

INTITUTO TECNOLOGICO DE PATRIMONIO Y TURISMO

“YAVIRAC”

NOMBRE: Tania Paola Acosta Calle

CURSO: 5to “A”

TEMA: DOCUMENTACIÓN DE SERVIDORES CONTROLADORES, MODELOS, ENTIDADES, DTO

El archivo service se va a encargar de comunicar con el backed



Lo que se realiza es un injectable el cual se inyecta una dependencia

```
@Injectable()
export class CurriculaServic {
  constructor(
    @InjectRepository (CurriculumEntity)
    private curriculumRepository: Repository<CurriculumEntity>,
  ) {}
```

Metodos:

Se crea lo que son los metodos y el nomnre de nuestra tabla en este caso tenemos el Create

Remove FindAll , FindOne y delete

Se realiza la comunicacon con el componente el cual se realiza mediante el contructor

Y se coloca frente de cada metodo que sea asincronico y espere una respuesta el cual se realiza con cada metodo respectivo.

```

    } {}

    async create(payload: CreateCurriculumDto) {
        const newCurriculum= this.curriculumRepository.create(payload);

        return await this.curriculumRepository.save(newCurriculum);
    }

    async remove(id: number) {
        return await this.curriculumRepository.softDelete(id);
    }

    async findAll() {
        return await this.curriculumRepository.find();
    }

    async findOne(id: number) {
        const curriculum = await this.curriculumRepository.findOne({
            where: {
                id: id,
            },
        });
    };

```

```

        if (curriculum === null) {
            throw new NotFoundException('El producto no se encontro');
        }

        return curriculum;
    }

    async update(id: number, payload: UpdateCurriculumDto) {
        const curriculum = await this.curriculumRepository.findOne({
            where: {
                id: id,
            },
        });

        if (curriculum === null) {
            throw new NotFoundException('El producto no se encontro');
        }

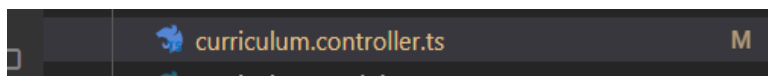
        this.curriculumRepository.merge(curriculum, payload);

        return this.curriculumRepository.save(curriculum);
    }
}

```

Controladores

Responde las solicitudes que se realiza al servidor



Para esto tenemos lo que son decoradores el cual se inicializa con un arroba cual agregan funcionalidades

Los nombres de las rutas en el controlador debe ir siempre en plural

Y tiene una clase el cual llamamos como **curriculaController** y tenemos un constructor

El cual nos sirve para dar un valor a la clase .

```
9
10
11 @Controller('curricula')
12 export class CurriculaController {
13   constructor(private curriculaService: CurriculaService) {}
14 }
```

El siguiente código tenemos un decorador que es un método el cual es el método GET

El decorador `@Httpcode()` me va a permitir modificar el estatus de la respuesta

En la siguiente línea tenemos un `@Query` el cual nos sirve para enviar un parámetro el cual se hace mediante con el signo de ?

tenemos una variable `const` que tiene un valor como `response` el cual hace una referencia con **this.CurriculaService** y el método **FINDALL()** el cual devuelve una lista de cadena.

```
1
2
3 @Get('')
4 @HttpCode(HttpStatus.OK)
5 findAll(@Query() params: any) {
6   const response = this.curriculaService.findAll();
7
8   return response;
9   // return {
10    //   data: response,
11    //   message: 'index',
12    // };
13 }
```

Tenemos lo mismo de la anterior línea pero si con la diferencia del método `@param` el cual envía parámetros con : y algo podemos enviar en nuestro cliente por ejemplo `{{host}}/curriculum/algo`.

-tenemos el método `FindOne` el cual nos sirve para buscar un objeto en específico

```
1
2
3 @Get('/:id')
4 @HttpCode(HttpStatus.OK)
5 findOne(@Param('id', ParseIntPipe) id: number) {
6   const response = this.curriculaService.findOne(id);
7
8   return response;
9   // {
10    //   data: response,
11    //   message: 'show',
12    // };
13 }
```

Tenemos el método `@Post` el cual nos sirve para actualizar `@Httpcode()` me va a permitir modificar el estatus de la respuesta en este caso tenemos como `create @Body` es el cuerpo y el **payload** nos va a servir como una variable y guardar toda la data

```
1
2
3 @Post('')
4 @HttpCode(HttpStatus.CREATED)
5 create(@Body() payload: CreateCurriculumDto) {
6   const response = this.curriculaService.create(payload);
7   return response;
8
9   // return {
10    //   data: response,
11    //   message: 'created',
12    // };
13 }
```

Tenemos el método @Put que es para actualizar y es casi similar al código anterior

```
@Put('/:id')
@statusCode(HttpStatus.CREATED)
update(
  @Param('id', ParseIntPipe) id: number,
  @Body() payload: UpdateCurriculumDto,
) {
  const response = this.curriculaService.update(id, payload);
  return response;
  // return {
  //   data: response,
  //   message: 'updated ${id}',
  // };
}
```

Tenemos el método @Delete el cual nos sirve para eliminar en este caso esta el ID para nosotros colocar mediante el método @param que id vamos a borrar.

```
@Delete('/:id')
@statusCode(HttpStatus.CREATED)
remove(@Param('id', ParseIntPipe) id: number) {
  const response = this.curriculaService.remove(id);
  return response;
  // return {
  //   data: response,
  //   message: 'deleted',
  // };
}
```

MODULO CURRICULUM

El import nos sirve para importar las funciones que han sido exportadas desde un moduli externo

```
import { Module } from '@nestjs/common';
import { TypeOrmModule } from '@nestjs/typeorm';
import { CurriculumEntity } from 'src/entities/curriculum.entity';
import { CurriculaController } from './curriculum.controller';
import { CurriculaService } from './curriculum.service';
```

@module que es un decorador

El cual tenemos lo que son clases de JavaScript el cual se nos crea ya por defecto

Y tenemos una clase con nombre CurricularModule

```
@Module({
  imports: [TypeOrmModule.forFeature([CurriculumEntity])],
  providers: [CurriculaService],
  controllers: [CurriculaController],
})
export class CurricularModule {}
```

ENTITIES

Las entidades nos sirven para mapear y poder conectar entre la base de datos y el backend en cual podemos crear actualizar borrar y consultar.

El cual tenemos un decorador con nombre de @entity y el nombre el plural y con una clase

En este caso tenemos un ID porque es incrementable y tiene de tipo NUMBER

Tenemos un decorador entonces estamos diciendo que es una @PrimaryGeneralColumn el cual convierte en dos cosas en una primary fk y que sea incrementable

```
@Entity('curricula')
export class CurriculumEntity {

  @PrimaryGeneratedColumn()
  id:number;
```

Tenemos que importar un decorador el cual es el @column y el nombre de nuestra tabla

Y colocar que tipo de dato es en nuestra base de datos en mi caso yo tengo como string y seguimos haciendo con los demás campos de la tabla

```
@Column('varchar', {
  name: 'code',
  length: 255,
  default: 'SN',
  comment: 'Nombre del producto',
})
code:string;
```

Aquí colocamos los decoradores con los nombres respectivos campos de la base de datos y colocando el tipo de dato

```
@CreateDateColumn()
name: 'created_at',
type: 'timestamp',
default: () => 'CURRENT_TIMESTAMP',
comment: 'Fecha de creación de la carrera'
})
createdAt: Date;

@CreateDateColumn()
name: 'started_at',
type: 'timestamp',
default: () => 'CURRENT_TIMESTAMP',
comment: 'Fecha de creación de la carrera'
})
startedAt: Date;

@Column('varchar', {
  name: 'name',
  length: 255,
  default: 'SN',
  comment: 'Nombre del producto',
})
name:string;

@Column('varchar', {
  name: 'description',
  length: 255,
  default: 'SN',
  comment: 'Nombre del producto',
})
description:string;

@Column('float', {
  name: 'price',
  comment: 'Precio del producto',
})
price:number;
```

DTO

Me van a servir para validar la data que viene en el cuerpo de la petición el cual se inicializa con un decorador y colocamos dependiendo los campos de nuestra tabla que estemos trabajando.

```

import {
  IsNumber,
  IsString,
  MinLength,
  IsDate,
  Min,
  Max,
  MaxLength,
  IsPositive,
  IsOptional,
} from 'class-validator';

export class CreateCurriculumDto {

  @IsNumber() //fk
  @IsPositive()
  readonly careerId:number;

  @IsNumber() //fk
  @IsPositive()
  readonly stateId:number;

  @IsString()
  @MinLength(2, {message: ' minimo 2 caracteres'})
  @MaxLength(20, {message: ' minimo 20 caracteres'})
  readonly code:string;

  @IsDate()
  @Type()
  @IsOptional()
  readonly startDate: Date;

  @IsDate()

```

Tenemos las 10 validaciones

El cual digo que sea de tipo String

Que sea mínimo de 2 y máximo de 100

Que sea sea opcional y que no sea alfanumérico Que sea de tipo type , Y que no sea una dirección

```

  @IsString()
  @MinLength(2, {message: ' minimo 2 caracteres'})
  @MaxLength(100, {message: ' minimo 100 caracteres'})
  @IsPositive()
  @Max(100)
  @Min(1)
  @IsOptional()
  @IsAlphanumeric()
  @Type()
  @IsEthereumAddress()

  readonly description:string;

  @IsNumber()

```