Introducción al Backend con NestJS

Winter Sessions 2023

Javier de Arcos
@Javier DeArcosTL

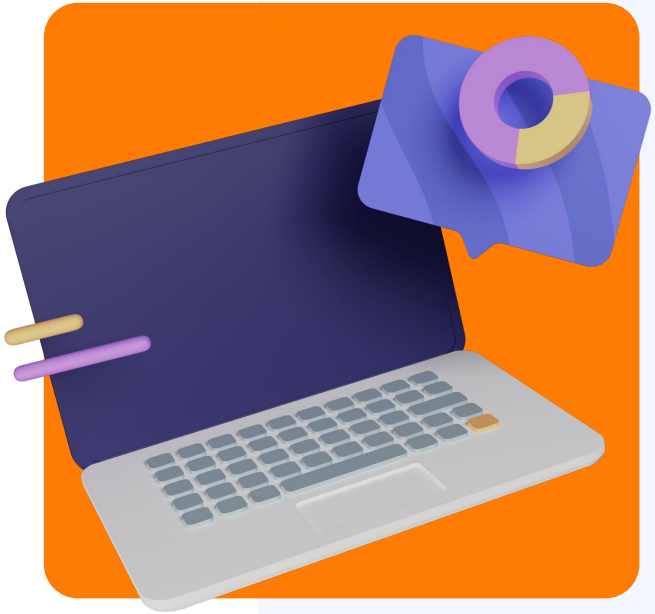
Última sesión

Introducción al Backend

En esta última sesión de las Winter Sessions 2023 haremos una introducción al Backend, conociendo alguno los conceptos principales de este stack y haciendo nuestro propio servicio web con NestJS

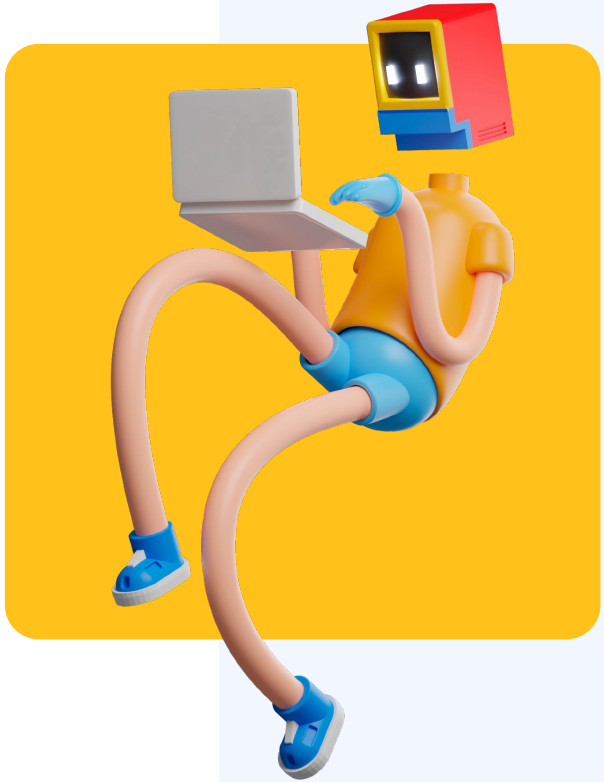
Requisitos

1. NodeJS
2. Docker
3. IDE de desarrollo (VS Code)
4. Cliente HTTP (Postman)



Temario

1. ¿Qué es el Backend?
2. NestJS y TypeScript
3. Primeros pasos con NestJS
4. Gestión de datos
5. Nest CLI
6. Resumen



Profesor

Javier de Arcos

Profe

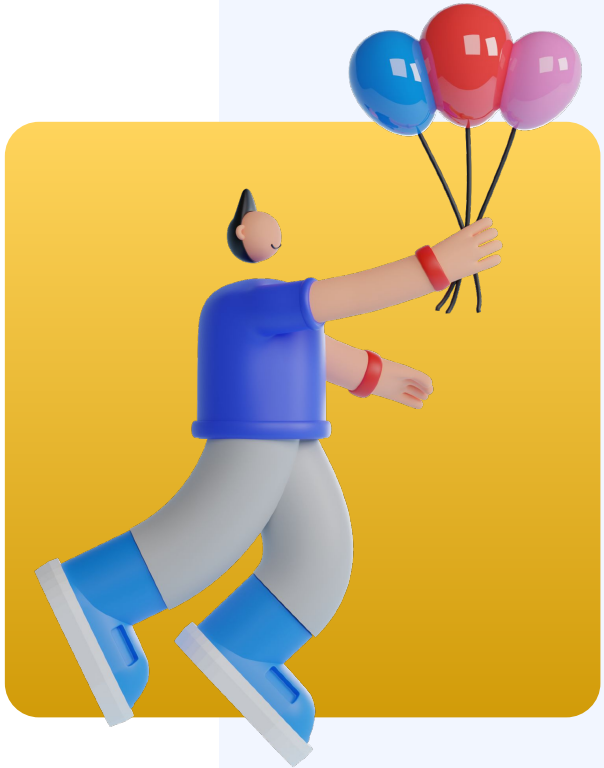
- Engineering Manager en Celonis
- 10 años de experiencia desarrollando SW
- 5 años liderando equipos
- Experiencia en campos como IA, IOT, Desarrollo Web

- Padre de 2 hijos maravillosos
- Me gusta leer, la música, aprender cosas nuevas
- Soy de perros y de pizza con Piña

Teach[t3ch]

Somos un grupo de apasionados de la programación que nos hemos unido para ayudar a gente como tú a perderle el miedo a programar. Queremos transmitirte la misma pasión que sentimos nosotros por nuestro trabajo y que veas que es posible conseguir eso que te ronda hace tiempo.

https://www.teacht3ch.com/



Teach[t3ch]

Empezamos!

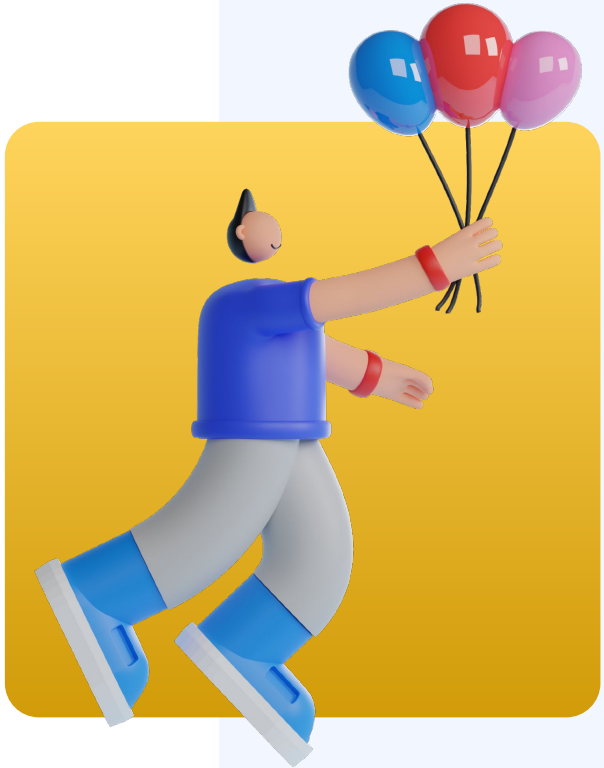
“Repeat something important & say something new”

-

David Heinemeier Hansson (DHH)

Filosofía

Partir de lo conocido hacia lo desconocido, relacionando, conectando, para llegar más allá.



¿Qué es Frontend?

Son todas las tecnologías de diseño y desarrollo web que corren en el navegador y que se encargan de la interactividad con los usuarios.

¿Qué es el Backend?

Es la parte del software que no es visible ni directamente accesible por los usuarios. Son tecnologías que corren en un servidor que es capaz de entender la forma en la que el navegador hace solicitudes.

El backend se suele encargar de la gestión de los datos (bases de datos), gestión de ficheros, lógica de negocio, envío de notificaciones (push y correo electrónico), autenticación, …

¿Cómo lo hace?

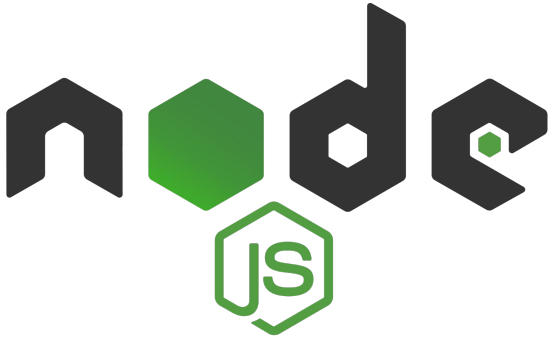
Ofreciendo servicios a otros software, máquinas a los que puedes acceder a través de internet que son accesibles a través de internet (REST, GraphQL, gRPC, SOAP, APIs asíncronas, …)

¿Qué es Node.js?

Node.js es un entorno de ejecución para JavaScript construido con V8, motor de JavaScript de Chrome.

Gracias a frameworks como Express, facilitó el desarrollo de aplicaciones Backend usando JavaScript, lenguaje más popular entre desarrolladores web.

Popularizaron stacks como **MEAN** y **MERN**, muy usados durante mucho tiempo y desplazaron otros más antiguos como LAMP.



¡Vamos a hacer nuestro primer Backend con Node.js y Express!

¿Qué es un Framework?

Un framework es un marco de trabajo, es decir, un conjunto estandarizado de conceptos, prácticas y criterios para enfocar un cierto tipo de problemas que nos sirve como referencia para resolver problemas similares.

“Es un framework de Node.js para crear aplicaciones del lado del servidor eficientes, confiables y escalables.”

Opinionated framework: Es un framework que te guía en la forma de hacer las cosas. Para ello define una forma “correcta” de hacer las cosas que facilita el desarrollo y reduce las decisiones a tomar siguiendo esa forma, aunque dificulta salirte de ella.

“JavaScript con sintaxis para tipos”

Es un lenguaje de **programación fuertemente** tipado construido encima de JavaScript.

Esto nos va a permitir obtener una mayor ayuda por parte del IDE al desarrollar y mayor facilidad de encontrar errores ya que los vamos a encontrar en tiempo de compilación (o desarrollo) y no de ejecución.

Instalar TypeScript y NestJS y crear nuestro primer proyecto

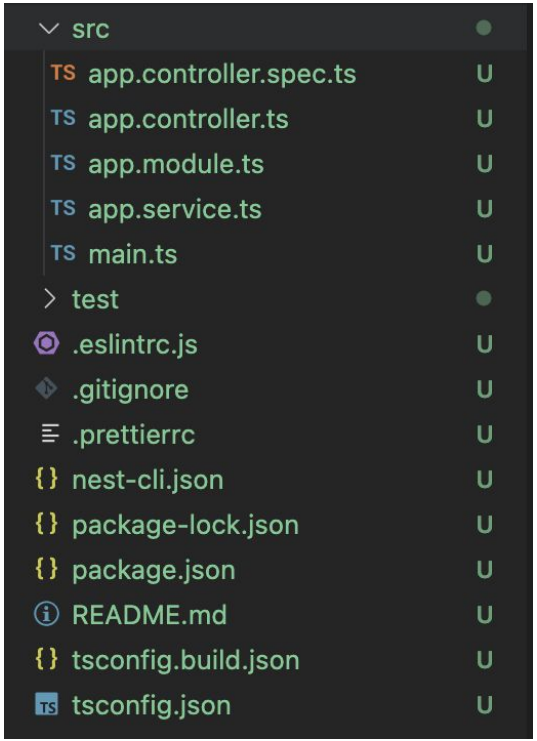
```
# Instalar TypeScript
npm install -g typescript

# Instalar NestJS
npm install -g @nestjs/cli

# Crear nuestro primer proyecto de NestJS
nest new project-name
```

NestJS hace un gran uso de **clases y decoradores**

Las **clases** pertenecen al mundo de la Programación Orientada a Objetos (POO) y nos van a permitir familiarizarnos con él.

La POO es un paradigma de programación cuya base son los objetos y la interacción entre ellos. Aquí iremos viendo conceptos como la herencia, la cohesión, la abstracción, el polimorfismo, el acoplamiento y el encapsulamiento

Los **decoradores** son un patrón de diseño clásico que permite añadir funcionalidades a un objeto "envolviendo" la funcionalidad original con otras nuevas.



Módulos

Los **módulos** son el elemento principal que usa NestJS para organizar la estructura de la aplicación.

Es una clase anotada con el decorador **@Module** con la metainformación que permite a NestJS resolver las relaciones y dependencias.

Los módulos pueden estar anidados.

Code Title here. Show some code!

```
import { Module } from '@nestjs/common';
import { AppController } from './app.controller';
import { AppService } from './app.service';

@Module({
 imports: [],
 controllers: [AppController],
 providers: [AppService],
 exports: []
})
export class AppModule {}
```

APIs REST

API que sigue los **principios REST**. Son APIs orientadas a recursos, que siguen el protocolo HTTP y aprovechan el potencial de los enlaces.

Son uno de los tipos más comunes de APIs en internet.

Los controladores son los elementos responsables de recibir las peticiones entrantes de los clientes y darles respuesta.

Utilizan el decorador **@Controller** junto con otros como @Get, @Post, @Param, @Body, …

Code Title here. Show some code!

```
import { Controller, Get } from '@nestjs/common';
import { AppService } from './app.service';

@Controller()
export class AppController {
 constructor(private readonly appService:
AppService) {}

 @Get()
 getHello(): string {
   return this.appService.getHello();
 }
}
```

Pipes

Los pipes o tuberías trabajan con los argumentos que utilizan los manejadores de las rutas de los controladores.

Sirven para **transformar o validar** estos argumentos.

La validación es una buena práctica que consiste en asegurar que recibimos los datos en el formato correcto.

Code Title here. Show some code!

```
# Instalación
$ npm i --save class-validator class-transformer

# Activación
async function bootstrap() {
  const app = await NestFactory.create(AppModule);
  app.useGlobalPipes(new ValidationPipe());
  await app.listen(3000);
}
bootstrap();

# Uso
◊import { IsEmail, IsNotEmpty } from 'class-validator';

export class CreateUserDto {
  @IsEmail()
  email: string;

  @IsNotEmpty()
  password: string;
}
```

Providers

Los providers son objetos que son capaces de ser **inyectados como una dependencia**. El motor de ejecución de NestJS es el encargado de “cablear” las relaciones entre los objetos.

Es un concepto fundamental en NestJS que es aplicado a Servicios, Factorías, Helpers, …

Para marcar una clase como Provider debe ser anotada con el decorador **@Injectable()** y estar presente en la lista de providers del módulo

Code Title here. Show some code!

```
import { Injectable } from '@nestjs/common';

@Injectable()
export class AppService {
 getHello(): string {
   return 'Hello World!';
 }
}
…

@Module({
 imports: [],
 controllers: [AppController],
 providers: [AppService],
 exports: []
})
export class AppModule {}
```

Providers

Los providers son objetos que son capaces de ser **inyectados como una dependencia**. El motor de ejecución de NestJS es el encargado de “cablear” las relaciones entre los objetos.

Es un concepto fundamental en NestJS que es aplicado a Servicios, Factorías, Helpers, …

Para marcar una clase como Provider debe ser anotada con el decorador **@Injectable()** y estar presente en la lista de providers del módulo

Code Title here. Show some code!

```
import { Injectable } from '@nestjs/common';

@Injectable()
export class AppService {
 getHello(): string {
   return 'Hello World!';
 }
}
…

@Module({
 imports: [],
 controllers: [AppController],
 providers: [AppService],
 exports: []
})
export class AppModule {}
```

Middlewares, Guards, FilterExceptions e Interceptors

Podemos implementar diferentes interfaces que nos van a permitir capturar una petición HTTP en medio de su ciclo de ejecución con diferentes fines.

Los Middleware e Interceptors son más generales. Los ExceptionFilters nos permiten capturar excepciones que se producen durante la petición para retornar la respuesta adecuada y los Guards nos permiten decidir si la petición se puede realizar o no.

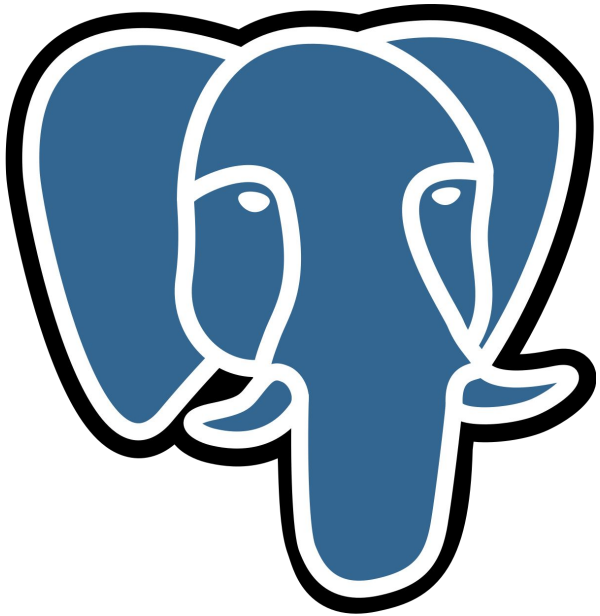
Code Title here. Show some code!

```
import { Injectable, NestMiddleware } from
'@nestjs/common';
import { Request, Response, NextFunction } from
'express';

@Injectable()
export class LoggerMiddleware implements
NestMiddleware {
 use(req: Request, res: Response, next:
NextFunction) {
   console.log('Request...');
   next();
 }
}
```

Las bases de datos son herramientas capaces de almacenar y manejar grandes cantidades de datos.

Existen diversos tipos de bases de datos, siendo las dos grandes categorías las bases de datos relacionales y las no relacionales.

PostgreSQL, también llamada Postgres, es una base de datos relacional de código abierto que usaremos en este taller.



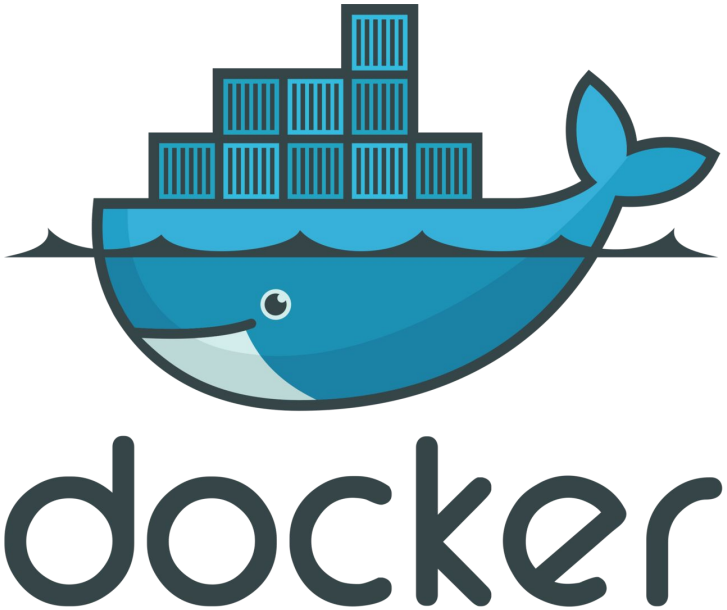
Docker es un software de código abierto utilizado para desplegar aplicaciones dentro de contenedores virtuales.

Añade una capa de abstracción sobre el ordenador para estandarizar el entorno de ejecución.

Un **contenedor** de Docker es un paquete de software con todas las dependencias necesarias para ejecutar una aplicación.

Una **imagen** de Docker es una receta que le dice a Docker cómo crear un contenedor específico.

En el Backend, Docker es una gran herramienta para preparar la infraestructura que necesita nuestra aplicación.



Configuraciones en NestJS

Normalmente, nuestras aplicaciones van a ejecutarse en diferentes entornos.

Según el entorno, hay parámetros que cambian y que conviene tener en una configuración externa al código que sea fácil de cambiar (como la configuración de nuestra Base de Datos).

NestJS cuenta con un paquete para manejar las configuraciones compatible con los ficheros .env usados normalmente en Node.js para manejar la configuración.

Code Title here. Show some code!

```
# Instalación
npm install @nestjs/config

# Configuración
@Module({
  imports: [ConfigModule.forRoot()],
})
export class AppModule {}
```

Un **ORM** es un software que nos permite mapear las estructuras en forma de tablas de nuestra Base de Datos, a entidades dentro de nuestro código con el objetivo de acelerar el desarrollo.

TypeORM

TypeORM es un ORM que puede ejecutarse en Node.js y TypeScript y que está perfectamente integrado dentro del ecosistema de NestJS.

Code Title here. Show some code!

```
# Instalación
 npm install @nestjs/typeorm typeorm pg

# Configuración
@Module({
  imports: [
    TypeOrmModule.forRoot({
      type: 'postgres',
      host: process.env.DB_HOST,
      port: +process.env.DB_PORT,
      database: process.env.DB_NAME,
      username: process.env.DB_USERNAME,
      password: process.env.DB_PASSWORD,
      autoLoadEntities: true,
      synchronize: true
    })],
})
export class AppModule {}
```

Entidades de TypeORM

Las entidades de TypeORM son clases anotadas con el decorador **@Entity()** con la información necesaria para mapear las propiedades del objeto a la tabla de Base de Datos.

Para dar esta información usaremos el decorador @Column con la información sobre la columna en base de datos.

Code Title here. Show some code!

```
@Entity()
export class Book {
    @PrimaryGeneratedColumn()
    id: number;
    @Column('text')
    title: string;
    @Column('text')
    author: string;
    @Column('numeric')
    pages: number;
    @ManyToOne(() => User, {
        eager: true
    })
    user: User;
}
```

Relaciones

TypeORM también te permite modelar las relaciones entre entidades usando los decoradores @OneToMany, @ManyToOne, @ManyToMany y @OneToOne

Code Title here. Show some code!

```
@Entity()
export class Book {
    @ManyToOne(() => Author, {
        eager: true
    })
    author: Author;
}

@Entity()
export class Author {
    @OneToMany(() => () => Book, book =>
book.author)
    books: Book[];
}
```

Comandos para todo

Una vez que ya sabemos la teoría y los bloques básicos podemos pasar a trabajar con la herramienta de línea de comandos de NestJS, que nos permitirá crear todos los tipos de objetos que hemos visto durante el curso.

De hecho, con un comando, podremos crear todo a la vez:

nest generate resource resourceName

```
Usage: nest generate|g [options] <schematic> [name] [path]

Generate a Nest element.
  Schematics available on @nestjs/schematics collection:

    ┌────────────────┬──────────────┬──────────────────────────────────────────┐
    │ name           │ alias        │ description                              │
    │ application    │ application  │ Generate a new application workspace     │
    │ class          │ cl           │ Generate a new class                     │
    │ configuration  │ config       │ Generate a CLI configuration file        │
    │ controller     │ co           │ Generate a controller declaration        │
    │ decorator      │ d            │ Generate a custom decorator              │
    │ filter         │ f            │ Generate a filter declaration            │
    │ gateway        │ ga           │ Generate a gateway declaration           │
    │ guard          │ gu           │ Generate a guard declaration             │
    │ interceptor    │ itc          │ Generate an interceptor declaration      │
    │ interface      │ itf          │ Generate an interface                    │
    │ middleware     │ mi           │ Generate a middleware declaration        │
    │ module         │ mo           │ Generate a module declaration            │
    │ pipe           │ pi           │ Generate a pipe declaration              │
    │ provider       │ pr           │ Generate a provider declaration          │
    │ resolver       │ r            │ Generate a GraphQL resolver declaration  │
    │ service        │ s            │ Generate a service declaration           │
    │ library        │ lib          │ Generate a new library within a monorepo │
    │ sub-app        │ app          │ Generate a new application within a monorepo │
    │ resource       │ res          │ Generate a new CRUD resource             │
    └────────────────┴──────────────┴──────────────────────────────────────────┘


Options:
  -d, --dry-run                    Report actions that would be taken without writing out results.
  -p, --project [project]          Project in which to generate files.
  --flat                           Enforce flat structure of generated element.
  --no-flat                        Enforce that directories are generated.
  --spec                           Enforce spec files generation. (default: true)
  --skip-import                    Skip importing (default: false)
  --no-spec                        Disable spec files generation.
  -c, --collection [collectionName]  Schematics collection to use.
  -h, --help                       Output usage information.
```

Para profundizar:

- **Programación Orientada a Objetos**: clases, objetos, herencia, polimorfismo, cohesión, acoplamiento, abstracción, encapsulamiento
- Principios **SOLID** y **DRY**
- **Patrones de diseño:** Decoradores, singletons, factorías
- **Inyección de dependencias**
- **APIs REST**
- **Programación orientada a aspectos (AOP)**
- **Testing: unitario y E2E**
- **Documentación** TypeScript
- **Documentación** NestJS
- **Bases de datos relacionales, SQL,** PostgreSQL
- Docker **y** Docker Compose

https://roadmap.sh/backend

Esto es todo por hoy y por las Winter Sessions 2023

Gracias!

@JavierDeArcosTL