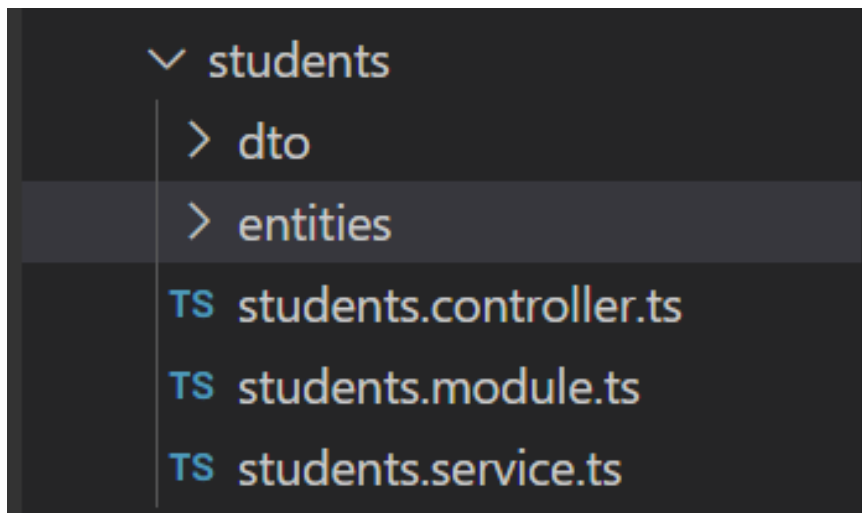


Carrera	Asignatura	Datos del estudiante	fecha
Tecnología en Desarrollo de Software	Tendencias actuales de programación	Apellidos: Motoche Roman	26/junio/2022
		Nombres: Kevin Joel	Docente: Ing. Hernan Mejia

## Documentación app NESTJS (backend)



### Students.service.ts

El archivo. service.ts nos sirve para poder conectarnos a la base de datos y este a su vez nos permitirá realizar un CRUD.

1.-Dependencias del archivo o librerías usadas para la factibilidad de interacción con el lenguaje

```
1 import { Injectable } from '@nestjs/common';
2 import { CreateStudentDto } from '../dto/create-student.dto';
3 import { UpdateStudentDto } from '../dto/update-student.dto';
4 import { InjectRepository } from '@nestjs/typeorm';
5 import { StudentEntity } from '../entities/student.entity';
6 import { Repository } from 'typeorm';
7 import { InformationStudentsService } from '../information_students/information-stud
```

Carrera	Asignatura	Datos del estudiante	fecha
Tecnología en Desarrollo de Software	Tendencias actuales de programación	Apellidos: Motoche Roman	26/junio/2022
		Nombres: Kevin Joel	Docente: Ing. Hernan Mejia

1.1.- Hacemos una inyección de dependencias que este a su vez con la ayuda de un constructor nos permite crearnos un repositorio aquí es donde uno la obtiene en la conexión de base de datos pero primero necesitaremos una entidad.

@Injectable = Decorador de nestjs. Proveedor asíncrono esperará hasta que Promise se resuelva

```
@Injectable()
export class StudentsService {
  constructor(
    @InjectRepository(StudentEntity)
    private studentRepository: Repository<StudentEntity>,
    private informationStudentsService: InformationStudentsService,
  ) {}
}
```

1.2.- Create : este método tiene un payload: CreateStudentDto de tipo Dto por que este Dto permite que cualquier ruta que use CreateStudentDto aplicará estas reglas de validación

```
async create(payload: CreateInformationStudentDto) {
  const newInformationStudent = this.informationStudentRepository.create(payload)
  return await this.informationStudentRepository.save(newInformationStudent);
}
```

Carrera	Asignatura	Datos del estudiante	fecha
Tecnología en Desarrollo de Software	Tendencias actuales de programación	Apellidos: Motoche Roman	26/junio/2022
		Nombres: Kevin Joel	Docente: Ing. Hernan Mejia

findOne: Método toma un argumento que representa un id parámetro de ruta extraído

```
async findOne(id: number) {
  const informationStudent = await this.informationStudentRepository.findOne({
    where: {
      id: id,
    },
  });

  if (informationStudent === null) {
    throw new NotFoundException('La informacion no se encontro');
  }

  return informationStudent;
}
```

Update: Permite modificar los campos pero este a su vez hace uso de un dto llamado UpdateStudentDto y aplicara igual las reglas de validación

```
async update(id: number, payload: UpdateInformationStudentDto) {
  const informationStudent = await this.informationStudentRepository.findOne({
    where: {
      id: id,
    },
  });

  if (informationStudent === null) {
    throw new NotFoundException('La informacion no se encontro');
  }

  this.informationStudentRepository.merge(informationStudent, payload);

  return this.informationStudentRepository.save(informationStudent);
}
```

Carrera	Asignatura	Datos del estudiante	fecha
Tecnología en Desarrollo de Software	Tendencias actuales de programación	Apellidos: Motoche Roman	26/junio/2022
		Nombres: Kevin Joel	Docente: Ing. Hernan Mejia

Delete: Permite borrar un dato que hemos creado

```
async delete(id: number) {  
  return await this.informationStudentRepository.softDelete(id);  
}
```

### .module.ts

El módulo simplemente organiza el código relevante para una función específica, manteniendo el código organizado y estableciendo límites claros

```
TS information-students.module.ts M X  
src > modules > information_students > TS information-students.module.ts > ...  
1  import {Module} from '@nestjs/common';  
2  import { TypeOrmModule } from '@nestjs/typeorm';  
3  import { InformationStudentEntity } from '../entities/information-student.entity';  
4  import { InformationStudentsController } from '../information-students.controller';  
5  import { InformationStudentsService } from '../information-students.service';  
6  
7  @Module({  
8    imports: [TypeOrmModule.forFeature([InformationStudentEntity])],  
9  
10     controllers:[InformationStudentsController],  
11     providers:[InformationStudentsService],  
12     exports:[InformationStudentsService, TypeOrmModule],  
13  })  
14  export class InformationStudentsModule{}  
15
```

Carrera	Asignatura	Datos del estudiante	fecha
Tecnología en Desarrollo de Software	Tendencias actuales de programación	Apellidos: Motoche Roman	26/junio/2022
		Nombres: Kevin Joel	Docente: Ing. Hernan Mejia

## .controller.ts

El controlador permite recibir solicitudes específicas para la aplicación.

### 1.1 Primero haremos un constructor que se relacione con el service

```
@Controller('information-students')
export class InformationStudentsController {
  constructor(private informationStudentsService: InformationStudentsService) {}
```

2.2 El @Get()decorador del método de solicitud HTTP antes del findAll()método le dice a Nest que cree un controlador para un punto final específico para las solicitudes HTTP. Este también ayuda para poder buscar solo un id , aquí también definiremos los @Query y @Params . Hacemos uso de los Pipes para validar la ID

```
@Get('')
@HttpCode(HttpStatus.OK)
findAll(@Query() params: any) {
  const response = this.informationStudentsService.findAll();

  return response;
  // return {
  //   data: response,
  //   message: `index`,
  // };
}

@Get('/:id')
@HttpCode(HttpStatus.OK)
findOne(@Param('id', ParseIntPipe) id: number) {
  const response = this.informationStudentsService.findOne(id);
  return response;
  // return {
  //   data: response,
  //   message: `show`,
  // };
}
```

Carrera	Asignatura	Datos del estudiante	fecha
Tecnología en Desarrollo de Software	Tendencias actuales de programación	Apellidos: Motoche Roman	26/junio/2022
		Nombres: kevin Joel	Docente: Ing. Hernan Mejia

El método post y put envían a través del body los campos ya validados donde el cliente enviara la data y este a su vez le permitirá un dato o actualizarlo

```
@Post('')
@HttpCode(HttpStatus.CREATED)
create(@Body() payload: CreateInformationStudentDto) {
  const response = this.informationstudentsService.create(payload);
  return response;
  // return {
  //   data: response,
  //   message: `created`,
  // };
}

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

Carrera	Asignatura	Datos del estudiante	fecha
Tecnología en Desarrollo de Software	Tendencias actuales de programación	Apellidos: Motoche Roman	26/junio/2022
		Nombres: Kevin Joel	Docente: Ing. Hernan Mejia

Delete aquí es donde se destruye un dato a través del id

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

Activar  
Ve a Conf

## Entities

Sirve para poder administrar de mejor manera los datos al enviar a la base de datos este a su vez puede definir una entidad y sus columnas directamente en el modelo, utilizando decoradores.

```
TS information-student.entity.ts X
src > modules > information_students > entities > TS information-student.entity.ts > InformationStudentEntity
9   PrimaryGeneratedColumn,
10  UpdateDateColumn,
11  } from 'typeorm';
12
13  @Entity('information_students')
14  export class InformationStudentEntity {
15    @PrimaryGeneratedColumn()
16    id: number;
17
18    @Column('varchar', {
19      name: 'address',
20      length: 1000,
21      comment: 'La dirección donde reside el estudiante',
22    })
23    address: string;
24
25    @Column('integer', {
26      name: 'community',
27      comment: 'Las horas realizadas por parte del estudiante en integración con la so
28    })
29    community: number;
30
31    @Column('varchar', {
32      name: 'contact_emergency_name',
33      length: 255,
34      comment: 'Nombre del contacto de emergencia para informar sobre el estudiante',
35    })
36    contactEmergencyName: string;
37
38    @Column('varchar', {
39      name: 'contact_emergency_kinship',
40      length: 255,
41      comment: 'Nombre del contacto de emergencia de parentescos para informar sobre e
42    })
```

Activar Windows  
Ve a Configuración para activar Windows.

Carrera	Asignatura	Datos del estudiante	fecha
Tecnología en Desarrollo de Software	Tendencias actuales de programación	Apellidos: Motoche Roman	26/junio/2022
		Nombres: Kevin Joel	Docente: Ing. Hernan Mejia

## DTO

Son las validaciones de datos que vienen del body, aquí hacemos uso de decoradores para dar las respectivas restricciones

```

TS create-information-student.dto.ts ×
src > modules > information_students > dtos > TS create-information-student.dto.ts > CreateInformationStudentDto

17 @IsString({ message: 'Debe ser un string' })
18 @MaxLength(1000, { message: 'Maximo 1000 caracteres' })
19 readonly address: string;
20
21 @IsNumber({}, { message: 'Debe ser tipo numero' })
22 @Min(0, { message: 'El número de dígitos mínimo es 0.' })
23 readonly community: number;
24
25 @IsNumber({}, { message: 'Debe ser tipo numero' })
26 @Min(0, { message: 'El número de dígito mínimo es 0.' })
27 @Max(100, { message: 'Maximo 100 dígito' })
28 readonly disabilityPercentage: number;
29
30 @IsNumber({}, { message: 'Debe ser tipo numero' })
31 @Min(0, { message: 'El número de dígito mínimo es 0.' })
32 readonly economicAmount: number;
33
34 @IsNumber({}, { message: 'Debe ser tipo numero' })
35 @Min(0, { message: 'El número de dígito mínimo es 0.' })
36 readonly educationalAmount: number;
37
38 @IsNumber({}, { message: 'Debe ser tipo numero' })
39 readonly familyIncome: number;
40
41 @IsString({ message: 'Debe ser un string' })
42 @MaxLength(1, { message: 'Maximo 1 caracter' })
43 readonly isLostGratuity: string;
44
45 @IsString({ message: 'Debe ser un string' })
46 @MaxLength(1, { message: 'Maximo 1 caracter' })
47 readonly isExecutedPractice: string;
48
49 @IsString({ message: 'Debe ser un string' })
50 @MaxLength(1, { message: 'Maximo 1 caracter' })

```

Activar Windows  
Ve a Configuración para activar Windows.

Ln 10, Col 1 Spaces: 2 UTF-8 CRLF {} TypeScript