CAMP / ies camp de morvedre

Contenidos interactivos en la web. REACT

PROF: ALICIA FERNÁNDEZ CATALÁN

CURSO 24-25

DAW - DIW

Contenidos

- Introducción
- React
- Frameworks y Herramientas Comunes en el Ecosistema de React
- Instalación y entorno de trabajo con REACT.js
- Configuración del proyecto
- Estructura del proyecto
- Código
- Sintaxis
- Estilos
- Componentes
- HOOKS
- NAVEGACIÓN

Introducción

- En el siguiente tema vamos a ver el framework de React para crear una página web sencilla.
- No veremos casos prácticos complicados ya que al estar en el módulo de diseño de interfaces web nos centraremos en el diseño.
- No obstante, con los contenidos vistos en esta unidad se sientan las bases para que podáis entender y desarrollar futuros proyecto usando esta tecnología

React

- React es una **biblioteca** de JavaScript para **construir interfaces de usuario**, creada por Facebook.
- Permite desarrollar aplicaciones web de manera modular y eficiente, usando **componentes reutilizables** que gestionan su propio estado y se actualizan de manera dinámica.
- React utiliza un **DOM virtual** para optimizar el rendimiento, actualizando solo las partes necesarias de la interfaz cuando cambian los datos, lo que mejora la velocidad y la experiencia del usuario.

Frameworks y Herramientas Comunes en el Ecosistema de React

- En el ecosistema de React, existen varios **frameworks** y **herramientas** que se utilizan para facilitar el desarrollo de aplicaciones.
- A continuación, te **menciono algunos framewoks de los más comunes y populares**:
- **Next.js:** Propósito: Next.js es un framework basado en React que permite la renderización del lado del servidor (SSR), la generación de sitios estáticos (SSG) y la creación de aplicaciones híbridas.
- Características clave:
  - Soporte para SSR (renderización del lado del servidor) y SSG (generación de sitios estáticos).
  - Enrutamiento basado en archivos.
  - Optimización automática de recursos.
  - Soporte para API Routes (funciones del servidor).

Frameworks y Herramientas Comunes en el Ecosistema de React

- **Gatsby**: Propósito: Gatsby es un generador de sitios estáticos que usa React y GraphQL para crear sitios rápidos y optimizados.
- Características clave:
  - Generación de sitios estáticos con enfoque en rendimiento.
  - Integración con APIs de datos, usando GraphQL para obtener datos desde múltiples fuentes.
  - Optimización automática de imágenes y recursos estáticos.
- **Remix:** Propósito: Remix es un framework para construir aplicaciones web modernas con React que se centra en la experiencia del usuario, la rapidez de carga y la optimización del rendimiento.
- Características clave:
  - Manejo avanzado del enrutamiento y los datos.
  - Soporte para SSR y la carga de datos en el servidor antes de la renderización.
  - Manejo eficiente de formularios, errores y cachés.

Frameworks y Herramientas Comunes en el Ecosistema de React

- **React Native**: Propósito: React Native permite el desarrollo de aplicaciones móviles nativas para iOS y Android utilizando JavaScript y React.
- Características clave:
  - Desarrollo móvil con componentes React.
  - Acceso a APIs nativas para funcionalidades de dispositivos móviles.

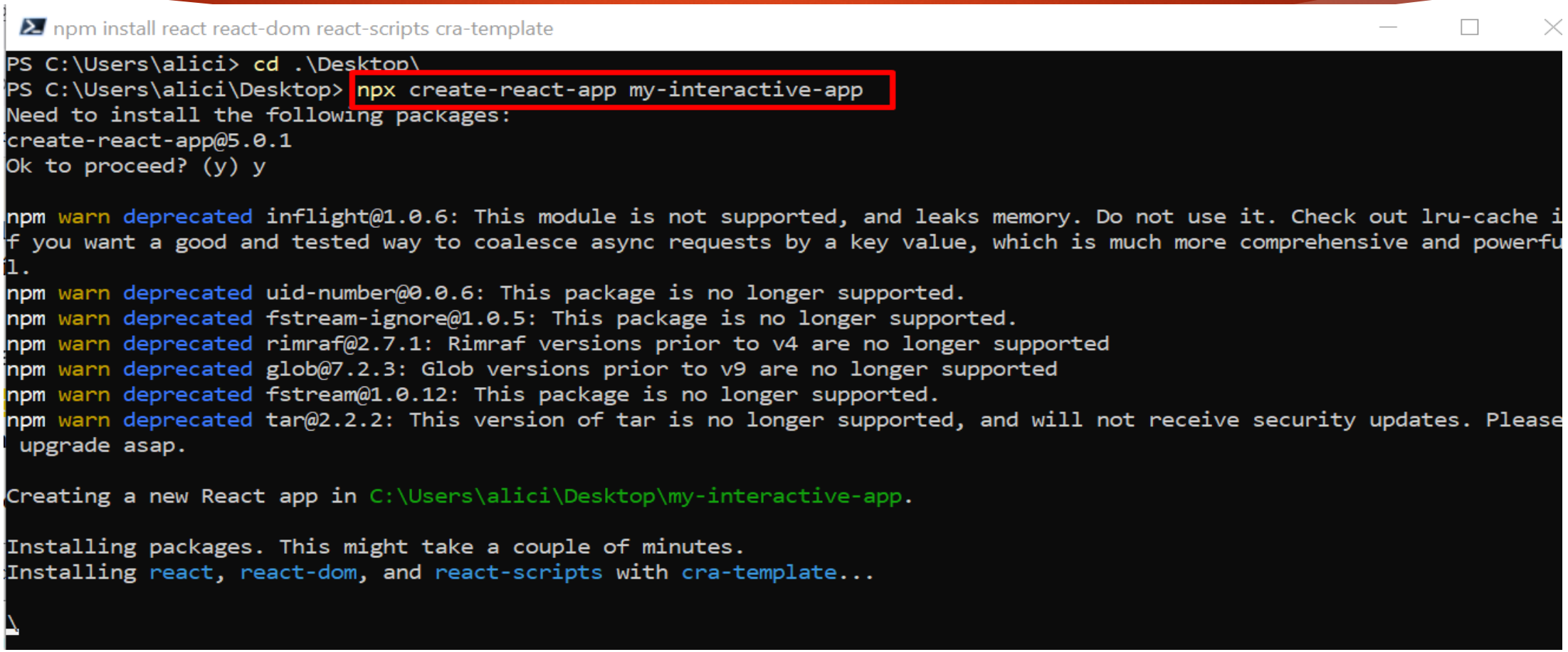
Frameworks y Herramientas Comunes en el Ecosistema de React

- **Herramientas Comunes en el Ecosistema de React**
- Existen numerosas herramientas, nosotros sólo veremos la siguiente:
- **React Router:** React Router es una librería para manejar el enrutamiento en aplicaciones React, permitiendo la navegación entre diferentes vistas sin recargar la página.
- Características clave:
  - Enrutamiento basado en componentes.
  - Soporte para rutas anidadas.
  - Funcionalidades avanzadas como protección de rutas, redirección y navegación dinámica.

Instalación y entorno de trabajo con REACT

- Para instalar React.js es necesario tener una versión reciente de Node.js y npm (el gestor de paquetes para Node) instalados.
  - Node.js ya lo instalamos en la unidad anterior
- **1. Instalar React y Crear un Proyecto**
- Lo primero que necesitas es configurar un entorno de React. Para hacerlo, puedes usar **Create React App**, que es una herramienta oficial que configura el entorno por ti.

Instalación y entorno de trabajo con REACT

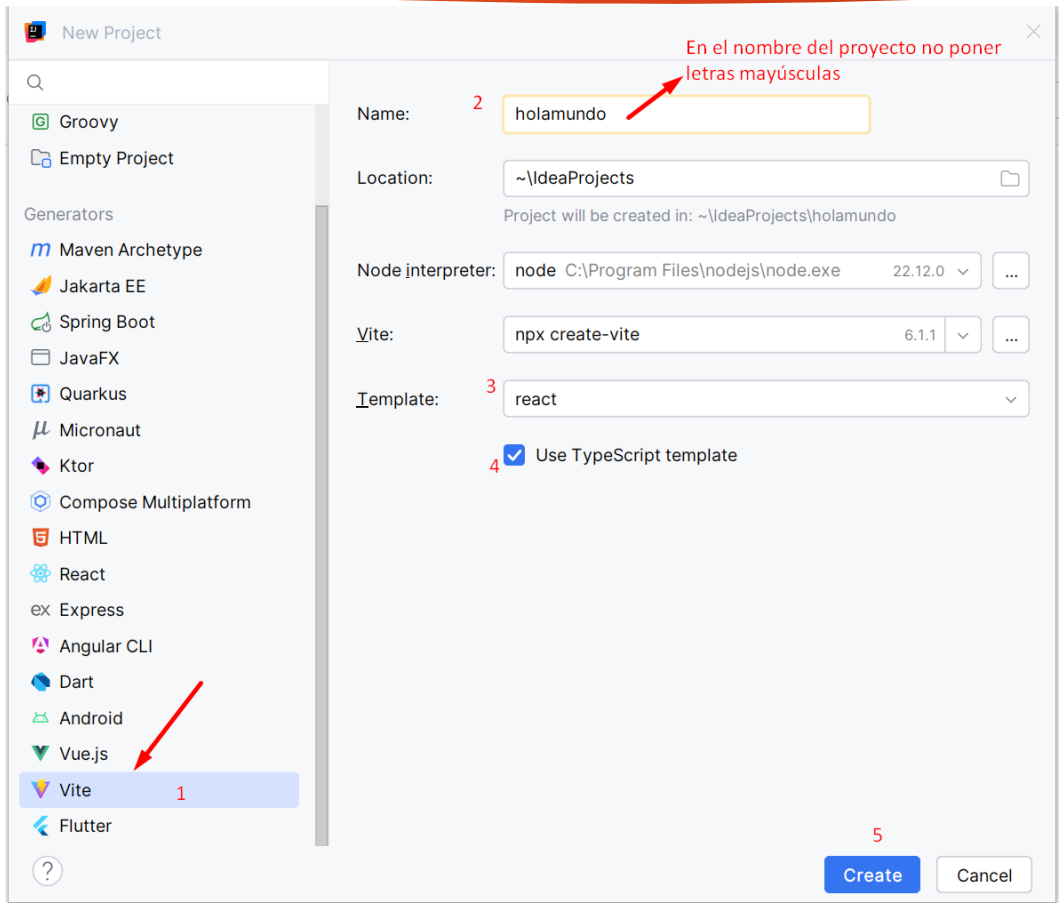


Configuración del proyecto

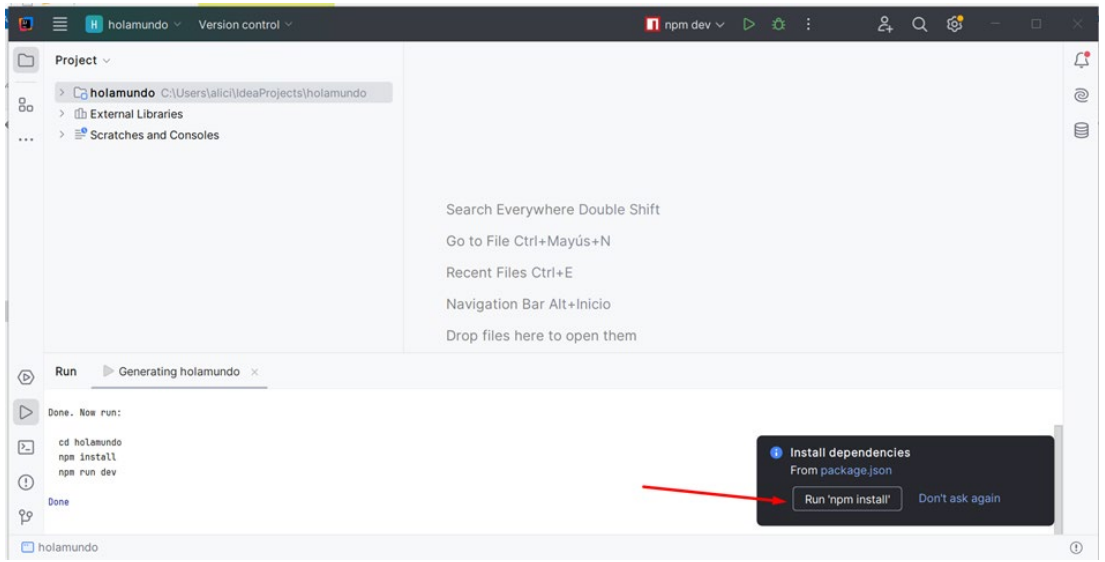
1. Vamos a nuestro IntelliJ, creamos un nuevo proyecto y seleccionamos Vite.
   - Veamos este vídeo primero **Nextjs vs Vite con React ¿cuál debería usar?**

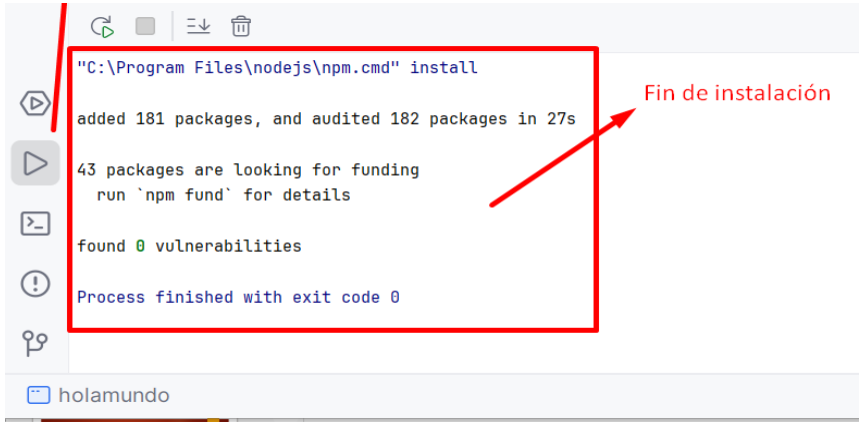
   https://www.youtube.com/watch?v=PLesX9ZJLNk
2. En template elegiremos React
3. Marcamos la opción `Use TypeScript template`
4. Click en `Create`
5. Nos debe de salir la opción en forma de banner para ejecutar el comando `npm install`. Aceptamos.
   - De no ser así, iremos a la terminal (barra inferior izquierda) y ejecutaremos el comando. Esto nos instalará los paquetes necesarios en el entorno de trabajo.
6. Le damos a correr (botón superior derecha, botón de play).

Configuración del proyecto

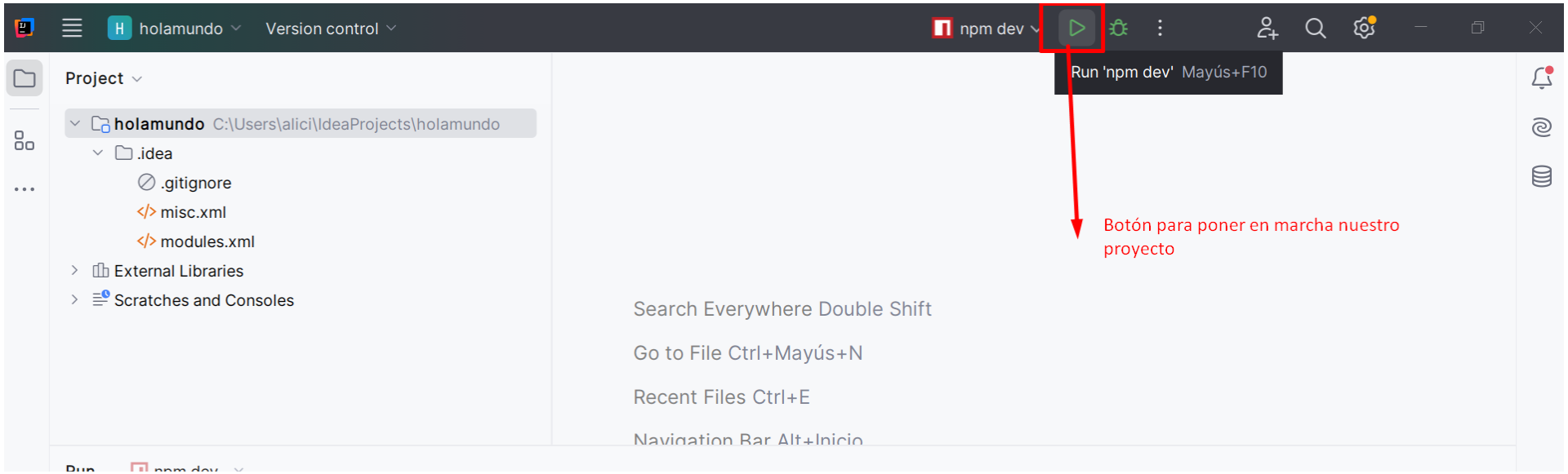


Configuración del proyecto

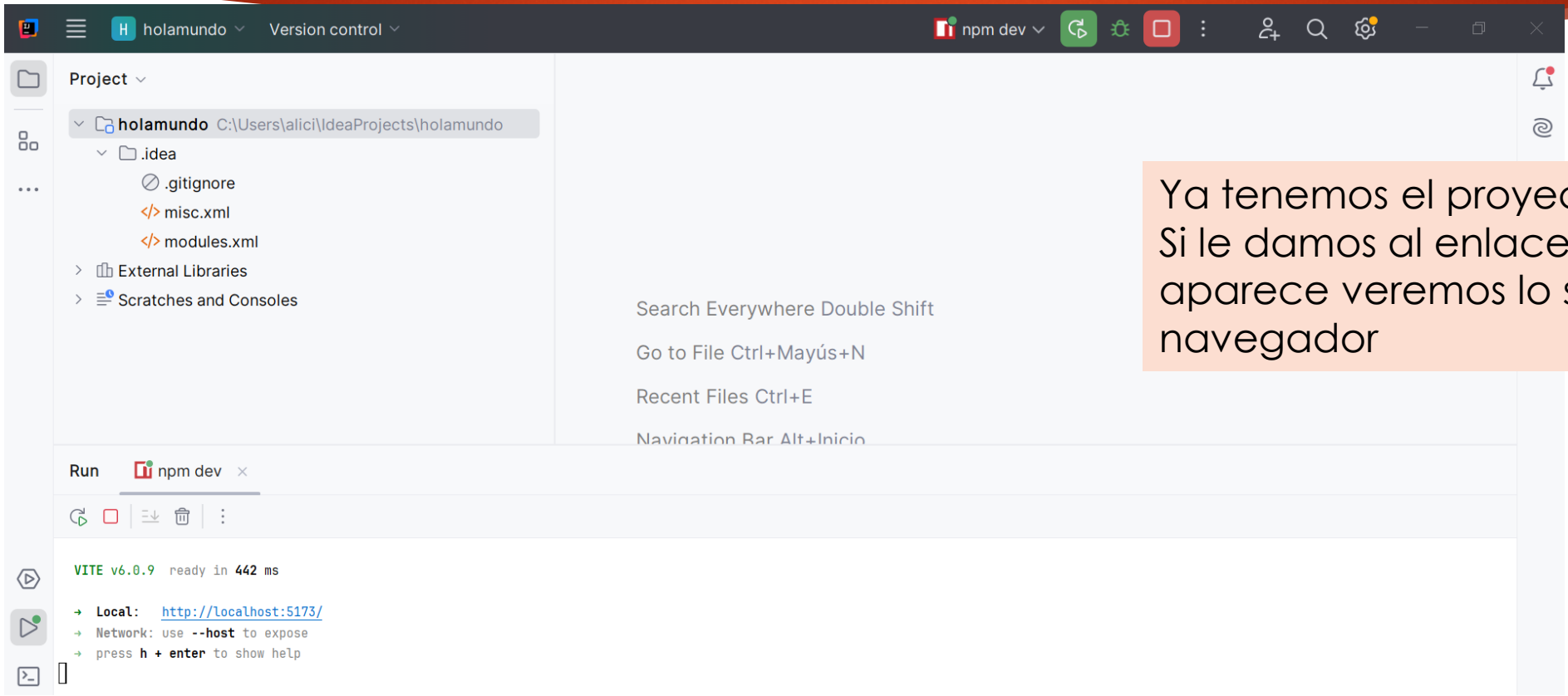




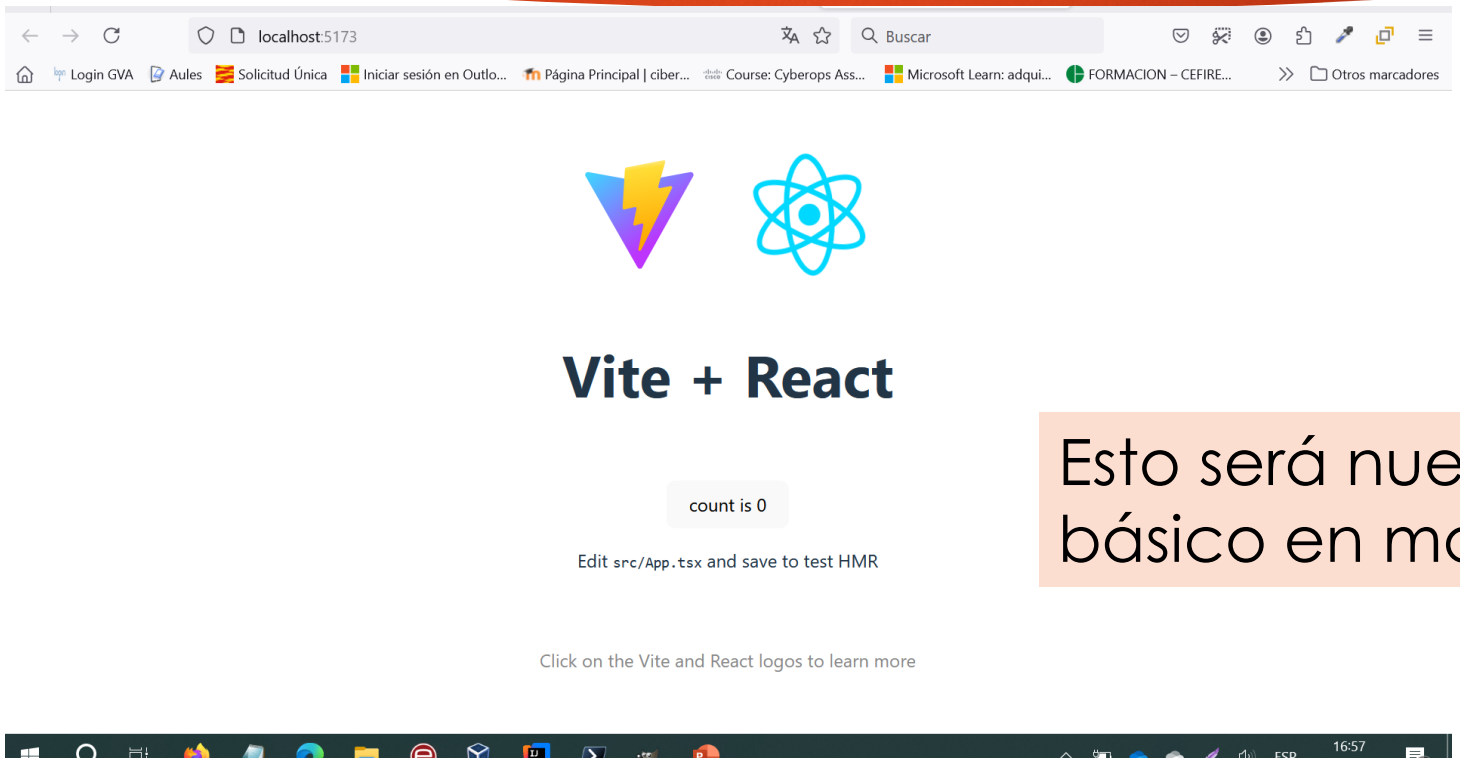
Configuración del proyecto



Configuración del proyecto



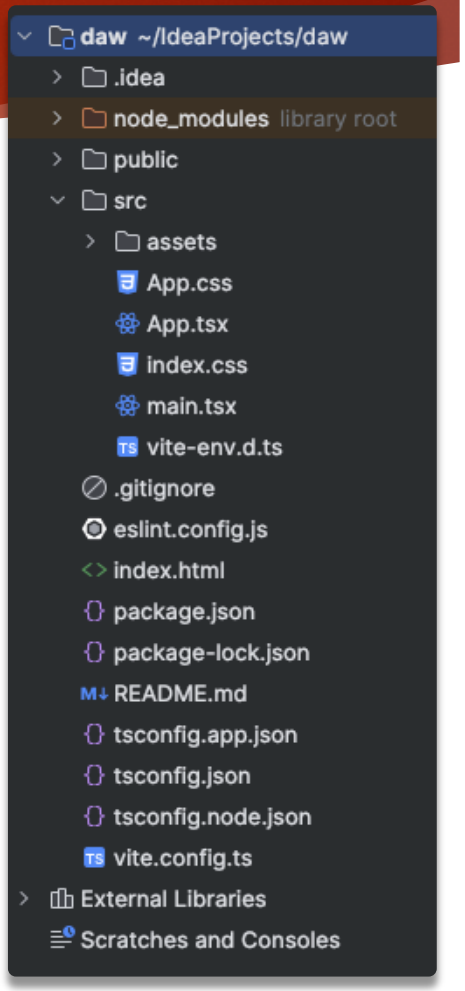
Ya tenemos el proyecto en marcha. Si le damos al enlace web que nos aparece veremos lo siguiente en el navegador



Esto será nuestro proyecto básico en marcha

Estructura del proyecto

- Nosotros trabajaremos dentro de la carpeta **src/**.
- La carperta **public/** contiene aquellos **elementos públicos** (típicamente logos e images).
- La carpeta **node_modules/** contiene las dependencias del proyecto.
- Dentro de src/ tenemos **assets/**, generalmente se utiliza para almacenar r**ecursos estáticos** que serán utilizados a lo largo de la aplicación (fuentes, audios, imágenes, jsons..).
- El fichero **main.tsx no se va modificar** será el fichero App.tsx, el que de ser necesario, se modificará.



Código en React

- Los archivos escritos en React acaban con la extensión `.tsx`. TSX (TypeScript XML) es una extensión de archivo que combina TypeScript y JSX.
- **JSX (JavaScript XML) es una extensión, nuevamente, de la sintaxis de JavaScript que permite escribir HTML dentro del código JavaScript.**

JSX

```
function App() {
  return <h1>¡Hola, mundo!</h1>;
}
```

¿Cómo funciona en la práctica?

Cuando usas TSX en un proyecto de React con TypeScript:

1. **Escribes tus componentes como si fueran HTML dentro de JavaScript**, pero con soporte para tipado estático (TypeScript).
2. **Al compilar, el código TSX se transforma en JavaScript puro que entiende el navegador**. Esto es manejado por herramientas como Babel o TypeScript Compiler.

```tsx
const App: React.FC = () => <h1>¡Hola, mundo!</h1>;
```

```tsx
const App = () => React.createElement("h1", null, "¡Hola, mundo!");
```

Sintaxis en React

- **En React existen distintas maneras de crear componentes:**
- *function( )* VS *const = ( ) =>*
  - La elección entre *function* y *const = () =>* **depende del contexto** en el que se utilicen, ya que ambos tienen **casos de uso específicos**.
- Según el **estándar** ES6 y las prácticas modernas, se **recomienda** usar **funciones flecha** (*const = () =>*) para la mayoría de los casos, pero hay excepciones.
  - Usa function solo cuando necesitas hoisting, un contexto propio de this, o para definir métodos tradicionales en clases.

Importación de estilos

- **CSS GLOBAL**

En React, puedes importar estilos de diferentes maneras dependiendo de cómo estructures tu proyecto.

```
import './styles.css'; /* Importa un archivo CSS con alcance global */
```

- **Por lo general, se suele crear un fichero .css global de la aplicación.**
- Este fichero determina *themes*, fuentes, etiquetas concretas, etc. Que van a ser iguales a lo largo de toda la aplicación (como hemos podido ver en temas anteriores).
- Esta manera de importar es la que ya conocemos donde especificamos los estilos de las clases y las etiquetas.

Importación de estilos

- CSS Modular

Permite que los estilos sean locales al componente, evitando conflictos de nombres o clases.

Tiene la extensión de fichero:
*.module.css

Es necesario la importación de esta manera o se importará como global

```css
/* FICHERO: styles.module.css */
.button {
  background-color: blue;
  color: white;
}
```

```tsx
import styles from './styles.module.css';

const Button: React.FC = () => {
  return <button className={styles.button}>Click me</button>;
};
```

Importación de estilos

- SASS/SCSS
- React soporta SASS/SCSS para estilos más avanzados.
- Tiene la extensión de fichero: *.scss
- Es necesario la **instalación** para su uso, por terminal:

npm install --save-dev sass

```scss
/* FICHERO: tyles.scss */
.button {
  background-color: blue;
  &:hover {
    background-color: darkblue;
  }
}
```

```jsx
import './styles.scss';

const Button: React.FC = () => {
  return <button className="button">Click me</button>;
};
```

Importación de componentes

- Exportación nombrada

Puedes exportar varios elementos desde un archivo y elegir cuál importar.

TypeScript-JSX
```
export const Button = () => <button>Click me</button>;
export const Input = () => <input />;
```

TypeScript-JSX
```
import { Button, Input } from './Button';

function App() {
  return (
    <>
      <Button />
      <Input />
    </>
  );
}
```

Importación de componentes

- Exportación por defecto

Solo puedes exportar un elemento como predeterminado por archivo.

TypeScript-JSX

```
const Button = () => {return(<button>Click me</button>)};
export default Button; /*Atendiende a como se exporta*/
```

TypeScript-JSX

```
import Button from './Button';

function App() {
  return <Button />;
}
```

No es una buena práctica hacer este tipo de exports múltiples por ficheros.
Por lo general, buscamos que el contenido de un fichero cumpla con una determinada función (principios SOLID, la letra S).
Eso no quiere decir que no se pueda, o incluso se deba, hacer en algunos casos.
Pero no es común la exportación de más de un componente por fichero.

Tip para la exportación

Componentes y estilos

- Es una buena práctica mantener componentes y estilos organizados.
- Los componentes se suelen almacenar en la carpeta **components/** y para cada componente, es una buena práctica, el crea otra para el componente, en la cual se almacena toda lo relacionado con él.

```
src/
  components/
    button/
      Button.tsx
      Button.module.css
```

Hooks

- Los hooks son unas funciones que permiten distintas funcionalidades en nuestros componentes de React. Es posible definir hooks propios combinando los hooks proporcionados.
- Solo nos vamos a centrar en conocer dos:
  - useState: permite definir estados.
  - useEffect: permite definir efectos secundarios (ojo con su uso).

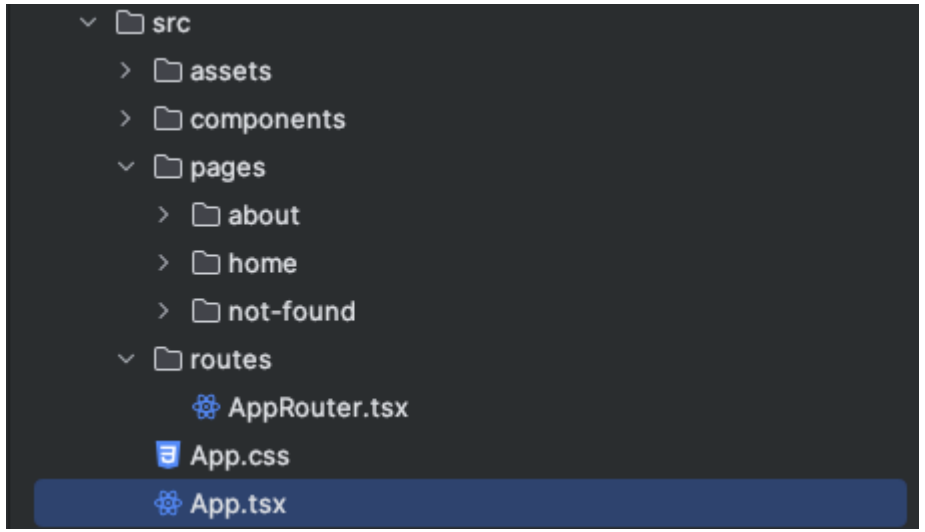
```
const [estado, setEstado] = useState(/*valorInicial*/);
```

```
useEffect(() => {
  //Código del efecto
}, [dependencias]); //Dependencias que disparan el efecto
```

Navegación y vistas

- React no cuenta con las funciones para proporcionar el enrutamiento en la página, por lo que hacemos uso de bibliotecas externas (React Router).
  - npm install react-router-dom
- **Se suelen crear una carpeta pages/ para la definición de vistas, al igual que hacíamos con los componentes, y una routes/ en la cual vamos a definir nuestro fichero para el árbol de navegación de nuestra web.**

Estructura de carpetas y fichero con la rutas




```tsx
import React from 'react';
import { BrowserRouter as Router, Routes, Route } from 'react-router-dom';
import Home from "../pages/home/Home";
import About from "../pages/about/About";
import NotFound from "../pages/not-found/NotFound";

const AppRoutes: React.FC = () : Element => { Show usages
    return (
        <Router>
            <Routes>
                <Route path="/" element={<Home />} />
                <Route path="/about" element={<About />} />
                <Route path="*" element={<NotFound />} />
            </Routes>
        </Router>
    );
};

export default AppRoutes; Show usages
```

Navegación y vistas

- Es importante definir correctamente las rutas que tendrán nuestras páginas.
- Será necesario el uso de una barra de navegación para poder navegar entre las vistas.
- También se puede añadir navegación por botones a las vistas mediante el hook useNavigate.
  - Recuerda de incluir tu componente con las rutas en tu App, solo dentro de las vistas enrutadas podrás navegar.

Navegación y vistas

```tsx
import React from 'react';
import { Link } from 'react-router-dom';

const Navbar: React.FC = () : Element => { Show usages
    return (
        <nav>
            <ul>
                <li>
                    <Link to="/">Home</Link>
                </li>
                <li>
                    <Link to="/about">About</Link>
                </li>
            </ul>
        </nav>
    );
};

export default Navbar; no usages
```

```tsx
import React from 'react';
import { BrowserRouter as Router, Routes, Route } from 'react-router-dom';
import Home from "../pages/home/Home";
import About from "../pages/about/About";
import NotFound from "../pages/not-found/NotFound";
import Navbar from "../components/nav-bar";

const AppRoutes: React.FC = () : Element => { Show usages
    return (
        <Router>

            <Navbar/>

            <Routes>
                <Route path="/" element={<Home />} />
                <Route path="/about" element={<About />} />
                <Route path="*" element={<NotFound />} />
            </Routes>
        </Router>
    );
};

export default AppRoutes; Show usages
```

Navegación y vistas

```tsx
import React from 'react';
import { useNavigate } from 'react-router-dom';

const Button: React.FC = () : Element => { Show usages
    const navigate : NavigateFunction = useNavigate();

    const handleNavigate : () => void = () : void => { Show usages

        navigate('/about');
    };

    return (
        <div>
            <h1>Home Page</h1>
            <button onClick={handleNavigate}>Go to About</button>
        </div>
    );
};

export default Button; Show usages
```

```tsx
import AppRoutes from "./routes/AppRouter.tsx";

function App() : Element { Show usages
  return (
      <>
          <AppRoutes />
      </>
  )
}

export default App Show usages
```

Biblioteca VS framework

Danger

Biblioteca vs framework

Note

Framework

Un framework es un **conjunto de herramientas y convenciones** que proporciona una estructura para desarrollar aplicaciones de manera más rápida y con menos esfuerzo. A diferencia de una biblioteca, un framework **define el flujo de control** y el desarrollador debe adaptarse a sus convenciones.

Note

Biblioteca:

Una biblioteca es una **colección de funciones y clases predefinidas** que puedes utilizar para realizar tareas específicas. Los desarrolladores eligen cuándo y cómo llamar a estas funciones.

Entrena y desarrolla tus habilidades

- Programación
  - CodeWars
  - LeetCode
  - CodinGame
- CSS
  - CodingFantasy
  - FlexBox Froggy
  - CSS Dine
- FullStack canal de divulgación
  - MiduDev
  - Curso React
- Otros
  - CodeAcademy
  - edX
  - kaggle
  - HugginFace
  - Ollama
  - pylint
  - replit
  - thomy