Curso Full Stack
Clase 21

En la clase de hoy retomaremos las nociones de GIT y GitHub. Veremos qué es cada uno, cómo implementar un repositorio y cómo subir contenido en él.

Además nos introduciremos en el mundo de la programación orientada a objetos con JavaScript.

GIT

GIT es una tenología desarrollada por Linus Trovals el creador Linux, en el 2005.

Su cometido es versionar el código de una manera que nos sea sencillo:

- Trabajar en equipo sin que se pisen las disversas versiones del proyecto.

- Generar un track de cada desarrollo.

Todo proyecto estará contenido en un REPOSITORIO. Éste será un espacio de memoria en donde almacenaremos los documentos pertenecientes a nuestro desarrollo.

En el caso de tener los permisos, cada repositorio podrá ser clonado por los programadores.

Hay una rama, que se la conoce como MASTER, que contendrá, en principio, la primera versión del proyecto.

Luego, cada programador deberá generar (por cada feature o fix que tenga) una rama en la que hará uso del CICLO DE VIDA del repositorio.

Básicamente, el ciclo de vida de un repositorio:

1. Se inicializa el repo;
2. Se genera el branch (o rama) Master;
3. Cada programador clona el proyecto;
4. En cuanto surge una feature o fix:
    - Se inicia un nuevo brach
    - Se hacen los cambios y se comittean
    - Se genera el pull request
5. Vuelve al paso 4.

Lo que acabamos de estudiar es la tecnología de GIT.
Sin embargo, el repositorio debe quedar almacenado en algún lugar físicamente. Por este motivo se crearon varios sitios que implementan la tecnología de GIT y los suben a "sus nubes".
Entre los mas conocidos están:
GitTHub, GitLab, Azure y BitBucket.

GITHUB

Crearemos la configuración básica en github, para ésto vamos a la página oficial: www.github.com
Para loguearnos podemos crear una cuenta a partir de nuestro mail.
Una vez creado y dentro del dashboard vas a encontrar a la izquierda un botón en verde que dice NEW. Presionalo.

De todo el formulario sólo cargá el nombre (el resto lo veremos en la clase). Asegurate que sea solo un nombre o unilos con un guión medio(-).

Andá al pie de la página y presioná CREATE REPOSITORY.

Tenemos nuestro repo creado!!

Comandos para subir/actualizar contenido:

init: Le indica al proyecto que utilizaremos git.

add: Deja reflejados los cambios que generamos a nivel local.

commit: Trackea internamente un código que utilizará como cabecera del registro a versionar.

push: Sube los cambios comiteados a la nube.

Comandos para obtener contenido:

branch: Muestra las ramas creadas.

checkout: Cambia de branch.

merge: Fusiona la rama local con la del repositorio remoto.

fetch: Obtiene los cambios de la rama remota.

pull: Obtiene los cambios de la rama remota y los fusiona con la rama local.

OBJETOS EN JAVASCRIPT

A partir de la versión 2015 de EcmaScript podemos utilizar clases en nuestros desarrollos.

¿Qué es la programación orientada a objetos?
Es un paradigma de programar que centra la manera de programar en objetos.
Los objetos son abstracciones de todo aquello que nos rodea en nuestra vida. Pueden tener variaciones como formas y color, tienen identidad y realizan acciones entre ellos.

Los objetos se contruyen a partir de clases.
Las clases son moldes que contienen la estructura básica que puede contener un objeto.

Las clases contienen un constructor, atributos y métodos.
Además basándose en "clases padres" se pueden heredar características hacia nuevas clases.

Estructura de una clase I.

```
class NombreDeLaClase {
        constructor(parámetro1, ... parámentroN){
                this.xxxx = parámetro1;
                this.yyyy = parámetroN;
        }
}
```

class es una palabra reservada que se encarga de definir que lo que contiene es una clase.
NombreDeLaClase es precisamente el nombre que le queremos dar a la clase. Note que comienza con mayúscula.
constructor es el sector en donde se definen los atributos (variables). Note que delante de cada atributo lleva un **this.**

Para crear un objeto requerimos la palabra reservada **new**.

```
const objeto1 = new NombreDeLaClase(param1);
```

En caso de pasarle parámetros lo hacemos entre sus paréntesis.

Con el log de la consola veamos el contenido:

```
console.log('objeto1 ', objeto1);
```

Estructura de una clase II.

```
class NombreDeLaClase {
        constructor(...)
        metodo1(param1, paramN){
                return this.XXXX;
        }
}
```

Luego de generar el constructor podemos generar los métodos. Éstos son funciones. No necesitamos definirlos como tal (ni como arrow functions).
Se le pueden pasar valores como parámetros.
Deben contener un **return**.

Estructura de una clase III.

```
class NombreDeLaClase {
        constructor(...)
        static metodo1(param1, paramN){...}
        get gMetodo1(){
                return this.metodo1();
        }
}
```

Los getters son métodos que serán invocados desde el objeto.
Se utilizan para retornar el valor de algún método que se encuentre de manera estática (static) dentro de la clase.
El return apunta al this del método a devolver.

Volvamos al objeto que instanciamos anteriormente.

```
const objeto1 = new NombreDeLaClase();
```

Y ahora accedemos a su getter:

```
console.log(objeto1.gMetodo1);
```

Estructura de una clase IV.

```
class NombreDeLaClase {...}
class ClaseHeredada extends NombreDeLaClase{
        constructor(){
                super(paramZ);
                this.ZZZZ = paramZ;
        }
}
```

La herencia permite generar subclases que tomen los atributos y métodos de la clase padre y que puedan agregarle los propios.
La nomenclatura contiene:
extends: que significa **herenda de...** (o extiende).
super: Se usa en el constructor y nos indica que podemos heredar los atributos de la clase padre.

¡Nos vemos en la próxima clase!

@programaDesdeTuCasa

Programa Desde Tu Casa

Programa Desde Tu Casa

ProgramaDesdeTuCasa