



ARQUITECTURA DEL SOFTWARE

2024-25

Jose Emilio Labra Gayo

Pablo González

Irene Cid Rico

Diego Martín



Escuela de
Ingeniería
Informática



Universidad de Oviedo

Laboratorio 5

React.js

Automatización construcción

Gestión de dependencias

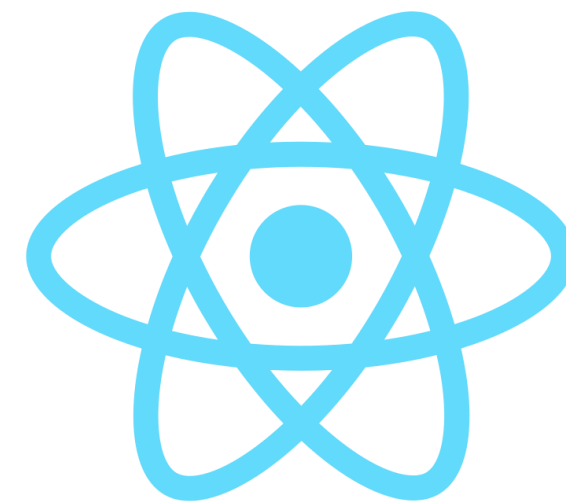
¿Qué es React.js?

- React.js = librería Javascript para construir interfaces de usuario para la web, así como aplicaciones móviles

- Código abierto
- Creado por Facebook (Meta)
- Basada en componentes

Como crear una aplicación react desde el scratch

- `npm create vite@latest my-app -- --template react`



¿Porqué React.js?

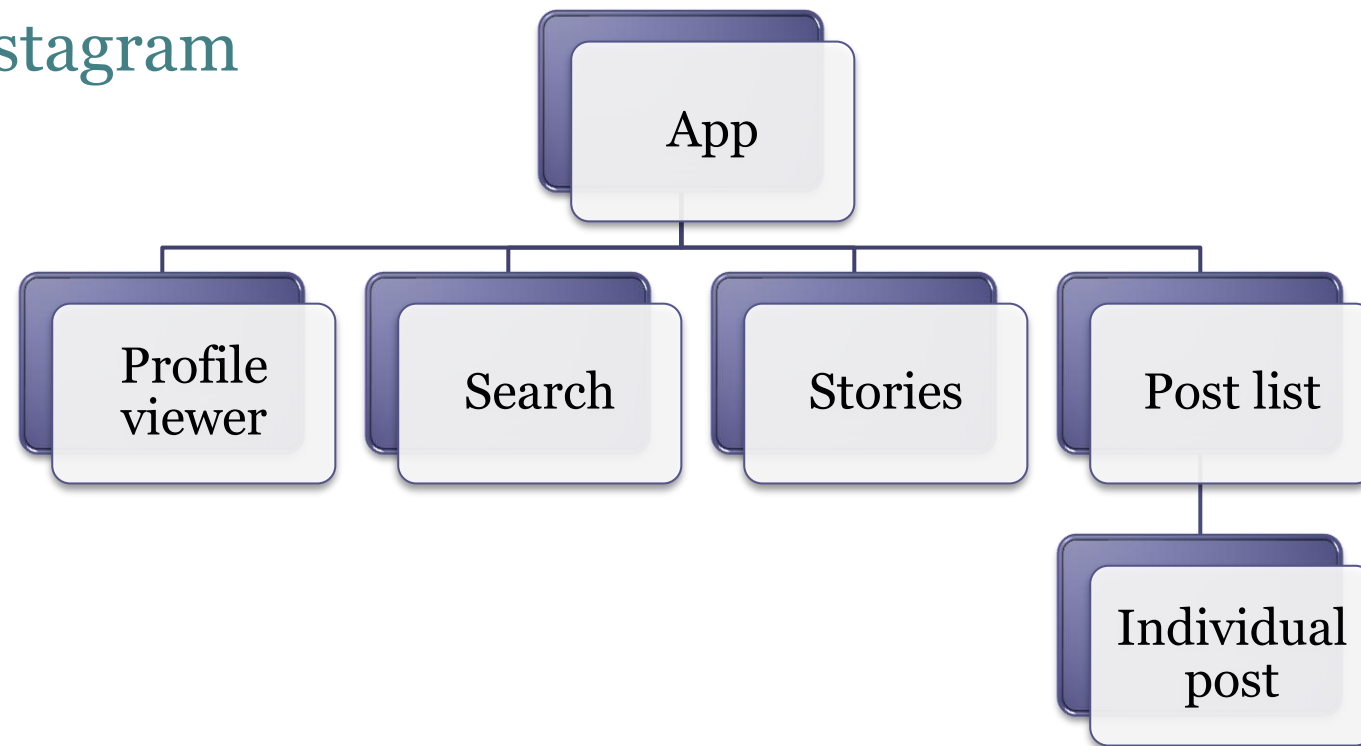
- **Varias razones:**
 - Popularidad
 - Simplicidad y fácil de aprender
 - Componentes reutilizables
 - También tiene posibilidad Native (React Native)
 - Ampliamente utilizada y muchas herramientas
 - Alta testabilidad

Componentes

Las páginas son modeladas usando componentes

Un componente es una parte del interfaz de usuario

Ejemplo: Instagram

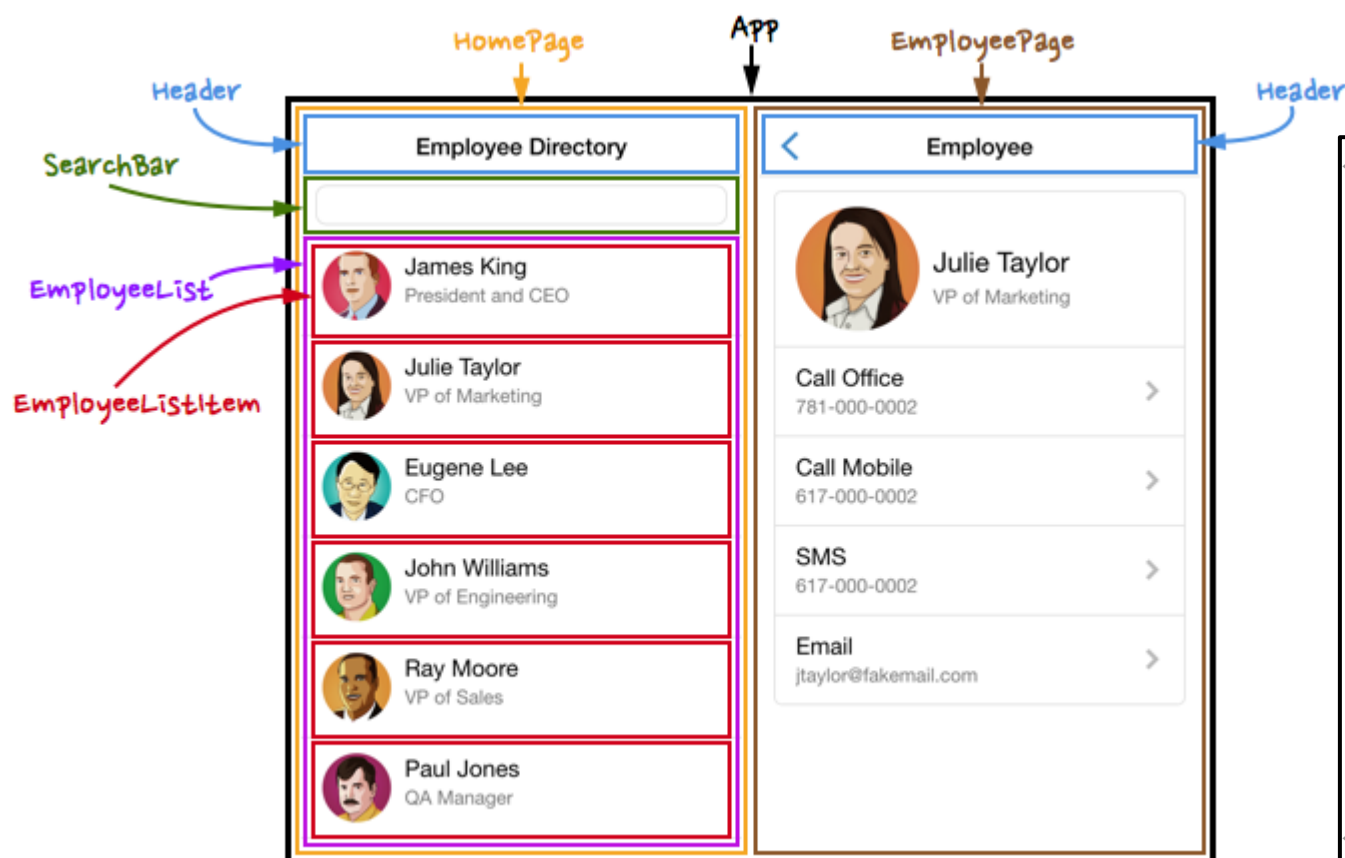


Componentes

- Un componente puede ser implementado como una función (Hook) JavaScript
- Puede tener estados(`useState`), subscribirse a eventos(`useEffect`), acceder a datos compartidos por varios componentes (`useContext`)
 - Y un método `render` que controla lo que se muestra en el interfaz de usuario
 - 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

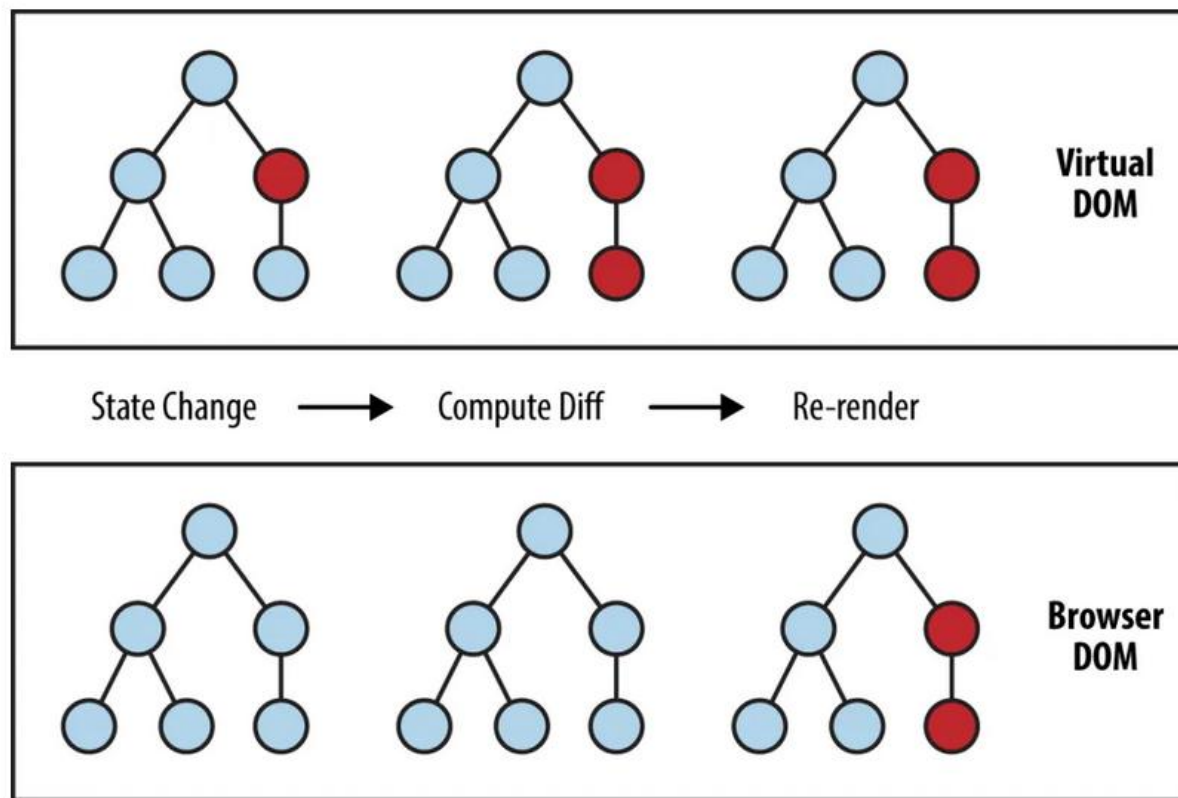


```

<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

JSX

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



Javascript

```
const element = React.createElement(  
  'h1',  
  {className: 'greeting'},  
  'Hello, world!'  
);
```


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

Ejercicios Estado React.js

[Ej1](#) Creamos un contador

[Ej2](#) Trabajamos con estados complejos (objetos)

[Ej3](#) Ejercicio con diferentes *handlers()*

[Ej4](#) Añadimos elementos a una lista

[Ej5](#) Cambiamos el comportamiento de un componente (color de fondo)

Documentación adicional

Ejercicios Renderización listas en React.js

[Ej1](#) Renderizar listas

[Ej2](#) Refactorización

[Ej3](#) Añadir elementos a la lista

[Ej4](#) Añadimos elementos desde un formulario

Documentación adicional

Ejercicios Programación asíncrona

[Ej1](#) *Fetch()* -> Hacer una petición a una API

[Ej2](#) *useEffect()*

[Ej3](#) Renderización condicional

[Ej4](#) Una refactorización

[Ej5](#) Peticiones utilizando librería *axios()*

Ejercicios TypeScript + React.js

[Ej1](#) Contador con TypeScript

[Ej2](#) Segundo ejercicio

[Ej3](#) Ejemplo de interface

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



Configuración npm: package.json

Fichero configuración: package.json

npm init crea un esqueleto simple

Campos:

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

Nota: Yeoman proporciona esqueletos completos



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



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

```
let 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>`

Tareas más complejas en NodeJs

Gulp, Grunt



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 comando y el nombre de un fichero. Por lo tanto el nombre no puede contener los caracteres no validos en URLs



NPM - Reglas versiones

- Versión del paquete: Debe ser parseable por [node-semver](#), que está empaquetada con npm a través de una dependencia
- 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>



Campos del npm 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.
- people fields: Autor, contribuyentes.
 - El autor “author” es una persona. “contributors” es una lista de personas. Cada contribuyente tiene asociado un objeto “person” con el campo “name” y opcionalmente “url” y “email”



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 de tu programa
 - Debe de ser el 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.
 - Puede ser de ayuda avisar a los usuario que pudieran estar usando ciertas primitivas (e.j. ventanas) que no están disponibles en Node.js.



Campos del npm package.json

- repository: el lugar donde reside el código

```
"repository": {  
  "type" : "git",  
  "url" : "https://github.com/npm/cli.git"  
}  
  
"repository": {  
  "type" : "svn",  
  "url" : "https://v8.googlecode.com/svn/trunk/"  
}
```



Campos del npm package.json

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

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



Campos del npm package.json

- **dependencies:** Son 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:**
 - Puedes especificar la URL de un tarball en vez de un rango de versiones.
 - Este tarball se descargará e instalará localmente en tu paquete 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
```



Task Execution : Grup y Gulp

Ejecutar tareas propias de JavaScript:

- Comprimir imágenes
- Empaquetar los módulos que van a ser usado en un proyecto (webpack)
- Minimizar ficheros js y css
- Ejecutar test
- Transcompilar – babel.js

Estas tareas pueden ejecutarse directamente con npm o pueden usarse dos herramientas muy famosas: Gulp y/o Grunt



Task Execution : Grup y Gulp

- Grup:

- Escrito sobre NodeJS.
Módulo fs
- Instalar:

```
npm install -g grunt  
npm install -g grunt-cli
```

- Configuración package.json

```
{  
  "name": "ASW",  
  "version": "0.1.0",  
  "devDependencies": {  
    "grunt-contrib-jshint": "~0.10.0",  
    "grunt-contrib-nodeunit": "~0.4.1",  
    "grunt-contrib-uglify": "~0.5.0"  
  }  
}
```

- Gulp:

- Escrito sobre NodeJS:
módulo stream
- Instalar:

```
npm install --save-dev gulp  
npm install -g gulp-cli
```

- Crea un gulpfile.js

```
function defaultTask(cb) {  
  // tareas  
  cb();  
}  
exports.default = defaultTask
```



Ejemplos

Wrapper

```
module.exports = function(grunt) {  
  // CONFIGURE GRUNT  
  grunt.initConfig({  
    (pkg.name)  
    pkg: grunt.file.readJSON('package.json'),  
  });  
  grunt.loadNpmTasks('grunt-contrib-uglify');  
  grunt.registerTask('default', ['uglify']);  
};
```

Wrapper

```
gulp.task('jpgs', function()  
{ return gulp.src('src/images/*.jpg')  
  .pipe(imagemin({ progressive: true }))  
  .pipe(gulp.dest('optimized_images')); });
```



Fin