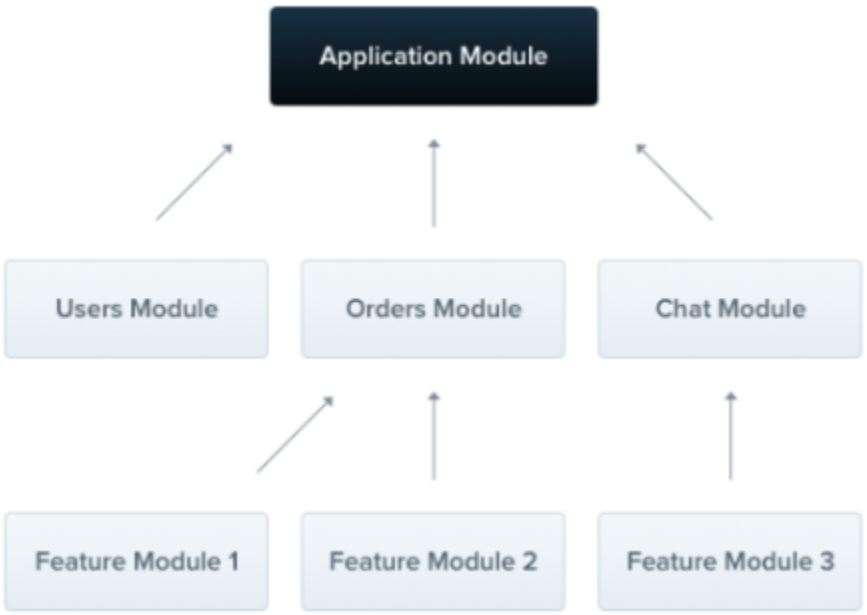
NEST js

Nest.js es un framework de desarrollo de aplicaciones backend basado en Node.js. Utiliza TypeScript para construir una estructura ampliamente escalable. Está inspirado en Angular y Spring y se centra en la creación de aplicaciones backend escalables y modulares.

Nest.js Esta compuesto de los siguientes elementos:

Elementos de Nestjs

Módulos (Module)



En Nestjs los módulos se utilizan para agrupar funcionalidades es la pieza central de nestjs. Un módulo Nestjs está compuesto controladores y servicios que trabajan en conjunto para realizar una funcionalidad especifica de la app. Cada aplicación Nestjs está compuesto de al menos un módulo (módulo raíz). Un módulo es una clase anotada con el decarador `@Module()`

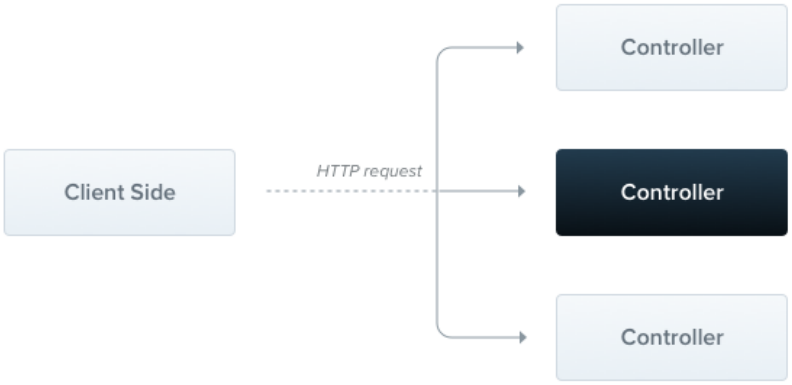
```
import { Module } from '@nestjs/common';

@Module({

  imports: [],
  controllers: [],
  providers: [],
})
export class AppModule {
```

```
}
```

Controladores (Controller)



Es el componente de nest.js que permite gestionar las peticiones HTTP y devolver respuestas. Los controladores en NestJS se definen utilizando el decorador `@Controller()`. Ejemplo

```
import { Controller, Get, Post, Body } from '@nestjs/common';
import { UsersService } from './users.service';

@Controller('users')
export class UsersController {
  constructor(private readonly usersService: UsersService) {}

  @Get()
  findAll() {
    return this.usersService.findAll();
  }

  @Post()
  create(@Body() user: any) {
    this.usersService.create(user);
  }
}
```

Proveedores

Un proveedor es un conjunto de componentes que se utilizan para encapsular la lógica del negocio de la aplicación y todas las operaciones para interactuar con datos (recuperar, grabar, y actualizar) o integrar con APIs externas.

Un proveedor en nestjs puede referirse a un servicio, repository, factory, helpers, etc.

Los Proveedores en Nestjs son clases decoradas con `@Injectable()`, lo que permite que sean inyectadas en controladores u otros servicios. A continuación se muesta un ejemplo básico de cómo se define y se utiliza un proveedor (servicio).

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

@Injectable()
export class UsersService {
  private users = [];

  findAll(): any[] {
    return this.users;
  }

  create(user: any) {
    this.users.push(user);
  }
}
```

Midleware

Un middleware es una función que se ejecuta antes de que la ruta correspondiente procese la solicitud. Se pueden utilizar para tareas como autenticación, registro, etc.

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

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

Guardias (Guard)

Los guardias son responsables de la autorización. Determinan si una solicitud dada debe ser manejada por el controlador de ruta correspondiente o no. Implementan la interfaz `CanActivate`. Cada guardia debe implementar el método `canActivate`

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

@Injectable()
export class AuthGuard implements CanActivate {
  canActivate(context: ExecutionContext): boolean {
    const request = context.switchToHttp().getRequest();
    return validateRequest(request);
```

```
  }
}

function validateRequest(request: any): boolean {
  // lógica de validación
  return true;
}
```

Interceptores

Los interceptores permiten modificar las solicitudes o respuestas. Implementan la interfaz `NestInterceptor` y el método `intercept()` y estan anotados con el decorador `@Injectable`.

```
import { Injectable, NestInterceptor, ExecutionContext, CallHandler } from
'@nestjs/common';
import { Observable } from 'rxjs';
import { map } from 'rxjs/operators';

@Injectable()
export class TransformInterceptor implements NestInterceptor {
  intercept(context: ExecutionContext, next: CallHandler): Observable<any> {
    return next.handle().pipe(map(data => ({ data })));
  }
}
```

Instalaciones necesarias

Para trabajar con nest debemos instalar nest cli de forma global:

```
npm i -g @nestjs/cli
```

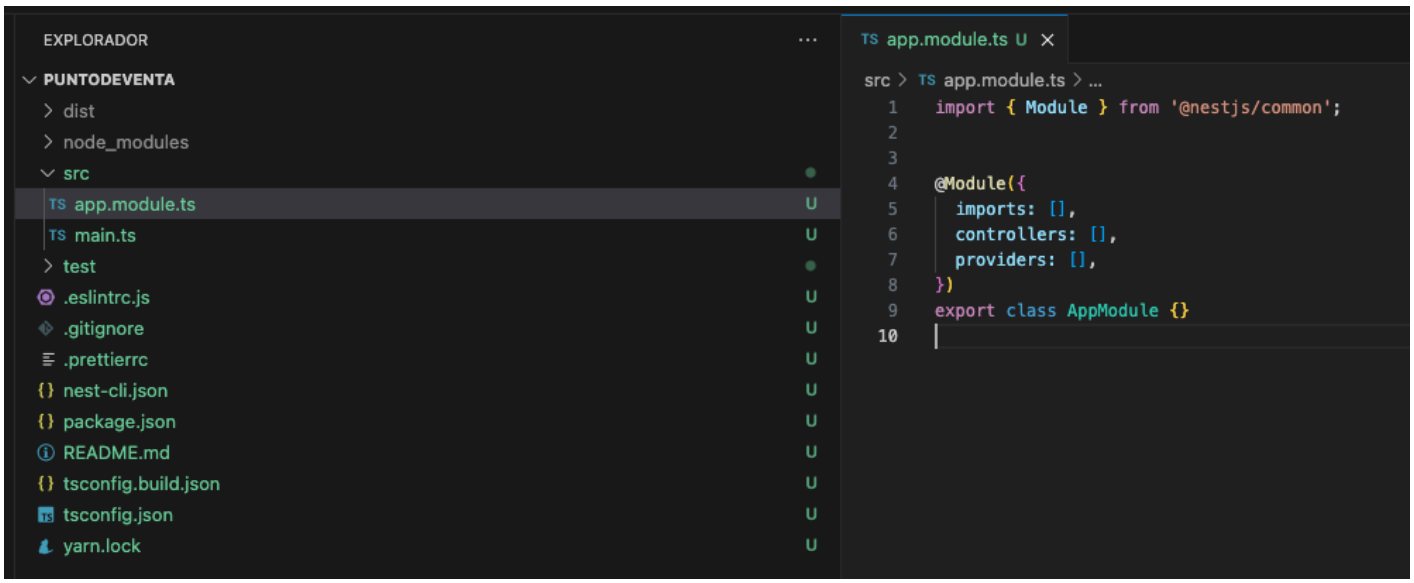
Creación de nuestro Primer proyecto

Una vez instalado nesjs de forma global tenemos disponible el comando `nest`. Para crer un proyecto ejecutamos el comando `nest new nombredemiproyecto`. Seleccionando `yarn` como manejador de paquetest (yarn es un manejador de paquetes que se instala con el comando `npm install –global yarn`. Si no lo tiene instalado debe instalar.

```
$ nest new puntodeventa
...
? Which package manager would you ❤  to use?
  npm
❯ yarn
```

```
pnpm
```

El comando `nest new puntodeventa` creará el directorio de nuestro proyecto `puntodeventa`, con todo el código y configuraciones necesarias del framework.



Ejecución proyecto nest

La primera vez debemos instalar las dependencias necesarias

```
$ cd puntodeventa/
```

```
puntodeventa$ yarn install
...
[1/4]   Resolving packages...
success Already up-to-date.
  Done in 0.15s.
```

Para ejecutar nuestro proyecto debemos ejecutar el comando `yarn run` dependiendo del entorno en que nos encontremos:

```
# development
puntodeventa$ yarn run start

# watch mode
puntodeventa$ yarn run start:dev

# production mode
puntodeventa$ yarn run start:prod
```

Para verificar si nuestro proyecto está en ejecución debemos ejecutar una petición GET desde postman o el navegador a http://localhost:3000. Si recibimos una respuesta con el código `200 OK` significa que nuestro proyecto se está ejecutando.

Creación del primer módulo

Para crear un módulo de nest se debe ejecutar el comando de `nest-cli nest g mo nombredelmodulo`. Por ejemplo si deseamos crear un módulo relacionado para la gestión de productos debemos ejecutar

```
puntodeventa$ nest g mo productos
```

El anterior comando crear el directorio y los archivos del módulo `src/productos/productos.module.ts`

Para listar toda la lista de comandos debemos ejecutar `nest -h`

```
puntodeventa$ nest -h
...
```

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
library	lib	Generate a new library within a monorepo
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
resource	res	Generate a new CRUD resource
service	s	Generate a service declaration
sub-app	app	Generate a new application within a monorepo

Creación del primer controlador

Para crear un controlador en nest debemos ejecutar el comando `nest g co nombredelcontrolador` desde la raiz de nuestro proyecxto. Por ejemplo para crear un controlador que permita gestionar las peticiones del módulo de productos debemos ejecutar:

```
$ nest g co productos
...
CREATE productos/productos.controller.spec.ts (513 bytes)
CREATE productos/productos.controller.ts (107 bytes)
UPDATE productos/productos.module.ts (186 bytes)
```

Si listamos el contenido de la carpeta productos podremos visualizar que se creó el archivo de definición del controlador `productos.controller.ts`, además se actualizará el archivo `productos.module.ts`, que contiene la definición del módulo de productos:

```
miproyecto/src$ ls productos
productos.controller.spec.ts productos.controller.ts      productos.module.ts
```

Añadimos la funcionalidad inicial para listar productos a partir de un array dentro del archivo `productos.controller.ts`

```
import { Controller, Get, Param } from '@nestjs/common';

@Controller('productos')
export class ProductosController {
    private productos = ['Xiaomi 14 Ultra','Xiaomi 14','Xiaomi 13T pro','Xiaomi
13T','Xiaomi 12T pro','Xiaomi 12'];

    @Get()
    getAllProductos(){
        return this.productos;
    }

    @Get(":id")
    getProductosById(@Param("id") id:String){
        return this.productos[+id]
    }
}
```

Para poder hacer la prueba de nuestro controlador desde el navegador o postman la ruta http://localhost:3000/productos que nos listará los productos y http://localhost:3000/productos/1 nos listara un producto en específico.

Creación del nuestro primer Proveedor (servicio)

Vamos a mejorar la lógica de nuestro controlador anterior implementando una capa adicional mediante el uso de proveedores (servicio). Para ello ejecutaremos el comando de nest-cli `nest g s` que nos permitirá crear el servicio productos.

```
puntodeventa$ nest g s productos --no-spec
...
CREATE src/productos/productos.service.ts (93 bytes)
UPDATE src/app.module.ts (307 bytes)
```

Editamos el archivo `src/productos/productos.service.ts` agregando la lógica de negocio para retornas productos:

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

@Injectable()
export class ProductosService {
```

```
    private productos0 = [
        {id:1,
         nombre:'Xiaomi 14 Ultra',
         categoria:'Telefonos'
        },
        { id:2,
          nombre:'Xiaomi 14',
          categoria:'Telefonos'
        },
        { id:3,
            nombre:'Xiaomi 13T',
            categoria:'Telefonos'
        },
        { id:4,
            nombre:'Xiaomi 12T pro',
            categoria:'Telefonos'
        },
        { id:5,
            nombre:'Xiaomi 12',
            categoria:'Telefonos'
        }
    ];

    findAll(){
        return this.productos0;
    }

    findById(id:Number){
        const prod = this.productos0.find( p => p.id === id );
        if ( !prod ) throw new NotFoundException(`Producto con el id '${ id }' no
encontrado`);
        return prod;
    }

}
```

Modificamos el archivo `productos.controller.ts` para que recupere datos del proveedor:

```
import { Controller, Get, Param } from '@nestjs/common';
import { ProductosService } from './productos.service';

@Controller('productos')
export class ProductosController {
   //private productos = ['Xiaomi 14 Ultra','Xiaomi 14','Xiaomi 13T pro','Xiaomi
13T','Xiaomi 12T pro','Xiaomi 12'];
    constructor(
        private readonly productosService: ProductosService
    ) {}

    @Get()
```

```
    getAllProductos(){
        // return this.productos;
        return this.productosService.findAll();
    }

    @Get(":id")
    getProductosById(@Param("id") id:String){
        // return this.productos[+id];
        return this.productosService.findById(+id);
    }
}
```

Creación de un recurso completo

La herramienta Nest-cli permite entre otras cosas generar un recurso completo es decir el conjunto de archivos que implementa un crud (Module, Controllers, Providers, DTOs, Entityes, etc) . El comando se llama `nest g resource <nombredelrecurso>`. Por ejemplo si a nuestro proyecto anterior queremos agregar un recurso clientes tendriamos que ejecutar el comando:

```
puntodeventa$ nest g res clientes
...
? What transport layer do you use? REST API
? Would you like to generate CRUD entry points? Yes
CREATE src/clientes/clientes.controller.spec.ts (596 bytes)
CREATE src/clientes/clientes.controller.ts (957 bytes)
CREATE src/clientes/clientes.module.ts (269 bytes)
CREATE src/clientes/clientes.service.spec.ts (474 bytes)
CREATE src/clientes/clientes.service.ts (651 bytes)
CREATE src/clientes/dto/create-cliente.dto.ts (33 bytes)
CREATE src/clientes/dto/update-cliente.dto.ts (181 bytes)
CREATE src/clientes/entities/cliente.entity.ts (24 bytes)
UPDATE package.json (1984 bytes)
UPDATE src/app.module.ts (382 bytes)
✔ Packages installed successfully.
```

El código de este repositorio lo puede descargar desde los siguientes enlaces:

- https://github.com/F1852D160/06-nestapp-puntodeventa
- https://github.com/F1852D160/07-nestappproviders-puntodeventa

Guía de Laboratorio

Objetivo

Objetivo El objetivo de este laboratorio es aprender a implementar un CRUD de productos(Crear, Leer, Actualizar, Eliminar) en una aplicación NestJS utilizando controladores y proveedores. La implementación se realizará en memoria utilizando un array para almacenar los datos.

Requisitos Previos

- Node.js y npm instalados.
- Conocimientos básicos de TypeScript.
- Familiaridad con NestJS.

El entregable de este laboratorio son las capturas de pantalla del procedimiento y el link del repositorio github de su proyecto.