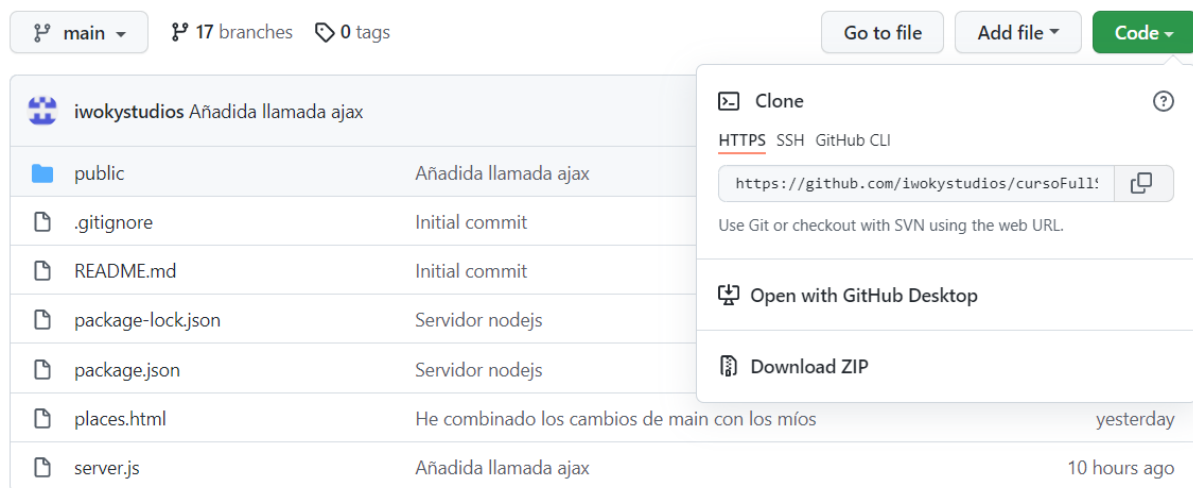


# Semana 1 - Martes - Git : compartir código en un equipo, Recorrer listas en javascript Map, Formato JSON, Funciones Arrow y funciones normales

1. Crear cuenta en Github.
2. Descargar Git Bash.
3. Descargar Github Desktop.
4. Clonar un repositorio Git
5. Crear branch propia.
6. Descargar código de una branch.
7. Integrar código de una branch a otra branch.

## Clonar un repositorio



```
git clone https://github.com/iwokystudios/cursoFullStack.git
```

## Crear una branch propia

```
git checkout -b nombreDeLaNuevaBranch
```

## Descargar código de una branch

Situarse en una branch concreta:

```
git checkout main
```

Descargar código de la branch:

```
git pull
```

## Descargar código de una branch

Si tenemos cambios locales, debemos hacer un commit antes de combinar cambios de otra branch:

```
git add .  
git commit -m "Mensaje que describe los cambios que habéis hecho"
```

Bajamos los últimos cambios de la branch que necesitamos integrar en la nuestra:

```
git checkout main  
git pull
```

Nos situamos en nuestra branch y combinamos los cambios descargados de la otra branch:

```
git checkout mibranch  
git merge main
```

## Recorrer listas en JS

Las listas en JavaScript disponen de una función para facilitar hacer **recorridos sobre todos sus elementos**. **map**

La función `.map` recibe como parámetro una función. Esa función es la que se va a ejecutar por cada uno de los elementos de la lista.

Esa función recibe dos parámetros, el primero es el elemento actual del recorrido que está llevando a cabo `map` y el segundo parámetro es la posición de ese elemento en la lista.

```
const lista = ["Spiderman", "Venom", "Hulk", "Ironman"];  
  
lista.map((elementoActual, posicion) => {  
  console.log( elementoActual, posicion );  
});
```

## Funciones

Las funciones se puede definir de varias formas en JavaScript:

1. `function nombreFuncion(parametroA, parametroB) { ..... }`
2. `const nombreFuncion = (parametroA, parametroB) => { ..... }`
3. `const nombreFuncion = function(parametroA, parametroB) { ..... }`

La función 2 se denomina arrow, debido a que se define empleando una flecha `=>`

La función 3 es equivalente a la arrow, no incluye ningún nombre de función por ello se denomina función anónima.

# Semana 1 - Miércoles - Cargar contenido en una página haciendo una petición a un servidor

## 1. Instalar NodeJS

### Cómo descargar los cambios desde casa

Si tenéis cambios en local no podréis descargar lo nuevo hasta que hagáis un commit.

```
git add .  
git commit -m "Mensaje que describe los cambios que habéis hecho"  
git push
```

### Iniciar el servidor NodeJS

Si es la primera que vez que os bajáis el proyecto en un ordenador, necesitaréis instalar todos los packages que necesita el servidor para ejecutarse. Con la siguiente instrucción:

```
npm install
```

Para iniciar el servidor:

```
node server.js
```

# Semana 1 - Jueves - Comunicación entre cliente y servidor

Básicamente tenemos código que se ejecuta en 2 sitios distintos:

Frontend y Backend  
Cliente y servidor  
Navegador y una ordenador en algun lugar

Estas tres descripciones son perfectamente equivalentes.

## EL FRONTEND

En el front tenemos el código HTML con referencias a código CSS y JavaScript.  
Desde el **Frontend** hacemos peticiones al **Backend**.

Palabras importantes sobre la comunicación:

Las peticiones que hacemos desde el cliente las llamamos **REQUEST**. Las respuestas a estas peticiones las llamamos **RESPONSE**.

Desde el navegador nos comunicamos con el servidor via AJAX. Esto lo hacemos gracias al uso del objeto **XMLHttpRequest**.

La comunicación vía HTTP más común es que enviamos tanto la Request en formato texto (string) y el servidor nos responde en formato texto (string).

NAVEGADOR		SERVIDOR
Si queremos enviar un objeto al servidor antes tenemos que convertirlo en texto. <b>JSON.stringify( objeto )</b>	-----Texto ----- >	<b>req.body</b> contiene el objeto que nos han enviado desde el cliente
Cuando sabemos que el servidor va a enviarnos un objeto, debido a que el servidor envía los datos en formato texto tenemos que convertirlos a un objeto. <b>JSON.parse ( texto )</b>	< ----- Texto -----	

## EL BACKEND

En el backend tenemos un abanico de tecnologías disponibles enorme, así como de lenguajes.

NodeJS (JavaScript y Typescript), PHP, Java, Python, Scala, ...

**Algo vital, si desde el Front queremos hacer una petición a una dirección concreta del backend. Debemos definir en el backend cómo procesar esa petición.**

```
app.post('/user', (request, response) => {  
  console.log( request.body );  
  response.json( { msg: "Usuario creado con éxito" } );  
});
```

## Instalar Spring Tool Suite en Eclipse

## 8. Help > Eclipse Marketplace


Eclipse Marketplace

Select solutions to install. Press Install Now to proceed with installation.  
Press the "more info" link to learn more about a solution.

Search Recent Popular Favorites Installed Giving IoT an Edge

Find: sts All Markets All Categories Go


**Spring Tools 3 Add-On for Spring Tools 4 3.9.21.RELEASE**

 Spring Tools 3 Add-On for Spring Tools 4 Attention: This add-on pack provides additional components from the previous Spring Tools 3 generation to be installed... [more info](#)

by VMware, EPL  
[J2EE](#) [spring](#) [Spring IDE](#) [Cloud](#) [jee](#)

★ 318 Install: 509K (10.324 last month) **Install**

**Spring Tools 4 (aka Spring Tool Suite 4) 4.14.0.RELEASE**

 Spring Tools 4 is the next generation of Spring Boot tooling for your favorite coding environment. Largely rebuilt from scratch, it provides world-class support... [more info](#)

by VMware, EPL  
[spring](#) [Spring IDE](#) [Cloud](#) [Spring Tool Suite](#) [STS](#)

★ 3474 Install: 2,17M (33.463 last month) **Install**

**Minimalist Gradle Editor 1.0.1**

 <https://github.com/Nodeclipse/GradleEditor> Minimalist Gradle Editor for build.gradle files with highlight for keywords, strings and matching brackets and

**Marketplaces**



? < Back Install Now > Finish Cancel

9. Aceptar los checkboxes.

10. Restart Eclipse.