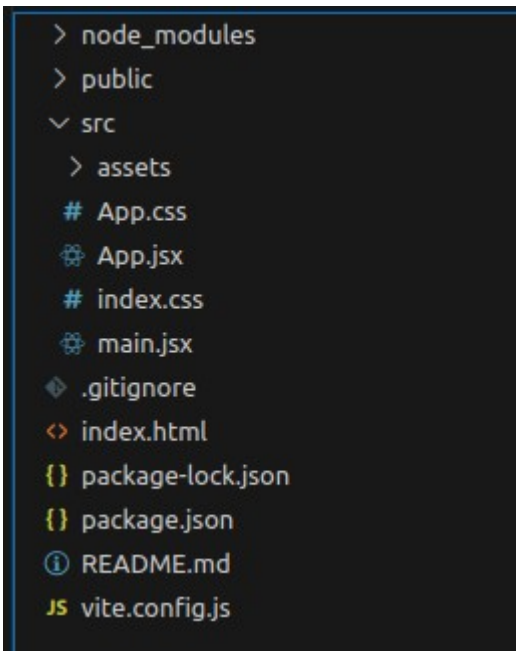
```
nidades/UD X  - React/Ejercicios$ npx create-react-app mi-app
Need to install the following packages:
create-react-app@5.0.1
Ok to proceed? (y)
```

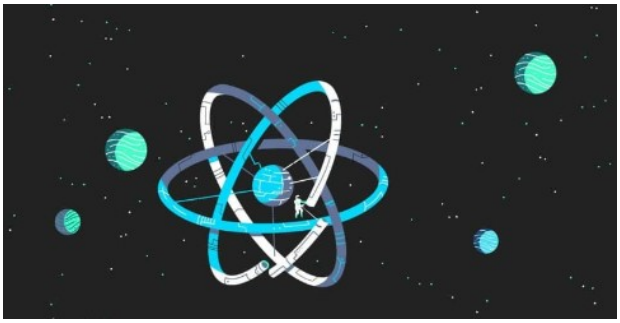
Es importante analizar los resultados del comando anterior:

```
Inside that directory, you can run several commands:

  npm start
    Starts the development server.

  npm run build
    Bundles the app into static files for production.

  npm test
    Starts the test runner.

  npm run eject
    Removes this tool and copies build dependencies, configuration files
    and scripts into the app directory. If you do this, you can't go back!

We suggest that you begin by typing:

  cd mi-app
  npm start
```

4. ESTRUCTURA INICIAL

Para dar tus primeros pasos en **React**, recorre y analiza la estructura en carpetas/archivos del proyecto anterior:





La **estructura del proyecto** es la siguiente:

- **src/**
  ***Carpeta principal** donde reside el código fuente de nuestra aplicación. Aquí es donde se encuentran todos los archivos relacionados con la lógica, los estilos, y los componentes principales de nuestra app. Es el núcleo del desarrollo.*
- **public/**
  Carpeta que contiene recursos estáticos y el archivo base de la aplicación. Este directorio generalmente incluye:
  - **assets/**
    Carpeta destinada a almacenar recursos estáticos como imágenes, fuentes, y otros archivos que no necesitan ser procesados por Webpack.
  - **index.html**
    Archivo base HTML de la aplicación. Este es el punto de entrada para el navegador, donde React inyecta los componentes de la app en el elemento con el identificador root.
- **src/components/**
  Carpeta que agrupa todos los componentes de la aplicación. Los componentes son bloques reutilizables de interfaz de usuario y pueden incluir sus propios estilos (CSS, SCSS, etc.), pruebas unitarias (tests), y otros recursos relacionados. La estructura interna puede variar según las necesidades del proyecto, pero comúnmente se organiza en subcarpetas por componente.
- **src/app.jsx**
  ***Archivo principal de la aplicación**. Aquí se define la estructura global y la lógica inicial de nuestra app, incluyendo la integración de rutas, contextos, o cualquier proveedor de estado global (por ejemplo, Redux o Context API).*
- **src/main.js**
  ***Archivo de inicialización**. Es el punto de partida de la aplicación React, donde se monta el árbol de componentes en el DOM. Aquí se encuentra la llamada a ReactDOM.createRoot() o ReactDOM.render() (según la versión de React).*

- **src/test/** *(opcional)*
  *Carpeta opcional dedicada a las pruebas de la aplicación. Puede incluir pruebas unitarias, de integración o de extremo a extremo (E2E), dependiendo del enfoque del proyecto. Las pruebas suelen escribirse con herramientas como **Jest**, **React Testing Library**, o **Cypress**.*

Esta estructura es una base común que puedes personalizar según los requisitos del proyecto. Por ejemplo, en proyectos más grandes, podrías incluir directorios adicionales como **hooks/** *(para custom hooks)*, **contexts/** *(para proveedores de contexto)*, o **services/** *(para manejar lógica de acceso a APIs).*

package.json

El archivo **package.json** es uno de los elementos más importantes en un proyecto de Node.js o React. Actúa como la "carta de presentación" del proyecto, proporcionando información clave y definiendo las dependencias, scripts y configuraciones necesarias.

Principales funciones

1. **Información básica del proyecto**:
   - **name**: Nombre del proyecto.
   - **version**: Versión del proyecto, generalmente siguiendo el estándar semver (por ejemplo, 1.0.0).
   - **description**: Breve descripción del proyecto.
   - **author**: Nombre del autor del proyecto.
   - **license**: Licencia bajo la cual se distribuye el proyecto (por ejemplo, MIT).

2. **Dependencias**:
   - **dependencies**: Lista de paquetes necesarios para que la aplicación funcione en producción. Por ejemplo:

```
"dependencies": {
  "react": "^18.2.0",
  "axios": "^1.2.3"
}
```

   - **devDependencies**: Lista de paquetes necesarios solo durante el desarrollo, como herramientas de testing o bundlers:

```
"devDependencies": {
  "jest": "^29.0.0",
  "eslint": "^8.10.0"
}
```

3. **Scripts**:
   - Contiene comandos personalizados que pueden ejecutarse con npm run. Por ejemplo:

```
"scripts": {
  "start": "react-scripts start",
  "build": "react-scripts build",
  "test": "jest"
}
```

Configuraciones adicionales:

- Puedes incluir configuraciones específicas para herramientas o frameworks, como Babel, ESLint o Webpack.

Propósito

El archivo **package.json** es fundamental para:

- Identificar y documentar las dependencias de tu proyecto.
- Establecer scripts y configuraciones para facilitar el desarrollo y despliegue.
- Permitir la instalación rápida de todas las dependencias con un solo comando (npm install o yarn install).

package-lock.json

El archivo **package-lock.json** es generado automáticamente cuando se ejecuta el comando npm install. Sirve como un archivo complementario a package.json, pero con un enfoque más técnico.

Principales funciones

1. **Congelación de versiones**:
   - Mientras que package.json puede indicar rangos de versiones (por ejemplo, "react": "^18.2.0"), package-lock.json especifica exactamente qué versión se instaló (por ejemplo, "18.2.0"). Esto asegura que cualquier desarrollador que instale las dependencias obtendrá las mismas versiones que las del entorno original.

2. **Resolución de dependencias**:
   - No solo guarda las versiones exactas de las dependencias directas, sino también de las dependencias transitivas (las dependencias de tus dependencias).

3. **Optimización de rendimiento**:
   - Facilita la instalación de dependencias al almacenar las ubicaciones exactas de los paquetes en el registro de npm, haciendo que futuras instalaciones sean más rápidas.

4. **Consistencia en el entorno de trabajo**:
   - Garantiza que el proyecto funcione de manera idéntica en diferentes máquinas o entornos.

Propósito

El archivo package-lock.json está diseñado para:

- Asegurar la estabilidad y consistencia del proyecto.
- Registrar todas las dependencias exactas para evitar errores inesperados en la producción.
- Optimizar la instalación y manejo de dependencias.

package.json vs package-lock.json

Aspecto	`package.json`	`package-lock.json`
Propósito	Define las dependencias y scripts del proyecto.	Congela las versiones exactas de las dependencias instaladas.
Mantenimiento	Editado manualmente por el desarrollador.	Generado y actualizado automáticamente por npm.
Alcance	Indica rangos de versiones de dependencias.	Almacena versiones exactas, incluyendo dependencias transitivas.
Obligatoriedad	Necesario para crear e instalar dependencias.	No es obligatorio, pero altamente recomendado para estabilidad.

node_modules

La carpeta **node_modules** es una parte esencial en proyectos de **Node.js** o frameworks como **React**. Contiene todos los paquetes y dependencias necesarias para que el proyecto funcione correctamente.

1. **Almacenamiento de dependencias**:
   - La carpeta **node_modules** guarda todas las dependencias definidas en el archivo **package.json**, así como las dependencias transitivas (dependencias de las dependencias).

2. **Organización**:
   - Cada paquete se instala como un subdirectorio dentro de **node_modules**.

```
node_modules/
  react/
  react-dom/
  axios/
```

   - Las dependencias transitivas también se almacenan en esta carpeta, organizadas según cómo las resuelve **npm**

3. **Tamaño**:
   - Puede ser una carpeta muy pesada porque almacena no solo los paquetes en sí, sino también todos los archivos relacionados, como documentos, herramientas de desarrollo y dependencias
   - Muchas veces es la carpeta más grande en el proyecto.

4. **Relación con package.json**:
   - Cuando ejecutas el comando **npm install**, las dependencias listadas en **package.json** se descargan e instalan en esta carpeta.
   - Si eliminas la carpeta **node_modules**, puedes regenerarla usando el comando anterior.

5. ¿QUÉ ES JSX?

JSX (*JavaScript XML*) es una extensión de JavaScript utilizada en **React** para describir cómo debería lucir la interfaz de usuario. Combina HTML con JavaScript, lo que hace que el código sea más legible y fácil de mantener. Aunque se parece a HTML, JSX se transpila a llamadas de JavaScript mediante herramientas como **Babel/vite** antes de ser interpretado por el navegador.

JSX permite insertar JavaScript directamente dentro de las etiquetas utilizando llaves {} y admite todas las capacidades del lenguaje JavaScript.

```jsx
const Li = ({ nombre, posicion }) => {
  return <li>{nombre} - {posicion}</li>
};

const X = () => {
    return <ul>
      <Li
        nombre={'Messi'}
        posicion={'Delantero'}>
      </Li>
      <Li
        nombre={'Ronaldinho'}
        posicion={'Centrocampista'}>
      </Li>
      <Li
        nombre={'Maradona'}
        posicion={'Media punta'}>
      </Li>
      <Li
        nombre={'Lamin Yamal'}
        posicion={'Extremo'}>
      </Li>
    </ul>
};

  root.render(
    <React.StrictMode>
      <X />
    </React.StrictMode>
  );
```

6. ANATOMÍA DE UN COMPONENTE COMO FUNCIÓN

En **React**, un **componente** es una unidad reutilizable que describe cómo una parte de la interfaz de usuario (UI) debe lucir y comportarse. Los componentes permiten dividir una aplicación en partes más pequeñas y manejables, facilitando la creación de interfaces complejas de manera modular.

```
const App = () => {

  const jugadores = [
    { nombre: "Messi", posicion: "Media punta" },
    { nombre: "Maradona", posicion: "Extremo" },
    { nombre: "Cristiano Ronaldo", posicion: "Delantero" }
  ];

  return (
   <div>
    <h1>Mejores Jugadores de la Historia</h1>
    <ul>

      {
        // Utilizamos un Map de JS, para recorrer los jugadores y dibujarlos

        jugadores.map((jugador, index) => (
          <Li key={index} nombre={jugador.nombre} posicion={jugador.posicion} />
        ))

      }
    </ul>
   </div>
  );
}

export default App;
```