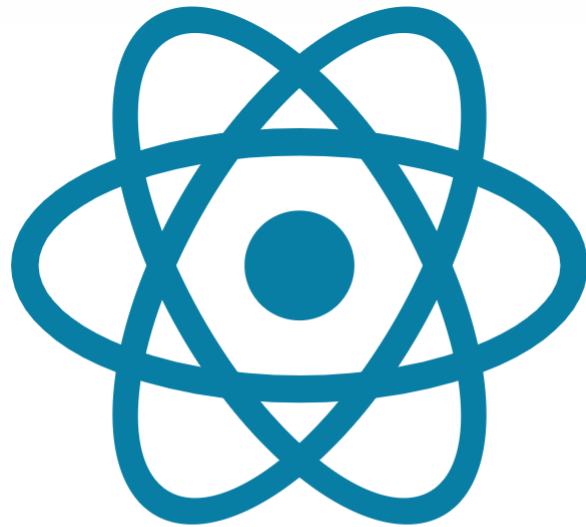


DESARROLLO EN ENTORNO CLIENTE/SERVIDOR

UD10: FRAMEWORK CLIENTE

REACT - Introducción



React

UD10 - Framework cliente - REACT Introducción

1. Evaluación.....	3
2. Introducción a REACT.....	3
2.1. Configuración del entorno de REACT.....	5
3. Instalando REACT en el servidor.....	5
3.1. Creación del contenedor en Docker.....	5

1. Evaluación

El presente documento, junto con sus actividades, cubren los siguientes criterios de evaluación:

RESULTADOS DE APRENDIZAJE	CRITERIOS DE EVALUACIÓN
RA0612.6. Desarrolla aplicaciones web analizando y aplicando las características del modelo de objetos del documento.	c) Se ha creado y verificado un código que acceda a la estructura del documento. d) Se han creado nuevos elementos de la estructura y modificado elementos ya existentes. e) Se han asociado acciones a los eventos del modelo. h) Se han independizado las tres facetas (contenido, aspecto y comportamiento), en aplicaciones Web.
RA0612.7. Desarrolla aplicaciones Web dinámicas, reconociendo y aplicando mecanismos de comunicación asíncrona entre cliente y servidor.	e) Se ha utilizado comunicación asíncrona en la actualización dinámica del documento Web. f) Se han utilizado distintos formatos en el envío y recepción de información. h) Se han clasificado y analizado librerías que faciliten la incorporación de las tecnologías de actualización dinámica a la programación de páginas Web. i) Se han creado y depurado programas que utilicen estas librerías.

2. Introducción a REACT

React (también llamada React.js o ReactJS) es una biblioteca Javascript de código abierto diseñada para crear interfaces de usuario con el objetivo de facilitar el desarrollo de aplicaciones en una sola página. Es mantenido por Facebook y la comunidad de software libre. En el proyecto hay más de mil desarrolladores libres.

React intenta ayudar a los desarrolladores a construir aplicaciones que usan datos que cambian todo el tiempo. Su objetivo es ser sencillo, declarativo y fácil de combinar. React sólo maneja la interfaz de usuario en una aplicación; React es la Vista en un contexto en el que se use el patrón MVC (Modelo-Vista-Controlador) o MVVM (Modelo-vista-modelo de vista).

React tiene varias características claves:

- **React es declarativo.** Cuando empezamos a trabajar con React, veremos un cambio en la forma en la que nos acercamos al código. En otros lenguajes de programación, nuestro código se basa en instrucciones. Es decir, línea a línea le decimos al programa cómo crear, colocar, nombrar y diseñar nuestros elementos. Este tipo de código es imperativo.

React es declarativo. Esto quiere decir que dejaremos de preocuparnos tanto por cómo se hacen las cosas y nos preocupamos más por qué es lo que queremos mostrar. Para ello,

diseñamos la vista que queremos para cada estado, con los elementos y datos que queremos que tenga. A partir de esto, React actualiza de modo eficiente cuando cambian los datos que importamos.

- **React se basa en componentes.** Entre los principios básicos de React, el segundo principio es que React se basa en componentes. Una forma de pensar en este estilo de programación es la construcción con legos. React nos permite ejecutar un tipo de desarrollo en el que nuestra aplicación es comprendida como pequeños componentes que se ensamblan para construir componentes un poco más grandes, que a su vez construyen componentes más grandes. Este proceso continúa hasta crear el componente más grande de todos, que es nuestra propia aplicación.
- **DOM Virtual.** El DOM virtual (VDOM) es un concepto de programación donde una representación ideal o “virtual” de la IU se mantiene en memoria y en sincronía con el DOM “real”, mediante una biblioteca como ReactDOM. Este proceso se conoce como reconciliación.

Este enfoque hace posible la API declarativa de React: le dices a React en qué estado quieres que esté la IU, y se hará cargo de llevar el DOM a ese estado. Esto abstrae la manipulación de atributos, manejo de eventos y actualización manual del DOM que de otra manera tendrías que usar para construir tu aplicación.

2.1. Configuración del entorno de REACT

Para trabajar con REACT podemos hacerlo de dos formas:

1.- Utilizando REACT desde una CDN como un script más:

```
<script src="https://unpkg.com/react@18/umd/react.development.js"></script>  
<script src="https://unpkg.com/react-dom@18/umd/react-dom.development.js"></script>
```

Esto permitiría usar REACT, por ejemplo, en proyectos existentes, pero no es lo adecuado y no se recomienda.

2.- Utilizando REACT como un framework:

Es la forma más usual, porque así se usa marcado JSX. JSX es totalmente opcional, pero la mayoría de los proyectos de React usan JSX por la comodidad que ofrece.

3. Instalando REACT en el servidor

Vamos a realizar los siguientes pasos para instalar REACT y podamos hacer peticiones al servidor.

3.1. Creación del contenedor en Docker

En primer lugar vamos a crear el contenedor que ejecutará el servidor donde se construirá la aplicación de REACT.

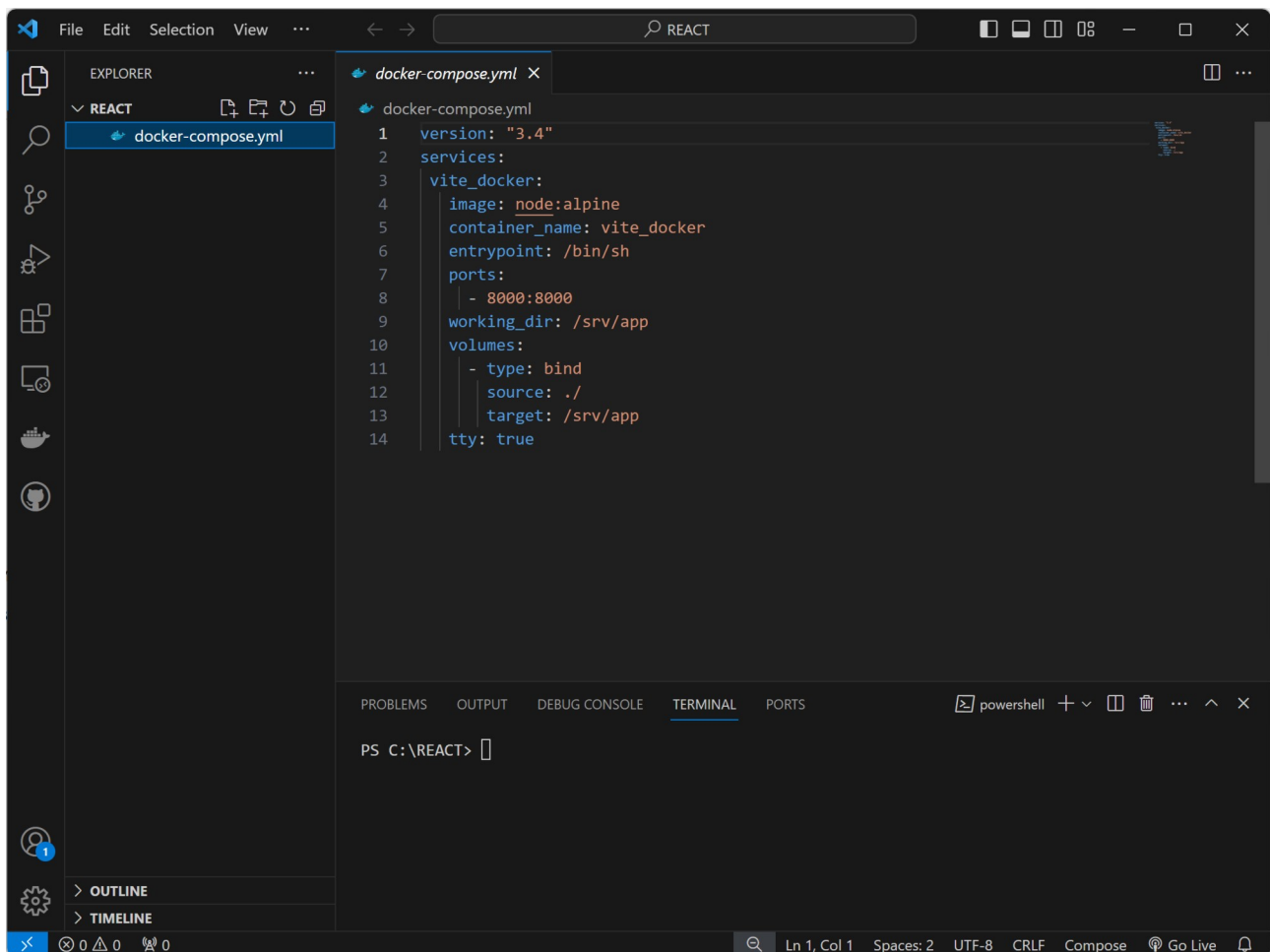
En una carpeta vacía creamos el fichero *docker-compose.yml* que contendrá lo siguiente:

```
version: "3.4"  
services:  
  vite_docker:  
    image: node:alpine  
    container_name: vite_docker  
    entrypoint: /bin/sh  
    ports:  
      - 8000:8000  
    working_dir: /srv/app  
    volumes:  
      - type: bind  
        source: ./  
        target: /srv/app  
    tty: true
```

Con este documento le decimos a Docker que construya una imagen a partir de una imagen de node:alpine, que se va a llamar vite_docker, que se va a poder llamar a través del puerto 8000 y que

el directorio actual se va a conectar a la ruta /srv/app, que es donde se cargará los documentos para que REACT los procese.

Abrimos esa carpeta en Visual Studio Code y abriendo el terminal vamos a poder ejecutar los comando correspondientes:



The screenshot shows the Visual Studio Code interface. The Explorer panel on the left shows a folder named 'REACT' with a file 'docker-compose.yml' selected. The main editor area displays the content of 'docker-compose.yml':

```
1 version: "3.4"
2 services:
3   vite_docker:
4     image: node:alpine
5     container_name: vite_docker
6     entrypoint: /bin/sh
7     ports:
8       - 8000:8000
9     working_dir: /srv/app
10    volumes:
11      - type: bind
12        source: ./
13        target: /srv/app
14    tty: true
```

At the bottom, the TERMINAL panel shows a PowerShell prompt: `PS C:\REACT>`. The status bar at the bottom indicates 'Ln 1, Col 1', 'Spaces: 2', 'UTF-8', 'CRLF', and 'Compose'.

Ejecutaremos `docker-compose up --build -d` para crear el container y se quede como demonio:

```
PS C:\REACT> docker-compose up --build -d
[+] Running 1/2
 - Network react_default Created
 ✓ Container vite_docker Started
PS C:\REACT> 
```

Crearemos un fichero que se llame `init.sh` y que contendrá lo siguiente:

```
#!/bin/sh

# Inicializamos proyecto

npm create vite@latest . -- --template react

# Añadimos la configuración del servidor para el Puerto 8000

echo "import { defineConfig } from 'vite'
import react from '@vitejs/plugin-react'

// https://vitejs.dev/config/
export default defineConfig({
  plugins: [react()],
  server: {
    host: true,
    port: 8000, // This is the port which we will use in docker
    // Thanks @sergiomoura for the window fix
    // add the next lines if you're using windows and hot reload doesn't work
    watch: {
      usePolling: true
    }
  }
})
" > vite.config.js
```

Ejecutaremos `docker-compose exec -it vite_docker "./init.sh"` para que se cree el script anterior en el servidor. Según el sistema operativo, es posible que tengas que darles permisos de ejecución con `chmod`.

Cuando nos pregunte si proceder, diremos que yes (y), y posteriormente la opción *ignore files and continue* ahora tendremos la estructura básica de un proyecto en docker.

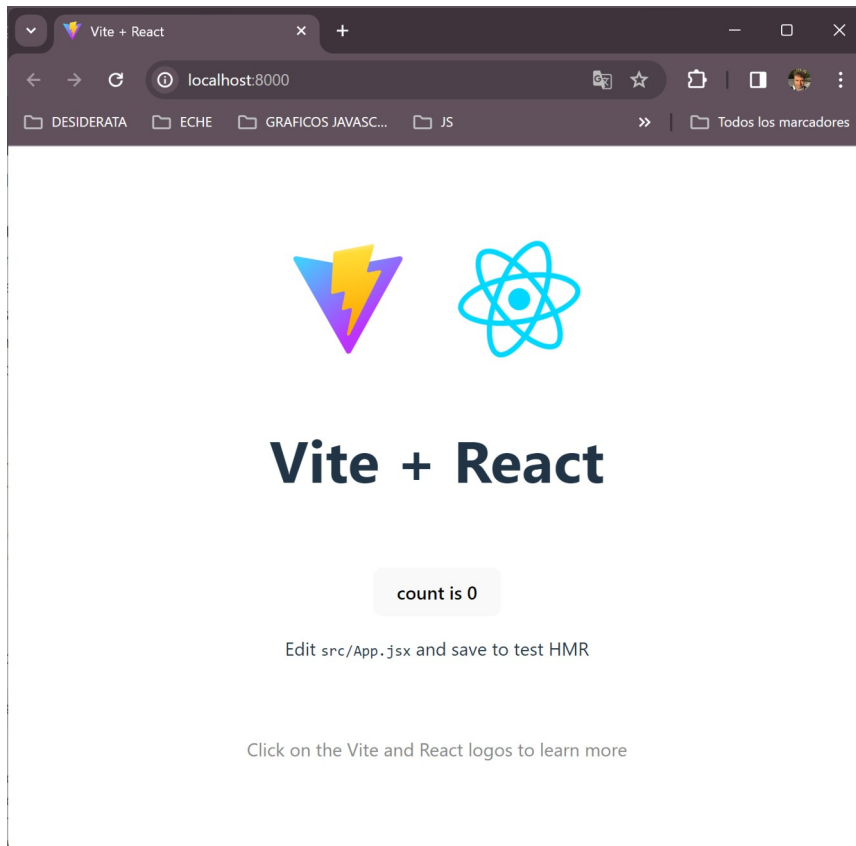
Si falta alguna dependencia, ejecutamos el comando:

```
docker-compose exec vite_docker npm install
```

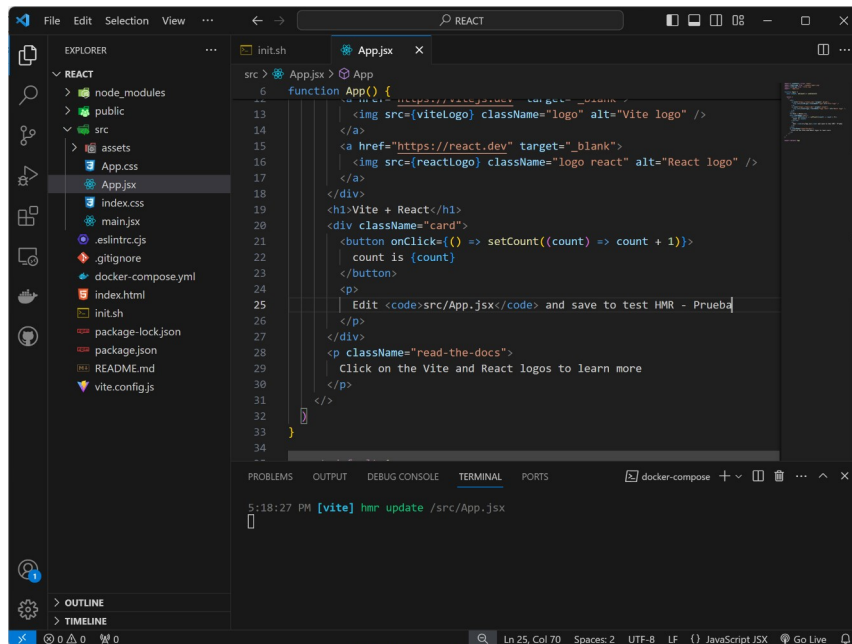
Sólo nos faltará lanzar el servidor en desarrollo:

```
docker-compose exec vite_docker npm run dev
```

Si abrimos un navegador al servidor localhost:8000, veremos la página inicial del proyecto:



Podemos hacer una prueba editando el fichero `src/App.jsx` y modificamos algo del fichero:



```
6 function App() {
  13   <img src={ViteLogo} className="logo" alt="Vite logo" />
  14   </a>
  15   <a href="https://react.dev" target="blank">
  16     <img src={reactLogo} className="logo react" alt="React logo" />
  17   </a>
  18 </div>
  19 <h1>Vite + React</h1>
  20 <div className="card">
  21   <button onClick={() => setCount((count) => count + 1)}>
  22     count is {count}
  23   </button>
  24 </p>
  25   Edit <code>src/App.jsx</code> and save to test HMR - Prueba
  26 </p>
  27 </div>
  28 <p className="read-the-docs">
  29   Click on the Vite and React logos to learn more
  30 </p>
  31 </>
  32 }
  33 }
  34 }
```

5:18:27 PM [vite] hmr update /src/App.jsx

Si ahora vamos a la página web, veremos que el contenido ha cambiado, tenemos REACT funcionando:

