

## Computación en Internet III

### Taller backend - Nestjs

#### Integrantes:

- Diana Lorena Balanta Solano
- Carlos Javier Bolaños Riascos
- Danna Alexandra Espinosa Arenas

**Nombre del proyecto:** UniLunch.

#### Objetivo del proyecto:

Facilitar el proceso de venta de almuerzos a estudiantes y restaurantes de la universidad.

#### Descripción:

En el presente informe se busca explicar el proceso de desarrollo del proyecto UniLunch. Esta propuesta busca ayudar a facilitar el proceso de venta de almuerzos/productos en los restaurantes de la universidad a los estudiantes. Para lograr esto, se debe hacer el desarrollo de una **API RESTFUL** con módulos y peticiones necesarias para cumplir con las necesidades identificadas. Los módulos identificados en el proyecto son: Gestión de reportes, gestión de usuarios, gestión de novedades, gestión de ventas y gestión de productos. Cabe destacar que la aplicación en su primera versión está destinada para los estudiantes y restaurantes de la Universidad Icesi.

#### Herramientas:

Las herramientas utilizadas en el desarrollo de este proyecto son:

- Node.
- NestJS.
- TypeScript.
- Git y GitHub.
- Visual Studio Code.
- Docker.
- MySQL.

#### Funcionalidades:

El sistema debe permitir

1. El acceso de usuarios con rol de estudiante y restaurante.
2. Registrar productos al restaurante.
3. Registrar la venta de productos del restaurante a estudiantes.
4. Registrar el pago de los estudiantes al restaurante.

5. Generar reportes de ventas del restaurante.
6. Publicar novedades/menú del día a los restaurantes.
7. Consultar las novedades/menú de los restaurantes a los estudiantes.

#### Proceso:

- **Configuración inicial del proyecto:** Para iniciar con la implementación, se usaron los comandos:

```
npm i -g @nestjs/cli
nest new uni-lunch
npm install @nestjs/typeorm typeorm mysql
```

- **Creación de los módulos:** Se hizo la configuración de la cada uno de los módulos descritos. Se debe tener en cuenta que algunos pueden ser “module” o “resource”. Los comandos para esto son:

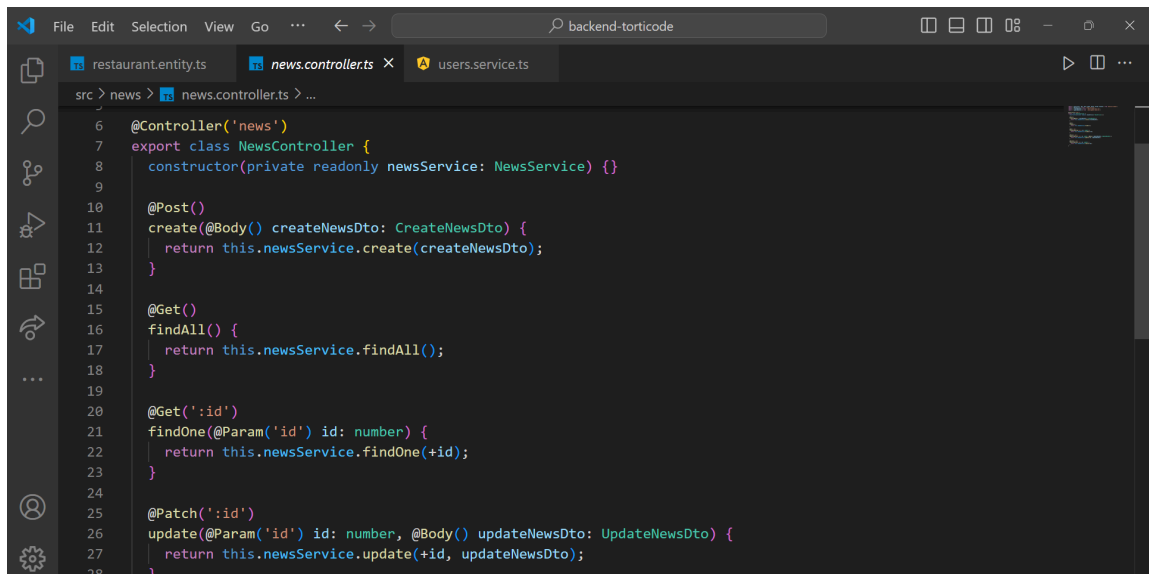
```
nest g mo user
nest g resource user
```

- **Creación de las entidades y DTO’S:** En cada uno de los recursos y módulos es necesario definir los atributos de las entidades, además de sus DTO’s para poder crear y actualizar los objetos según las necesidades del usuario. Ya con las entidades claramente definidas, se pasa a realizar las relaciones necesarias ajustadas a la lógica del negocio.
- **Implementación de los CRUD’S:** Con el fin de poder gestionar las peticiones de los usuarios y consultas a las bases de datos, se hace la capa Controller y Service con cada uno de los módulos, siguiendo así los principios de la arquitectura web.
- **Implementación de Autenticación y Seguridad:** Al hablar de gestión de usuarios, debemos garantizar que solo los usuarios registrados en el sistema, con datos correctos y un rol definido, puedan ingresar sin mayor problema, completando así sus funciones o necesidades en la aplicación. Para esto, se usó jwtoken y bcrypt para dar seguridad a la aplicación.
- **Pruebas:** Al completar los CRUD’S y la autenticación pasamos a hacer el apartado de las pruebas. Se usaron mock para poder simular la consulta en el repositorio de los diferentes módulos. Para esto, se deben definir los métodos CRUD en el mock, además de dar la ruta para obtener la respuesta HTTP de la simulación (mock).

#### Endpoints:

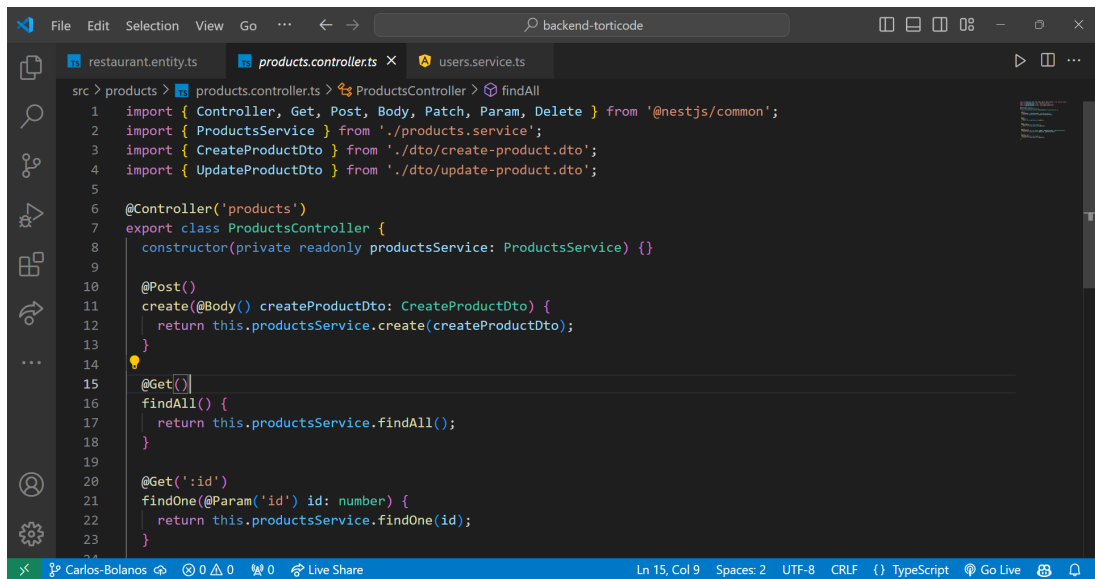
Las rutas definidas en los controller del proyecto siguen los estándares definidos al inicio del curso con el artículo Restful Routes, What are they?, 2017. Los endpoints definidos son:

- Endpoints en módulo “news”.



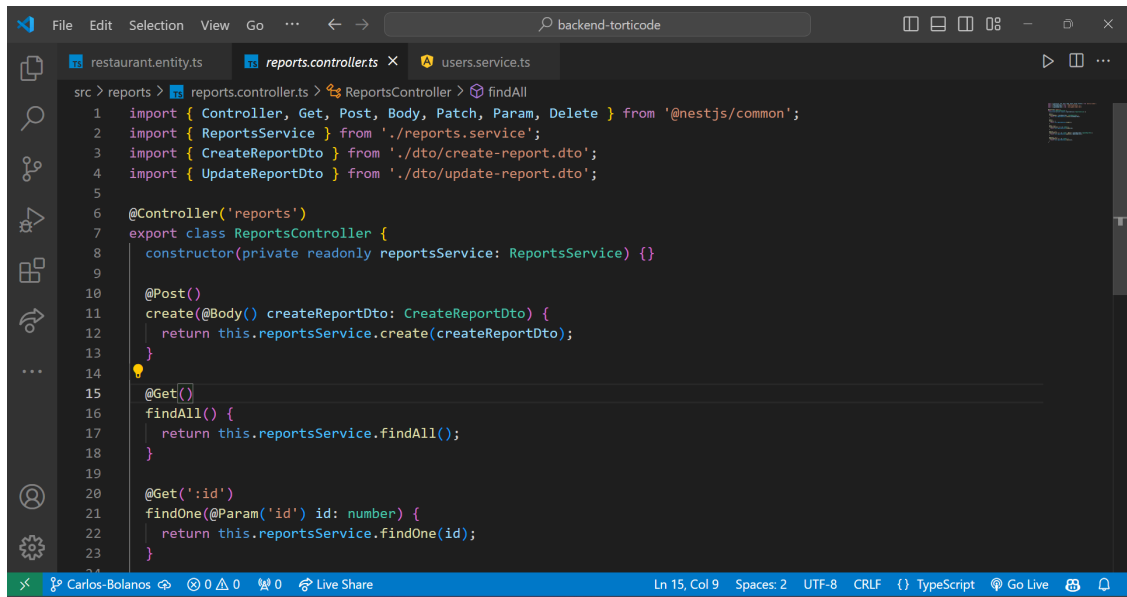
```
src > news > news.controller.ts > ...
6  @Controller('news')
7  export class NewsController {
8    constructor(private readonly newsService: NewsService) {}
9
10
11  @Post()
12  create(@Body() createNewsDto: CreateNewsDto) {
13    return this.newsService.create(createNewsDto);
14  }
15
16  @Get()
17  findAll() {
18    return this.newsService.findAll();
19  }
20
21  @Get('/:id')
22  findOne(@Param('id') id: number) {
23    return this.newsService.findOne(+id);
24  }
25
26  @Patch('/:id')
27  update(@Param('id') id: number, @Body() updateNewsDto: UpdateNewsDto) {
28    return this.newsService.update(+id, updateNewsDto);
29  }
30}
```

- Endpoints en módulo “products”.



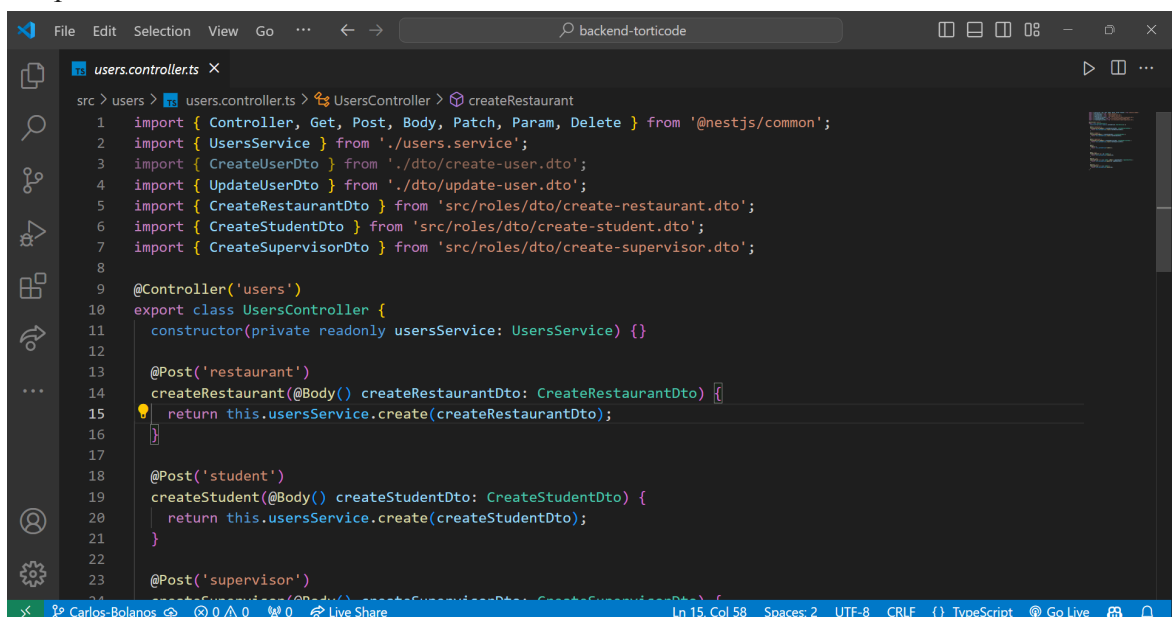
```
src > products > products.controller.ts > ProductsController > findAll
1  import { Controller, Get, Post, Body, Patch, Param, Delete } from '@nestjs/common';
2  import { ProductsService } from './products.service';
3  import { CreateProductDto } from './dto/create-product.dto';
4  import { UpdateProductDto } from './dto/update-product.dto';
5
6  @Controller('products')
7  export class ProductsController {
8    constructor(private readonly productsService: ProductsService) {}
9
10
11  @Post()
12  create(@Body() createProductDto: CreateProductDto) {
13    return this.productsService.create(createProductDto);
14  }
15
16  @Get()
17  findAll() {
18    return this.productsService.findAll();
19  }
20
21  @Get('/:id')
22  findOne(@Param('id') id: number) {
23    return this.productsService.findOne(id);
24  }
25}
```

- Endpoints en módulo “reports”.



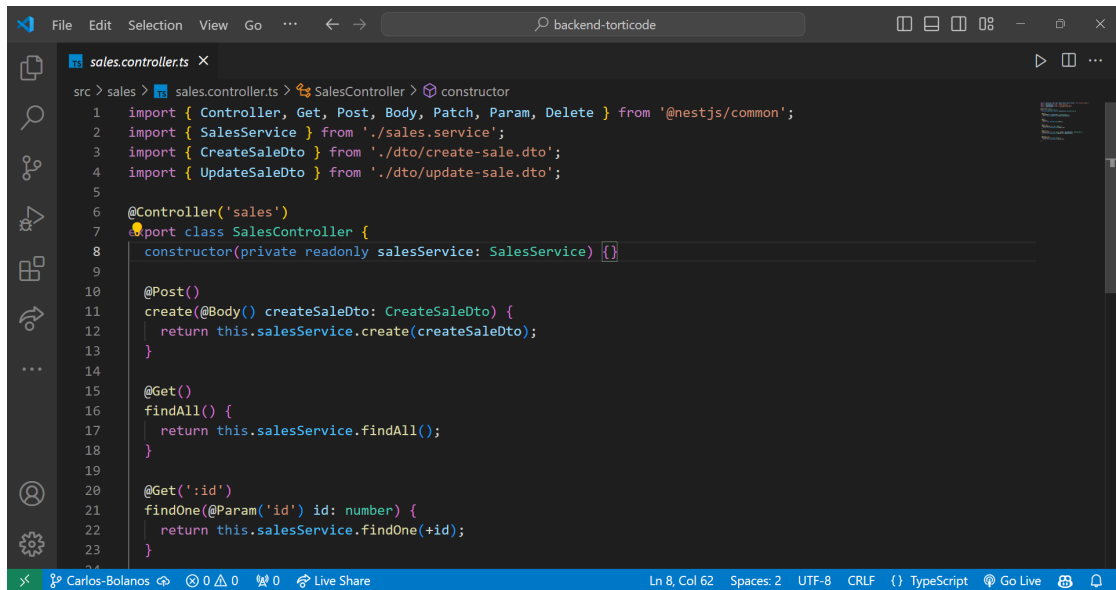
```
src > reports > reports.controller.ts > ReportsController > findAll
1 import { Controller, Get, Post, Body, Patch, Param, Delete } from '@nestjs/common';
2 import { ReportsService } from './reports.service';
3 import { CreateReportDto } from './dto/create-report.dto';
4 import { UpdateReportDto } from './dto/update-report.dto';
5
6 @Controller('reports')
7 export class ReportsController {
8   constructor(private readonly reportsService: ReportsService) {}
9
10  @Post()
11  create(@Body() createReportDto: CreateReportDto) {
12    return this.reportsService.create(createReportDto);
13  }
14
15  @Get()
16  findAll() {
17    return this.reportsService.findAll();
18  }
19
20  @Get(':id')
21  findOne(@Param('id') id: string) {
22    return this.reportsService.findOne(id);
23  }
24 }
```

- Endpoints en módulo “users”.



```
src > users > users.controller.ts > UsersController > createRestaurant
1 import { Controller, Get, Post, Body, Patch, Param, Delete } from '@nestjs/common';
2 import { UsersService } from './users.service';
3 import { CreateUserDto } from './dto/create-user.dto';
4 import { UpdateUserDto } from './dto/update-user.dto';
5 import { CreateRestaurantDto } from 'src/roles/dto/create-restaurant.dto';
6 import { CreateStudentDto } from 'src/roles/dto/create-student.dto';
7 import { CreateSupervisorDto } from 'src/roles/dto/create-supervisor.dto';
8
9 @Controller('users')
10 export class UsersController {
11   constructor(private readonly userService: UsersService) {}
12
13   @Post('restaurant')
14   createRestaurant(@Body() createRestaurantDto: CreateRestaurantDto) {
15     return this.userService.create(createRestaurantDto);
16   }
17
18   @Post('student')
19   createStudent(@Body() createStudentDto: CreateStudentDto) {
20     return this.userService.create(createStudentDto);
21   }
22
23   @Post('supervisor')
24   createSupervisor(@Body() createSupervisorDto: CreateSupervisorDto) {
25     return this.userService.create(createSupervisorDto);
26   }
27 }
```

- Endpoints en módulo “sales”.



```
src > sales > sales.controller.ts > SalesController > constructor
1 import { Controller, Get, Post, Body, Patch, Param, Delete } from '@nestjs/common';
2 import { SalesService } from '../sales.service';
3 import { CreateSaleDto } from '../dto/create-sale.dto';
4 import { UpdateSaleDto } from '../dto/update-sale.dto';
5
6 @Controller('sales')
7 export class SalesController {
8   constructor(private readonly salesService: SalesService) {}
9
10  @Post()
11  create(@Body() createSaleDto: CreateSaleDto) {
12    return this.salesService.create(createSaleDto);
13  }
14
15  @Get()
16  findAll() {
17    return this.salesService.findAll();
18  }
19
20  @Get('/:id')
21  findOne(@Param('id') id: string) {
22    return this.salesService.findOne(+id);
23  }
24 }
```

### Complicaciones:

- Como grupo creamos que al plantear nuestros mismos proyectos podemos llegar a dos límites. Uno es una versión compleja y otra muy básica.
- Las bases de datos relacionales combinadas con el framework nestjs pueden complicarse un poco, las relaciones bilaterales pueden jugar algunos dolores de cabeza.
- El cargar datos desde un módulo seed permite controlar la inserción de datos iniciales a la aplicación, permitiendo así mejorar la interacción y las pruebas de las rutas y servicios en general.