INSTITUTO TECNOLÓGICO SUPERIOR DE CALKINÍ

INGENIERIA EN SISTEMAS COMPUTACIONALES

SEPTIMO SEMESTRE

AEB-1055 PROGRAMACIÓN WEB

DOCENTE: DRA. YAQUELINE PECH HUH

Manual Backend

INTEGRANTES DEL EQUIPO:

- CHAN CHIN CRISMAR DE JESUS MATRICULA: 7612.
- EK MAS DYLAN FERNANDO MATRICULA: 7618.
- KU CENTENO BERENISE ANTONIA MATRICULA: 7628.
- MAY VERGARA ROBERTO CARLOS MATRICULA: 7630.
- UC PERALTA ADIEL EDUARDO MATRICULA: 7666.

GRUPO: “A”

CICLO ESCOLAR: 2024 – 2025N

Contenido

Framework

- **Framework Utilizado**: NestJS
- **Versión**: 10.4.3
- **Requisitos Previos**:
  - **Node.js**: Versión recomendada 20.3.1 o superior.
  - **npm**: Versión compatible con Node.js.
  - **Cuenta de Firebase**: Necesaria para acceder a Firestore y el almacenamiento en la nube.
  - **Credenciales de Firebase**: Archivo de configuración JSON descargado desde Firebase Console.
- **Base de Datos (BD)**:
  - **Firestore**: Base de datos NoSQL de Firebase para almacenamiento y gestión de datos de usuarios y solicitudes de admisión.

Dependencias y versiones

Dependencias de Producción:

- @google-cloud/firestore: ^7.10.0
- @google-cloud/storage: ^7.13.0
- @nestjs/common: ^10.4.3
- @nestjs/config: ^3.2.3
- @nestjs/core: ^10.4.3
- @nestjs/platform-express: ^10.4.5
- bcrypt: ^5.1.1
- firebase: ^10.13.1
- firebase-admin: ^12.4.0
- jsonwebtoken: ^9.0.2
- multer: ^1.4.5-lts.1
- pdfkit: ^0.15.0
- puppeteer: ^23.5.3
- reflect-metadata: ^0.1.13
- rxjs: ^7.8.1

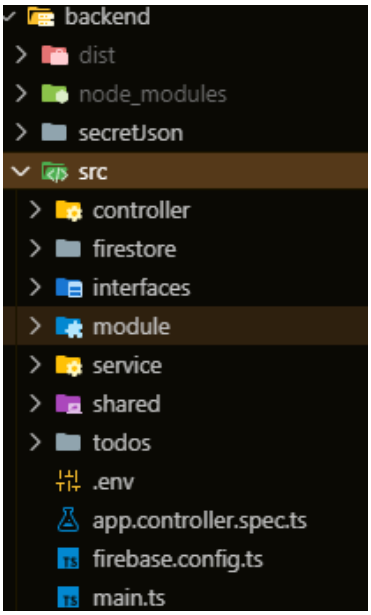
Dependencias de Desarrollo:

- @nestjs/cli: ^10.0.0
- @nestjs/schematics: ^10.0.0
- @nestjs/testing: ^10.0.0
- @types/express: ^4.17.17
- @types/firebase: ^3.2.1
- @types/jest: ^29.5.2
- @types/multer: ^1.4.12
- @types/node: ^20.3.1
- @types/supertest: ^2.0.12
- @typescript-eslint/eslint-plugin: ^6.0.0

- @typescript-eslint/parser: ^6.0.0
- eslint: ^8.42.0
- eslint-config-prettier: ^9.0.0
- eslint-plugin-prettier: ^5.0.0
- jest: ^29.5.0
- prettier: ^3.0.0
- source-map-support: ^0.5.21
- supertest: ^6.3.3
- ts-jest: ^29.1.0
- ts-loader: ^9.4.3
- ts-node: ^10.9.1
- tsconfig-paths: ^4.2.0
- typescript: ^5.1.3

Estructura de carpetas

La estructura de carpetas se centra en organizar el código de manera clara, agrupando componentes similares en diferentes directorios dentro de src.



controller

Propósito: Esta carpeta contiene los controladores que gestionan las solicitudes HTTP y llaman a los servicios para realizar operacionesde negocio.

Componentes principales:

- Cada archivo controlador (por ejemplo, student.controller.ts) define endpoints específicos que manejan operaciones como GET, POST, PUT, y DELETE.

Firestore

Propósito: Almacena configuraciones y proveedores para conectar con Firestore.

Componentes principales:

- firestore.providers.ts: Define y exporta proveedores de Firestore que los servicios utilizan para interactuar con la base de datos.

Interface

- **Propósito**: Define interfaces TypeScript que describen la estructura de datos manejados por el backend.
- **Componentes principales**:
    - Interfaces como StudentInterface que aseguran que las propiedades de los objetos sean consistentes a lo largo del código.

Module

Propósito: Organiza y agrupa controladores y servicios relacionados dentro de módulos de NestJS.

Componentes principales:

- app.module.ts: Módulo raíz del proyecto, donde se registran todos los módulos, controladores y servicios.

Service

Propósito: Contiene la lógica de negocio y operaciones CRUD para cada entidad.

Componentes principales:

- Cada archivo de servicio (por ejemplo, student.service.ts) se encarga de las operaciones específicas de una colección en Firestore.

Shared

Propósito: Almacena funciones y componentes reutilizables entre varias partes del proyecto, como validaciones y utilidades.

Componentes principales:

- Cada archivo de servicio (por ejemplo, student.service.ts) se encarga de las operaciones específicas de una colección en Firestore.

todos

Propósito: Carpeta de referencia donde se documentan las tareas y los puntos pendientes de desarrollo.

Genericos

Importancia

funcionalidad

generic.service

Servicio base que proporciona métodos CRUD comunes para interactuar con Firestore. Sirve como una clase que otros servicios pueden extender para reutilizar código.

generic. Controller

Controlador base que implementa métodos comunes para la exposición de rutas HTTP y puede ser extendido por otros controladores específicos.

Flujo de Trabajo

Modulo

Creación del modulo

Crea un archivo de módulo en src/module (por ejemplo, student.module.ts).

Define y exporta el módulo con los controladores y servicios necesarios:

1. grega el módulo al app.module.ts para registrar el módulo en la aplicación principal.

Principales Componentes

- **Decorador @Module**: Define las dependencias del módulo, como controladores y servicios.
- **providers**: Lista de servicios utilizados en el módulo.
- **controllers**: Lista de controladores expuestos por el módulo.

Servicio

Creación del servicio

 Crea un archivo de servicio en src/service (por ejemplo, student.service.ts).

 Define la lógica de negocio y las operaciones con la base de datos para la entidad.

Uso del genérico

Si el servicio realiza operaciones CRUD estándar, extiende el generic.service.ts para reutilizar métodos:

Principales componentes

 Métodos CRUD: Métodos como findAll, findOne, create, update, delete.

 Extensión de GenericService: Permite reutilizar lógica común en otros servicios.

Controlador

Creación del controlador.

 Crea un archivo de controlador en src/controller (por ejemplo, student.controller.ts).

 Define los endpoints que llaman al servicio para realizar operaciones.

Uso del genérico.

Extiende el generic.controller.ts si el controlador sigue una estructura CRUD básica:

typescript

Copiar código

Elementos Clave.

 Métodos HTTP (GET, POST, PUT, DELETE): Definidos en cada endpoint para manejar las operaciones.

 Extensión de GenericController: Simplifica el código reutilizando lógica de endpoints comunes.

Documento

Creación del Documento

Define la interfaz del documento en src/interfaces

Principales componentes

 Propiedades del Documento: Define campos como id, name, email.

 Consistencia con Firestore: Asegura que la estructura del documento coincida con la base de datos.

Firestore.providers.ts

Uso

 Propósito: Configura la conexión y exportación de proveedores de Firestore, permitiendo que los servicios accedan a las colecciones de la base de datos.

 Importancia: Asegura que cada colección esté correctamente registrada y accesible para todos los servicios que la necesiten.

Importancia

Ejemplo de creación de un modulo

Modulo de ejemplo

Servicio de ejemplo

Controller de ejemplo

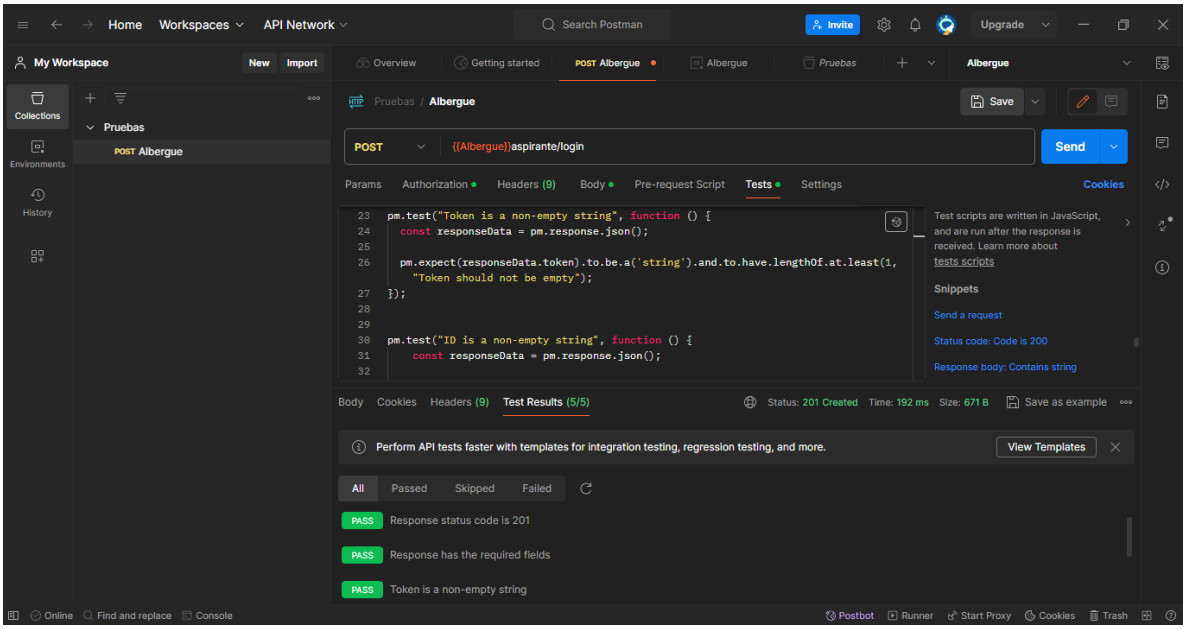
Documento d eejemplo

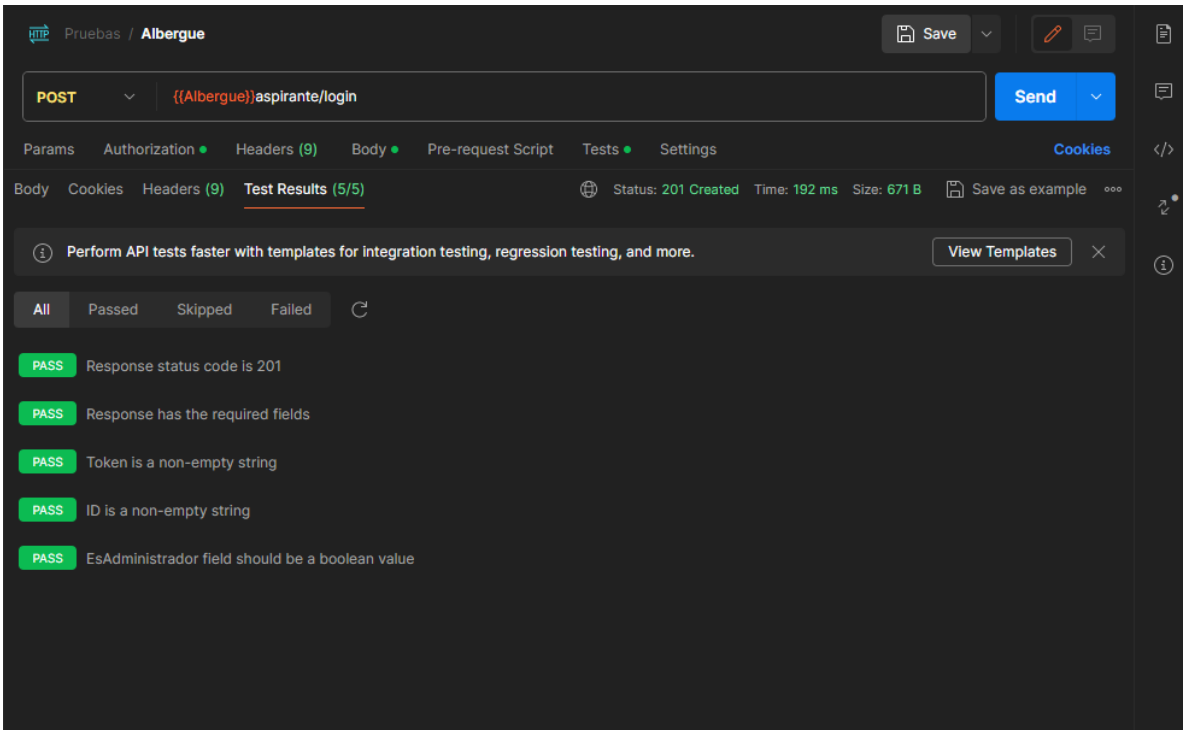
Inyección en providers

Buenas prácticas y Recomendaciones

1. **Modularidad**: Organiza tu código en módulos para mejorar la mantenibilidad y la legibilidad.
2. **Uso de Servicios Genéricos**: Utiliza generic.service y generic.controller siempre que sea posible para reducir duplicación de código.
3. **Manejo de Errores**: Implementa manejo de errores adecuado en controladores y servicios.
4. **Documentación**: Agrega comentarios descriptivos a métodos complejos y asegúrate de que las interfaces estén claramente definidas.

Pruebas


Home
Workspaces
API Network
Search Postman
Invite
Upgrade
My Workspace
New
Import
Collections
Environments
History
Pruebas
POST Albergue
Overview
Getting started
POST Albergue
Albergue
Pruebas
Albergue
Save
POST
{{Albergue}}aspirante/login
Send
Params
Authorization
Headers (9)
Body
Pre-request Script
Tests
Settings
Cookies
pm.test("Token is a non-empty string", function () {
const responseData = pm.response.json();
pm.expect(responseData.token).to.be.a('string').and.to.have.lengthOf.at.least(1,
"Token should not be empty");
});
pm.test("ID is a non-empty string", function () {
const responseData = pm.response.json();
Test scripts are written in JavaScript, and are run after the response is received. Learn more about tests scripts
Snippets
Send a request
Status code: Code is 200
Response body: Contains string
Body
Cookies
Headers (9)
Test Results (5/5)
Status: 201 Created
Time: 192 ms
Size: 671 B
Save as example
Perform API tests faster with templates for integration testing, regression testing, and more.
View Templates
All
Passed
Skipped
Failed
PASS
Response status code is 201
PASS
Response has the required fields
PASS
Token is a non-empty string
Online
Find and replace
Console
Postbot
Runner
Start Proxy
Cookies
Trash


Pruebas / Albergue
Save
POST
{{Albergue}}aspirante/login
Send
Params
Authorization
Headers (9)
Body
Pre-request Script
Tests
Settings
Cookies
Body
Cookies
Headers (9)
Test Results (5/5)
Status: 201 Created
Time: 192 ms
Size: 671 B
Save as example
Perform API tests faster with templates for integration testing, regression testing, and more.
View Templates
All
Passed
Skipped
Failed
PASS
Response status code is 201
PASS
Response has the required fields
PASS
Token is a non-empty string
PASS
ID is a non-empty string
PASS
EsAdministrador field should be a boolean value