

# Proyecto Final - Desarrollo Backend con NestJS

Profesor: Edgar Alejandro Mora

## Descripción General

El proyecto final tiene como objetivo que los estudiantes implementen un **backend completo utilizando el framework NestJS**, aplicando los principios de diseño modular, buenas prácticas de desarrollo y el uso de decoradores, servicios e inyección de dependencias. Cada grupo o estudiante deberá desarrollar una API funcional conectada a una base de datos relacional (la cual será proporcionada por el docente), implementando las siguientes capas: **entidades, DTOs (Data Transfer Objects), servicios, y la interacción con el frontend o parte gráfica**.

## Objetivos Específicos

- Comprender y aplicar el patrón **Modelo-Servicio-Controlador** en NestJS.
- Implementar entidades y relaciones usando decoradores de TypeORM.
- Aplicar correctamente la validación de datos mediante DTOs y los pipes de NestJS.
- Implementar lógica de negocio en los servicios.
- Incorporar la encriptación de contraseñas de manera segura.
- (Opcional) Implementar un sistema de autenticación basado en **JWT y estrategias de Passport**.
- (Opcional) Implementar el uso de **Guards** para restringir el acceso a determinados recursos.
- Diseñar una interfaz gráfica básica que consuma los endpoints desarrollados.

## Requisitos Técnicos

- El proyecto deberá estar desarrollado en **NestJS** (última versión estable).
- Uso de **TypeORM** para la comunicación con la base de datos.
- Uso de **class-validator** y **class-transformer** para la validación de DTOs.

- Las contraseñas deberán encriptarse con la librería **bcrypt**.
- El proyecto deberá incluir un **archivo README** con las instrucciones de instalación y uso.

## Criterios de Evaluación

Criterio	Porcentaje
Definición y correcta implementación de entidades y relaciones (TypeORM)	25 %
Creación de DTOs, validación y uso adecuado de Pipes	20 %
Implementación del servicio con la lógica de negocio correspondiente	20 %
Encriptación de contraseñas y buenas prácticas de seguridad	15 %
(Optativo) Autenticación de usuarios mediante JWT y Strategy	5 %
(Optativo) Implementación de Guards para control de acceso	5 %
Diseño de la interfaz gráfica y conexión con el backend	10 %
<b>Total</b>	<b>100 %</b>

## Ejemplo de Contexto y Base de Datos

### Contexto del proyecto

La institución educativa **EduTrack** requiere el desarrollo de un sistema para **gestionar la información de usuarios, profesores, estudiantes, cursos e inscripciones**. El sistema permitirá registrar a los usuarios, diferenciar si son profesores o estudiantes, asignar cursos a los profesores y permitir que los estudiantes se inscriban en dichos cursos. El propósito del sistema es servir como base para el control académico interno de la institución.

### Entidades y atributos

#### Usuario

- id
- nombre\_completo
- correo
- contraseña
- rol (*'profesor'* o *'estudiante'*)

### **Profesor (especialización de Usuario)**

- id
- especialidad

### **Estudiante (especialización de Usuario)**

- id
- año\_ingreso

### **Curso**

- id
- nombre
- descripción
- créditos
- profesor\_id

### **Inscripción**

- id
- fecha\_inscripción
- nota
- estudiante\_id
- curso\_id

## Relaciones entre entidades

**1. Usuario – Profesor / Estudiante** Cada registro en las entidades *Profesor* y *Estudiante* representa una especialización del *Usuario*, es decir, un profesor o un estudiante hereda los datos básicos del usuario. **Cardinalidad:** 1 a 1 (Un usuario solo puede ser o profesor o estudiante.)

**2. Profesor – Curso** Un profesor puede dictar varios cursos, pero cada curso solo puede ser dictado por un único profesor. **Cardinalidad:** 1 a N (Un profesor tiene muchos cursos.)

**3. Estudiante – Inscripción** Un estudiante puede tener varias inscripciones, una por cada curso en el que participe, pero cada inscripción pertenece a un único estudiante. **Cardinalidad:** 1 a N (Un estudiante tiene muchas inscripciones.)

**4. Curso – Inscripción** Un curso puede tener varias inscripciones de distintos estudiantes, pero cada inscripción está asociada a un solo curso. **Cardinalidad:** 1 a N (Un curso tiene muchas inscripciones.)

## Cronograma de Entregas

### Primera Entrega - 14 de noviembre

En esta primera entrega, los estudiantes deberán presentar el **esqueleto funcional del backend** implementado en NestJS. El objetivo es validar la correcta configuración del entorno de desarrollo y la estructura general del proyecto.

#### Requisitos mínimos:

- Proyecto NestJS correctamente configurado y funcional.
- Definición de las entidades y relaciones utilizando TypeORM.
- Implementación de los DTOs con validaciones básicas.
- Creación del servicio correspondiente a una de las entidades principales.
- Endpoint funcional para operaciones CRUD básicas (al menos una entidad completa).
- Archivo README.md con instrucciones para ejecutar el proyecto.

**Propósito de la entrega:** Validar la correcta estructura del proyecto, el uso de los principios del patrón *Modelo–Servicio–Controlador* y el dominio de las bases del framework NestJS.

## Entrega Final - 28 de noviembre

La entrega final consistirá en el proyecto completo y funcional, cumpliendo con todos los requisitos establecidos en este documento. Deberá incluir el sistema de encriptación de contraseñas, validaciones, servicios, relaciones entre entidades, y la parte gráfica que consuma los endpoints creados.

(De forma opcional, se evaluará positivamente la implementación de autenticación con JWT, estrategias de Passport y Guards de autorización.)

## Entregables

1. Repositorio en GitHub con el código fuente completo del backend y el frontend.
2. Archivo `README.md` con instrucciones para la instalación, configuración y ejecución del proyecto.
3. Archivo `.env.template` con las variables de entorno necesarias.
4. Capturas o video corto de funcionamiento del sistema.

## Parámetros de Evaluación

El proyecto deberá ejecutarse correctamente en el entorno definido por el docente. Se valorará la **claridad del código**, la **organización del proyecto** y la **implementación de buenas prácticas de desarrollo** en NestJS. Cualquier plagio o copia parcial de código sin referencia será motivo de anulación del trabajo.

## Observaciones Finales

Los estudiantes podrán implementar funcionalidades adicionales que enriquezcan el proyecto, tales como manejo de roles, paginación, envío de correos o integración con APIs externas, las cuales serán tenidas en cuenta como puntos extra en la calificación final.