



Interfaces de Usuario Web Avanzadas

Programación de aplicaciones para dispositivos móviles

Curso 2024/2025



Interfaz Web: Librerías

- Hacer modificaciones a través del DOM resulta tedioso.
- Existen modelos de Ingeniería del Software que han demostrado ser muy útiles para gestionar interfaces de usuario, pero el API del DOM no sería un buen ejemplo.
- Con la evolución de los navegadores y el uso de JavaScript se han habilitado librerías que permiten hacer interfaces de usuario de manera mucho más óptima y sencilla. Ejemplos:
 - [Angular](#) (Google)
 - [React](#) (Facebook)
 - [Vue.js](#)
 - [Svelte](#)

Interfaz Web: React (1/4)

- [React](#) es una librería de JavaScript que permite implementar aplicaciones web.
- Creada por Facebook en 2013 para construir interfaces de usuario.
- Desde entonces se ha mantenido por Facebook y la Comunidad (se publica como [código abierto](#)).
- El desarrollo en React es muy activo y es uno de los proyectos más utilizados para crear interfaces de usuario. Ejemplo: <https://www.facebook.com/>

Interfaz Web: React (2/4)

- Se distinguen dos tipo de aplicaciones:
 - **Single page applications** (SPA): Tiene componentes que todos están controlados por React. Cada componente de la página es un componente React. Por ejemplo, Facebook, Gmail y X (antes Twitter).
 - **Multi page aplicaciones** (MPA): No toda la página está controlada por React, el contenido se renderiza en el servidor.
- Está basado en componentes que gestionan su propio estado.
- Se ejecutan en el navegador no en el servidor.

Interfaz Web: React (3/4)

- React construye un árbol DOM según se van creando componentes en el sistema. Sin embargo, utiliza el DOM virtual.
- El DOM virtual es una representación interna de la aplicación en la que se almacena el estado actual del DOM (es decir, la estructura de la página), independientemente de cómo lo esté almacenando el navegador.
- Aunque aparentemente se modifique mucha parte del DOM a través de los componentes, React solamente aplica las diferencias entre el estado anterior y el nuevo (es decir, solamente realiza llamadas al DOM ante cambios de estado).
- De esta manera, el DOM virtual ofrece flexibilidad y mejora del rendimiento.

Interfaz Web: React (4/4)

- Ventajas:
 - Optimización de código.
 - Uso de características Next-Gen JavaScript: más rápido, escritura de código menos propensa a error... ([ECMAScript](#): ES6 vs ES5).
 - Más productivo (CSS).
- Requiere herramientas de gestión de dependencias: **npm** o **yarn** ([Node.js](#)).
- Disponibilidad del **boilerplate** para React: [create-react-app](#)

Interfaz Web: JSX (1/2)

- Extensión de la sintaxis de JavaScript que permite incluir etiquetas HTML dentro del mismo código. Ejemplo:

```
const element = <h1>¡Hola, mundo!</h1>;
```

- Además, permite interpolar contenido de JavaScript dentro de las etiquetas a través del uso de llaves ({}). Ejemplo:

```
const name = 'Richard';
```

```
const element = <h1>Hola, {name}</h1>;
```

- Sin embargo, la máquina virtual de los navegadores no entiende JSX:
 - Se requiere transformar un archivo .jsx a otro con extensión .js.
 - Se requiere transformar las etiquetas HTML a código JavaScript normal.

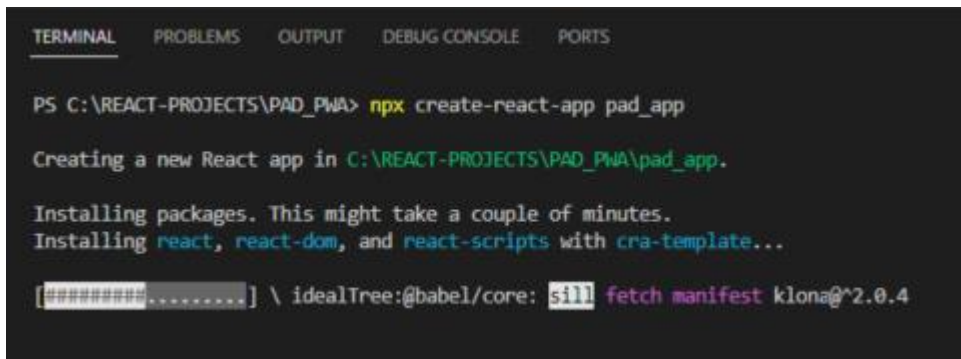
Interfaz Web: JSX (2/2)

- Se requieren herramientas (es decir, “transpiladores”), como por ejemplo, [Babel](#), para procesar JavaScript y transformarlo.
- Babel puede transformar JSX en JS a través de cierta configuración y puede transformar JavaScript “moderno” en JavaScript más antiguo para que funcione en navegadores sin compatibilidad.
- Se recomienda utilizar JSX con [React](#) para describir las interfaces de usuario.
- **JSX con Babel y React funciona muy bien** (soportado por muchos editores como Visual Studio Code), pero requiere desplegar muchas herramientas.

Interfaz Web: React – Creación de una app (1/2)

- Para la asignatura utilizaremos el editor de código [Visual Studio Code](#) y el navegador [Google Chrome](#).
- [Crear React App](#):

```
npx create-react-app pad-app
```



```
TERMINAL  PROBLEMS  OUTPUT  DEBUG CONSOLE  PORTS

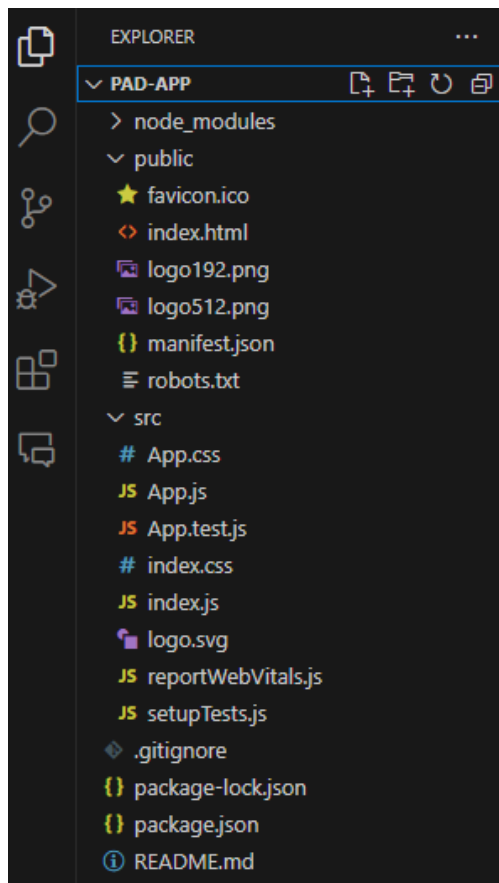
PS C:\REACT-PROJECTS\PAD_PWA> npx create-react-app pad_app

Creating a new React app in C:\REACT-PROJECTS\PAD_PWA\pad_app.

Installing packages. This might take a couple of minutes.
Installing react, react-dom, and react-scripts with cra-template...

[#####.....] \ idealTree:@babel/core: sill fetch manifest klonaf^2.0.4
```

Interfaz Web: React – Creación de una app (2/2)

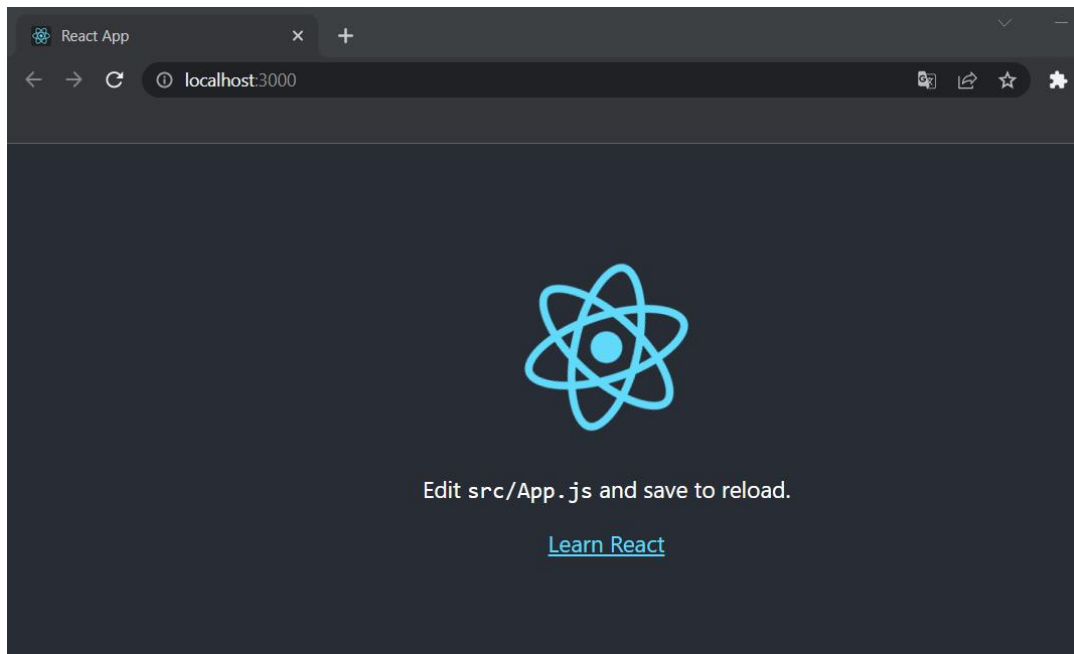


Interfaz Web: React – Ejecución de una app

- Ejecutar la aplicación:

```
cd pad-app
```

```
npm start
```



Interfaz Web: React – Componentes

- Con React se pueden definir **componentes** que representarán los objetos de interfaz de una aplicación.
- Un **componente** es un objeto JavaScript que representa un nodo complejo en el DOM (tiene que empezar siempre con una letra mayúscula).
- Distingue entre **dos tipos de componentes**:
 - Componentes funcionales.
 - Componentes de clase (**ya obsoletos aunque siguen funcionando**).
- De esta manera, las interfaces de usuario se crean de una manera óptima.

Interfaz Web: React – Componentes funcionales

- Funciones JavaScript: forma más sencilla de implementar un componente en React.
- Acepta un único argumento de objeto “props” con datos.

components/Home.js

```
function Home(props) {  
  return <h1>¡Hola, {props.name}!</h1>;  
}  
export default Home;
```

App.js

```
import Home from "../components/Home"  
...  
<Home name="Juan" />  
...
```

Interfaz Web: React – Componentes de clase

- Con un enfoque de orientación a objetos.
- Hereda de la clase **Component** de React.

components/Home.js

```
import React from "react";
class Home extends React.Component {
  render() {
    return <h1>¡Hola, {this.props.name}!</h1>;
  }
}
export default Home;
```

App.js

```
import Home from "../components/Home"
...
<Home name="Juan" />
...
```

Interfaz Web: React – Componentes High-Order

- Toma un componente como entrada y lo devuelve a la salida, pero con funcionalidades añadidas.

components/Home.js

```
const Home = (props) => {  
  return <h1>¡Hola, {props.name}!</h1>;  
}  
export default Home;
```

App.js

```
import Home from "../components/Home"  
...  
<Home name="Juan" />  
...
```

Interfaz Web: React – Combinación de componentes

- Para combinar componentes simplemente hay que utilizarlos como si se tratara de HTML. Ejemplo:

- Componente base: **components/Greeting.js**

```
function Greeting(props) {  
  return <h1>¡Hola, {props.name}!</h1>;  
}  
export default Greeting;
```

- Componente que utiliza el componente `Greeting`. Ejemplo: **components/Paragraph.js**

```
import React from "react";  
import Greeting from "../Greeting";  
class Paragraph extends React.Component {  
  render() {  
    return (  
      <p>  
        Esto es un párrafo con un saludo: <Greeting name="Juan" />  
      </p>  
    );  
  }  
}  
export default Paragraph;
```


Interfaz Web: React – Eventos

- También se puede programar la funcionalidad de responder a **eventos**.
- Para ello, en el “código HTML” se van a registrar los eventos JavaScript como si se trataran de etiquetas en un archivo .html. Ejemplo:

components/EventTest.js

```
function EventTest(props) {  
  const buttonClickHandler = (e) => {  
    alert(";Has pulsado el botón!");  
  };  
  
  return (  
    <>  
      <h1>Testing events</h1>  
      <button onClick={buttonClickHandler}>Pulsar</button>  
    </>  
  );  
}  
export default EventTest;
```

Interfaz Web: React – Estado (1/2)

- La renderización se produce cuando cambia el estado de un componente.

Ejemplo: **components/StateTest.js**

```
import { useState } from "react";  
import Home from "../Home";
```

```
function StateTest(props) {  
  const [inputText, setInputText] = useState("");  
  const [displayText, setDisplayText] = useState(false);  
  
  const inputTextHandler = (e) => {  
    const value = e.target.value;  
    setDisplayText(false);  
    setInputText(value);  
  };  
  
  const onSubmit = (e) => {  
    e.preventDefault();  
    setDisplayText(true);  
  };  
}
```

Interfaz Web: React – Estado (2/2)

```
return (  
  <>  
    <form onSubmit={onSubmit}>  
      <input type="text" onChange={inputTextHandler}></input>  
      <button type="submit">Aceptar</button>  
    </form>  
    {displayText ? <Home name={inputText} /> : null}  
  </>  
>;  
>  
  
export default StateTest;
```

Interfaz Web: React – Axios (1/4)

- Muchas aplicaciones web tienen que interactuar con una API REST en su desarrollo.
- [Axios](#): cliente HTTP que facilita la gestión de las peticiones. Similar a la [API FETCH de JavaScript](#).
- Axios está basado en *promises*, por lo que permite generar código asíncrono más legible a través de **async** y **await** de JavaScript.
- Instalar en Visual Studio Code a través de un terminal desde la carpeta de nuestro proyecto:

```
$ npm install axios
```

Interfaz Web: React – Axios (2/4)

- Ejemplo petición **GET** en un componente React:

```
import axios from "axios";
...
const [people, setPeople] = useState([]);
...
axios.get("https://api-url/users")
  .then((res) => {
    const users= res.data;
    setPeople(users);
  });

render() {
  return (
    <ul>
      { people.map(person => <li>{person.name}</li>)}
    </ul>
  )
}
```

Interfaz Web: React – Axios (3/4)

- Ejemplo petición **POST** en un componente React (el parámetro no va en la url):

```
import axios from "axios";  
...  
axios.post("https://api-url/users", { user })  
  .then((res) => {  
    console.log(res.data);  
  });
```

Interfaz Web: React – Axios (4/4)

- Ejemplo petición **DELETE** en un componente React (el parámetro va en la url):

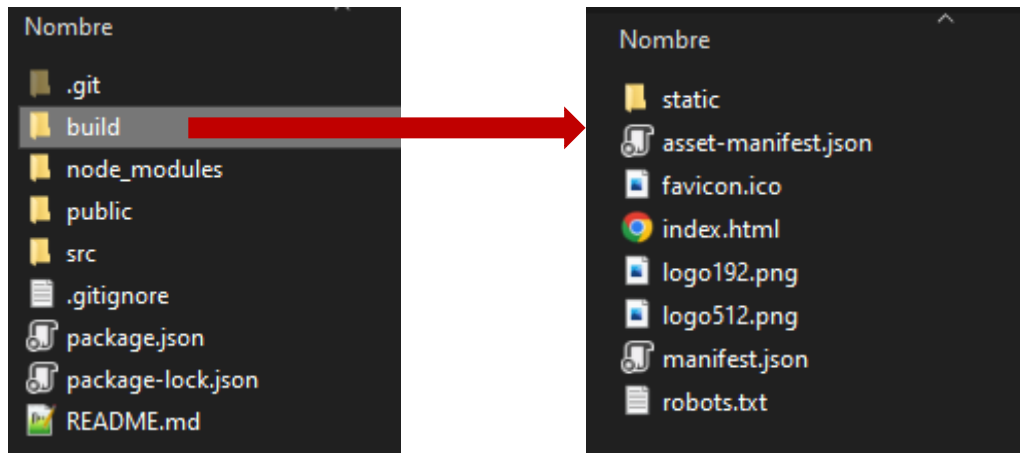
```
import axios from "axios";  
...  
axios.delete("https://api-url/users/" + {userID})  
  .then((res) => {  
    console.log(res.data);  
  });
```

Interfaz Web: React – Despliegue app (1/2)

- Para desplegar la app en un servidor ejecutar el siguiente comando desde la carpeta del proyecto:

```
npm run build
```

- Creará la carpeta “build” con los ficheros necesarios para su despliegue en un dominio.



Interfaz Web: React – Despliegue app (2/2)

■ Despliegue de un proyecto React a través de **GitHub Pages**:

- Instalar gh-pages desde la carpeta del proyecto:

```
npm install gh-pages
```

- Editar el fichero **package.json** y añadir la siguiente entrada antes de "dependencies":

```
"homepage": "https://<<profilegh>>.github.io/pad-app",
```

- Editar el fichero **package.json** y añadir la siguiente entrada en la parte de "scripts":

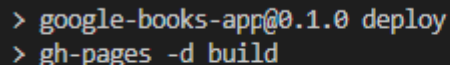
```
"predeploy": "npm run build",
```

```
"deploy": "gh-pages -d build",
```

- Ejecutar los siguientes comandos desde la carpeta del proyecto para desplegar la app (*Published* si todo ha ido bien):

```
npm run predeploy
```

```
npm run deploy
```



```
> google-books-app@0.1.0 deploy  
> gh-pages -d build
```

```
Published
```

```
PS C:\Projects\REACT-PROJECTS\google-books-app>
```

Enlaces de interés

- <https://react.dev/>
- <https://react.dev/blog/2023/03/16/introducing-react-dev>