



ARQUITECTURA
DEL SOFTWARE

2025-26

Pablo González
Celia Melendi
Diego Martín
Jose E. Labra



Escuela de
Ingeniería
Informática



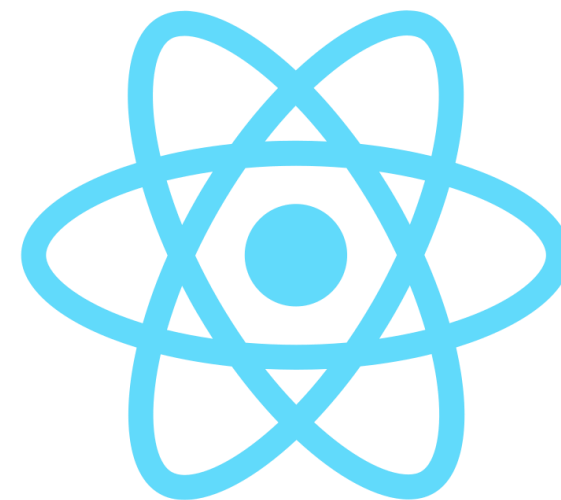
Universidad de Oviedo

Laboratorio 5

React.js
Automatización construcción
Gestión de dependencias

¿Qué es React.js?

- React.js = librería Javascript para construir interfaces de usuario para la web, así como aplicaciones móviles
 - Código abierto
 - Creado inicialmente Meta
 - Basada en componentes



¿Porqué React.js?

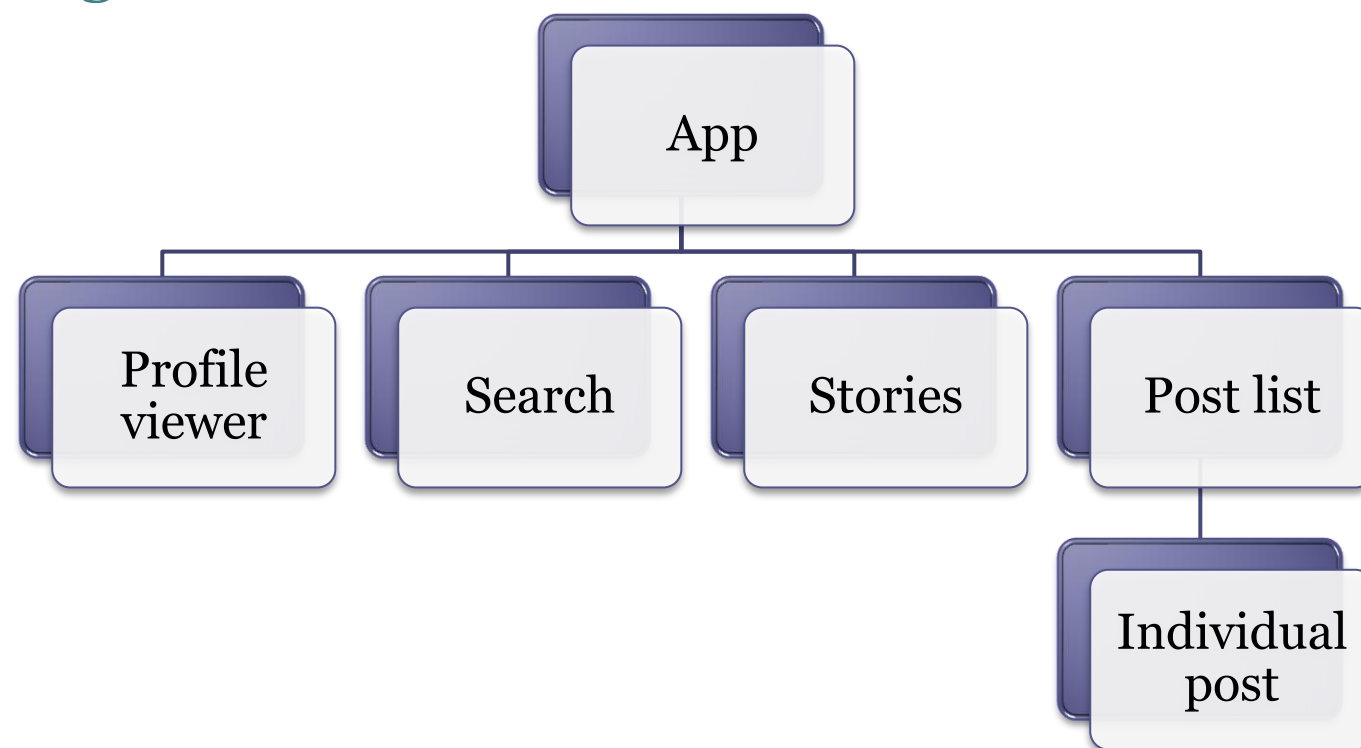
- Varias razones:
 - Popularidad
 - Simplicidad y fácil de aprender
 - Componentes reutilizables
 - También tiene posibilidad Native (React Native)
 - Ampliamente utilizada y muchas herramientas
 - Alta testabilidad

Componentes

Las páginas son modeladas usando componentes

Un componente es una parte del interfaz de usuario

Ejemplo: Instagram



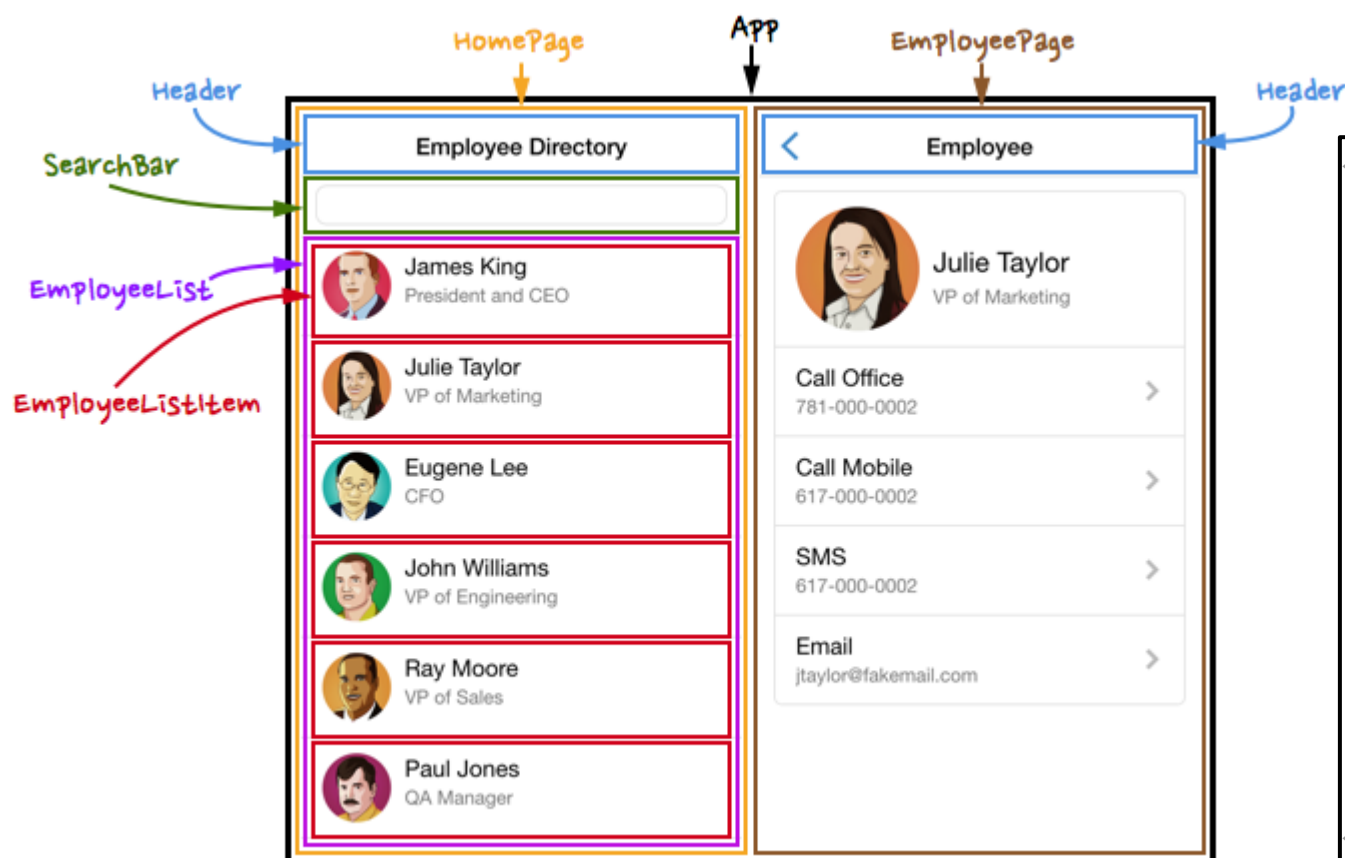
Componentes

Un componente puede ser implementado como una función (Hook) JavaScript

- Maneja el estado a través de hooks de estado
- Devuelve la UI que el componente va a mostrar en pantalla
- Cuando cambia el estado, React.js actualiza el elemento y sus hijos en memoria
- La representación de este elemento en memoria se llama Virtual DOM

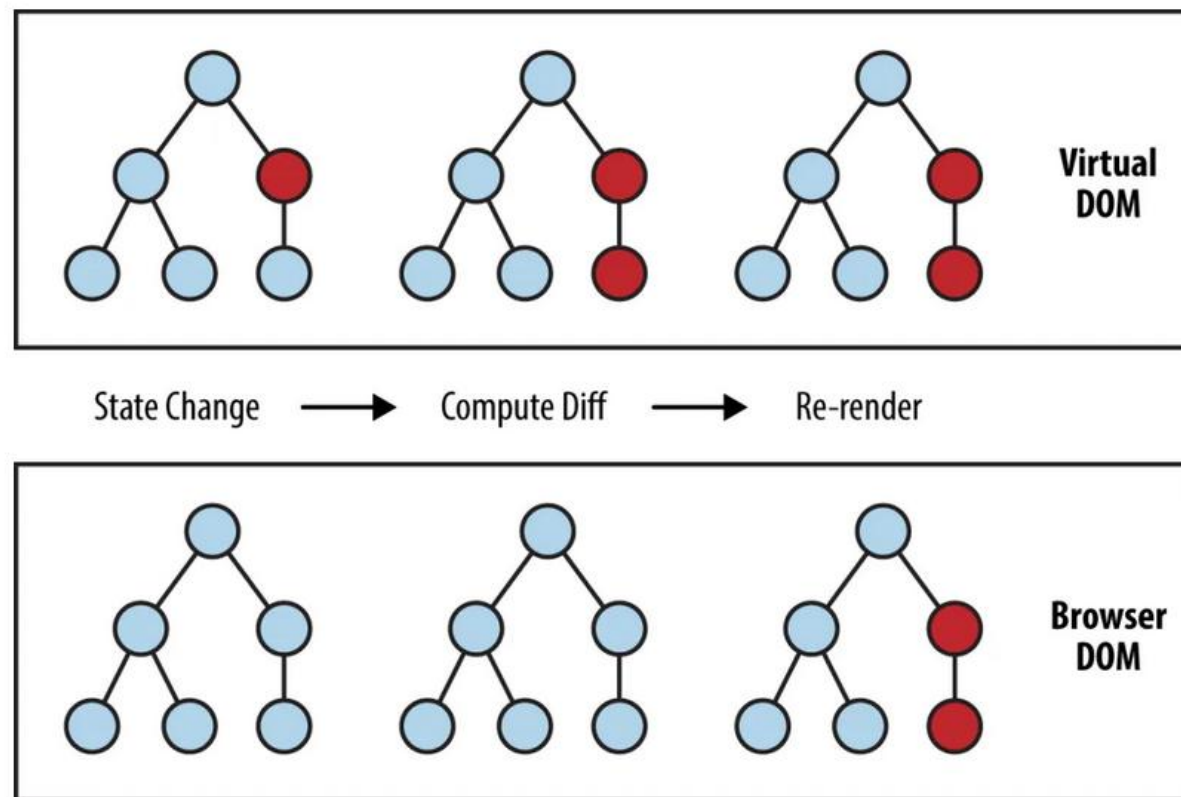
React.js
reacciona
a cambios

Ejemplo de componentes



```
<App>
  <HomePage>
    <Header />
    <SearchBar />
    <EmployeeList>
      <EmployeeListItem person="James King" />
      <EmployeeListItem person="Julie Taylor" />
      <EmployeeListItem person="Eugene Lee" />
      ...
    </EmployeeList>
  </HomePage>
  <EmployeePage>
    <Header />
    ...
  </EmployeePage>
</App>
```

Virtual DOM



JSX (Javascript XML)

JSX es una extensión sintáctica de Javascript

Puede parecer similar a HTML

Los ficheros JSX se compilan a Javascript automáticamente

Permiten el uso de expresiones dinámicas

Facilitan la creación de interfaces de usuario

JSX

```
const element = (  
  <h1 className="greeting">  
    Hello, world!  
  </h1>  
);
```


Utilización de Hooks

Recomendado a partir de versión 16

Substituye clases por funciones

En el ejemplo, la hook *useState()* gestiona los cambios de nombre en la app

Al pulsar el botón, el estado se cambia, el DOM virtual es actualizado, y la página se refresca automáticamente

```
const App = () => {  
  const [name, setName] = useState('World');  
  return (  
    <div className="App">  
      <h1>Hello, {name}!</h1>  
      <button onClick={() => setName('James')}>  
        Click me to change the name  
      </button>  
    </div>  
  );  
}
```

Documentación adicional

Enlaces de interés

Página de curso [Bootcamp Fullstack](#)

[Primera conferencia de Node.js](#) de Ryan Dahl

Vite (<https://vitejs.dev/>)

Permite realizar y visualizar cambios durante desarrollo

Construcción de Software

Tareas

Compilación

De código fuente a código binario

Empaquetado

Gestión de dependencias e integración

También llamado enlace (linking)

Ejecución de pruebas

Despliegue

Crear documentación/*release notes*

Automatización de la construcción

Automatizar tareas de construcción

Objetivos:

- Evitar errores (minimizar "*malas construcciones*")

- Eliminar tareas redundantes y repetitivas

- Gestionar complejidad

- Mejorar calidad de producto

- Tener un histórico de construcciones y *releases*

- Integración continua

- Ahorro de tiempo y dinero

Herramientas de automatización

makefile (mundo C)

ant (Java)

maven (Java)

sbt (Scala, lenguajes JVM)

gradle (Groovy, lenguajes JVM)

rake (Ruby)

npm (Node.js)

cargo (Rust)

etc.

npm

Node.js Package Manager

Creado inicialmente por Isaac Schlueter

Posteriormente empresa: Npm Inc.

3 cosas

1. Sitio web (<https://www.npmjs.com/>)

Gestión de usuarios y organizaciones

2. Almacén de software

Paquetes públicos/privados

3. Aplicación en línea de comandos

Gestión tareas y dependencias

Fichero configuración: package.json



npm: package.json (manifiesto)

Fichero configuración: package.json

Metadatos

Nombre, version, autor, licencias,...

Dependencias

Declaración de módulos requeridos

Para Desarrollo/ejecución

Scripts

Ciclo de vida: build, test, start

```
{
  "name": "...obligatorio...",
  "version": "...obligatorio...",
  "description": "...opcional...",
  "keywords": "...",
  "repository": {...},
  "author": "...",
  "license": "...",
  "bugs": {...},
  "homepage": "http://. . .",
  "main": "index.js",
  "devDependencies": {...},
  "dependencies": {...},
  "scripts": { "test": " ... " },
  "bin": {...},
}
```



Paquetes npm

Almacén: <http://npmjs.org>

Instalación de paquetes:

2 opciones:

Local

```
npm install <packageName> --save (--save-dev)
```

Descarga los contenidos de <packageName> en node_modules

Global

```
npm install -g <packageName>
```

Guarda dependencia en package.json

Solo para desarrollo



npm: package-lock.json

- Se genera al ejecutar npm install
- Contiene una instantánea de las dependencias
 - Registro exacto de lo que se instaló
 - Árbol completo (resuelve dependencias de dependencias)
- Objetivo
 - Reproducibilidad y exactitud (bloquea las versiones exactas)
 - Seguridad (incluye hashes del código descargado)
- Se recomienda incluir package-lock.json en control de versiones
 - Permite que se utilizan las mismas versiones
 - Aumenta seguridad e integridad

Dependencias npm

Gestión dependencias

Paquetes locales son guardados en `node_modules`

Acceso a través de: `require('...')`

Paquetes Global (instalados con opción `--global`)

Guardados en `/usr/local/npm` (en Linux)

Paquetes *Scoped* se marcan con `@`

Para usar un módulo dentro del proyecto u otro módulo

```
const uc = require('upper-case');
```



Comandos y scripts npm

npm tiene muchos comandos

start -> node server.js

test -> node server.js

ls lista paquetes instalados

...

Scripts personalizados:

run <nombre>

<https://docs.npmjs.com/cli-documentation/>



Paquetes npm

Dependencias: especificadas en el package.json

Package: Identificado por su nombre y versión

Reglas para los nombres:

- 214 caracteres o menos.

- No puede empezar por punto o guión bajo

- Los nuevos paquetes no pueden tener letras mayúsculas en los nombres

- El nombre formará parte de la URL, un argumento de la línea de commando y el nombre de un fichero.

- Por tanto, el nombre no puede contener caracteres no validos en URLs



npm - Reglas versiones

Versión del paquete: Debe ser parseable por [node-semver](#)

Rangos: Conjunto de comparadores que especifican versiones que satisfacen el rango.

Por ejemplo el comparador `>=1.2.7` permitiría 1.2.7, 1.2.8, 2.5.3, y 1.3.9, pero no 1.2.6 o 1.1.0.

Más en <https://docs.npmjs.com/misc/semver>



npm: Campos del package.json

Referencia: <https://docs.npmjs.com/files/package.json>

Campos:

- description
- keywords
- homepage: URL a la página principal del proyecto
- bugs: URL del rastreador de incidencias del Proyecto o/y la dirección de correo de reporte de las mismas.
- author, contributors,...



Campos del npm package.json

- files: Es una lista de patrones de archivos que describen las entradas a ser incluidas cuando tu paquete se instala como dependencia
- Los patrones de ficheros siguen una sintaxis similar al .gitignore, pero revertida:
 - Incluir un fichero, Directorio o patron global(*, **/*, entre otros) hará que ese fichero se incluya en el tarball cuando se empaquete.
 - Omitir el campo hace que por defecto incluya todos los archivos ["*"].



Ficheros npm incluidos

- Ciertos ficheros se incluyen siempre, independientemente de la configuración:
 - `package.json`
 - `README`
 - `CHANGES / CHANGELOG / HISTORY`
 - `LICENSE / LICENCE`
 - `NOTICE`
 - El fichero especificado en el campo “main”.



Campos del npm package.json

- **main:** module ID punto primario de entrada del programa
 - ID de un módulo, especificado de forma relativa desde la raíz del paquete.
 - Para la mayor parte de los paquetes, tiene sentido tener un script principal y habitualmente no mucho más.
- **browser:** Si el módulo va a ser ejecutado en el lado del cliente en un navegador, se debe usar este campo en vez de main.



Campos del npm package.json

- repository: el lugar donde reside el código

```
"repository": {  
  "type" : "git",  
  "url" : "https://github.com/npm/cli.git"  
}
```



Campos del npm package.json

- **config**: Usado para especificar los parámetros de configuración que persisten entre diferentes scripts-actualizaciones:

```
{  
  "name" : "foo" ,  
  "config" : { "port" : "8080" }  
}
```



npm: Dependencias

- **dependencies:** Especificadas en un objeto simple que mapea el nombre del paquete con su versión o rango de versiones:
 - El rango de versiones es una cadena que tiene uno o varios descriptores separados por espacios.
 - Los rangos de versiones se basan en versiones semánticas:
 - Ver <https://docs.npmjs.com/misc/semver>



Campos del npm package.json

- **devDependencies:** Dependencias requeridas en desarrollo, como por ejemplo las relacionadas con las pruebas unitarias.
- **URL dependencies:**
 - Se puede indicar URL de tarball en lugar de rango de versiones
 - El tarball se descargará e instalará localmente en tiempo de instalación

```
<protocol>://[<user>[:<password>]@]<hostname>[:<port>][:][/]<path>[#<commit-ish> | #semver:<semver>]
```



npm

- GIT URLs: Siguen la estructura:

```
<protocol>://[<user>[:<password>]@]<hostname>[:<port>][:][/]<path>[#<commit-ish>|#semver:<semver>]
```

- Ejemplo

```
git+ssh://git@github.com:npm/cli.git#v1.0.27  
git+ssh://git@github.com:npm/cli#semver:^5.0  
git+https://isaacs@github.com/npm/cli.git  
git://github.com/npm/cli.git#v1.0.27
```



cargo

- cargo: herramienta de automatización de construcción de Rust

Concepto	npm	Cargo
Manifiesto	package.json	Cargo.toml
Fichero Lock	package-lock.json	Cargo.lock
Librerías	Packages/modules	Crates
Registro	https://npmjs.com/	https://crates.io/
Carpeta descarga	node_modules/	target/
Binarios	node_modules/.bin	target/debug ó target/release

Fin