



UANL

UNIVERSIDAD AUTÓNOMA DE NUEVO LEÓN



FCFM

FACULTAD DE CIENCIAS FÍSICO MATEMÁTICAS

Desarrollo Web: Back-End

## **PIA: APLICACIÓN CRUD**

Prof. Arturo Gabriel García Fernández

Grupo 031

### **Integrantes:**

Alberto Carlos Almaguer Rodríguez

Ester Abigail Celada López

Destiny Yareli de la Fuente Aguilar

Eduardo Alán Hernández Villasana

Brayan Adrián Montoya Morales

Janyan Aridaí Rodríguez Puente

Ian Mauricio Saucedo Alemán

Mayo 2021

# ÍNDICE

	Página
Introducción	3
Objetivo	3
Tecnologías involucradas	3
Requisitos	4
Software necesario	4
Instalación de software	5
Estructura	14
Tablas por trabajar, Modelos	15
Service Components	16
Controllers	18
Login Controller	20
Cors	22
Pruebas en Postman	23
Front-End	30

## **INTRODUCCIÓN**

Las aplicaciones CRUD son aquellas vinculadas a la gestión de bases de datos. Es decir, CRUD se encarga de las funciones necesarias para que un usuario sea capaz de crear y gestionar esos datos. Esto es logrado por medio de las operaciones básicas: Create, Read, Update y Delete; de donde sale el nombre de la aplicación.

## **OBJETIVO**

El fin de este manual es informar y guiar a desarrolladores y administradores del sistema, sobre la estructura y funcionamiento de la aplicación CRUD.

Esta aplicación tiene el objetivo de realizar operaciones básicas sobre la base de datos NORTHWIND por medio de un front-end (cuyo diseño abarca lo esencial) y un back-end enfocado al desarrollo de API's capaces de satisfacer las peticiones del tipo CRUD que le sean solicitadas.

## **TECNOLOGÍAS INVOLUCRADAS**

C#: Lenguaje predominante en el desarrollo Back-End

TypeScript: Lenguaje predominante en el desarrollo del Front-End

HTML: Utilizado para estructurar la interfaz de la aplicación CRUD

JavaScript: Interactuar con los datos de la aplicación

SCSS: Encargado de dar formato a la interfaz visual de la aplicación.

MySQL: Almacenamiento de datos

JSON: Peticiones

## **REQUISITOS**

Estos requisitos fueron determinados basándonos en los softwares involucrados en el desarrollo de la aplicación:

### **REQUISITOS DE HARDWARE**

- 2 GB de RAM. 4 GB recomendados
- 6 GB de espacio mínimo en disco duro
- Velocidad del procesador mínima: 1.6 GHz. 2GHz recomendados

### **REQUISITOS DE SOFTWARE**

- Sistema Operativo: Windows 10, Windows Server 2016 o más recientes
- .NET framework

## **SOFTWARE NECESARIO**

SQL Server y SQL Management Studio: Almacenamiento de la base de datos.

Visual Studio: Desarrollo de la estructura de la aplicación. Back-End y Front-End

Postman: Usado durante el desarrollo para hacer pruebas de requests.

# INSTALACIÓN DE SOFTWARE

## SQL Server Express y SQL Server Management Studio

1. Descargar instalador: <https://www.microsoft.com/es-mx/sql-server/sql-server-downloads>



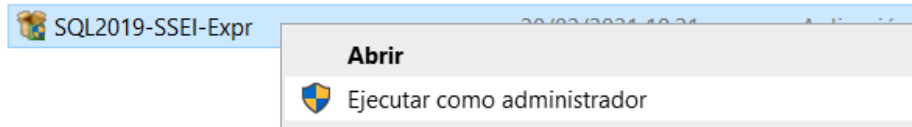
### Express

SQL Server 2019 Express es una edición gratuita de SQL Server, que es ideal para el desarrollo y la producción, para aplicaciones de escritorio, Internet y pequeños servidores.

[Descargue ahora >](#)

*Ilustración 1: Opción a descargar*

2. Empezar instalación seleccionando “Ejecutar como administrador”.



*Ilustración 2: Ejecución del instalador*

3. Seleccionar “Básica” como tipo de instalación



*Ilustración 3: Tipo de instalación*

#### 4. Aceptar advertencias de idiomas

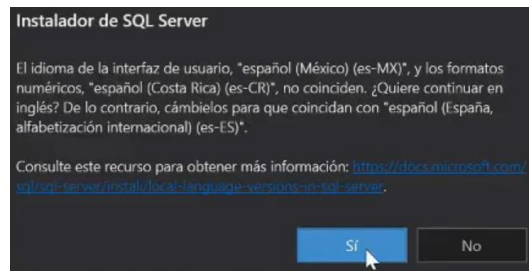


Ilustración 4: Cuadro con mensaje expresando el idioma que se usará

#### 5. Aceptar contrato de licencia para el uso de SQL Server

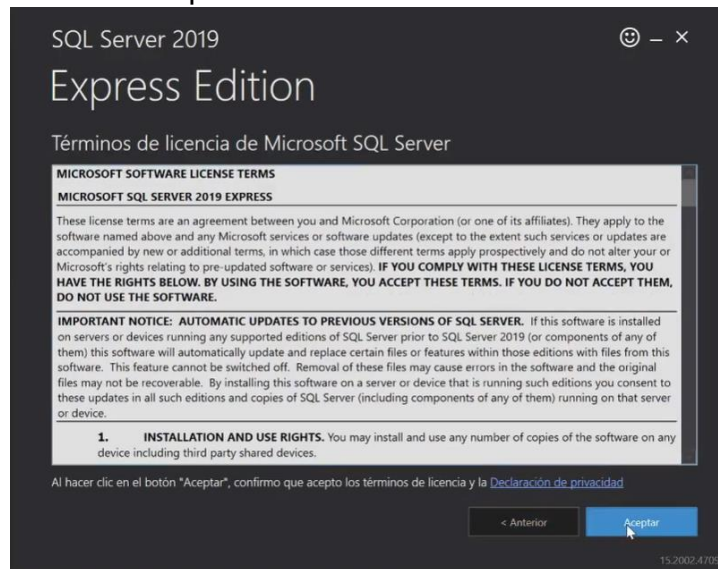


Ilustración 5: Términos de licencia

#### 6. Elegir el directorio de instalación o aceptar el default

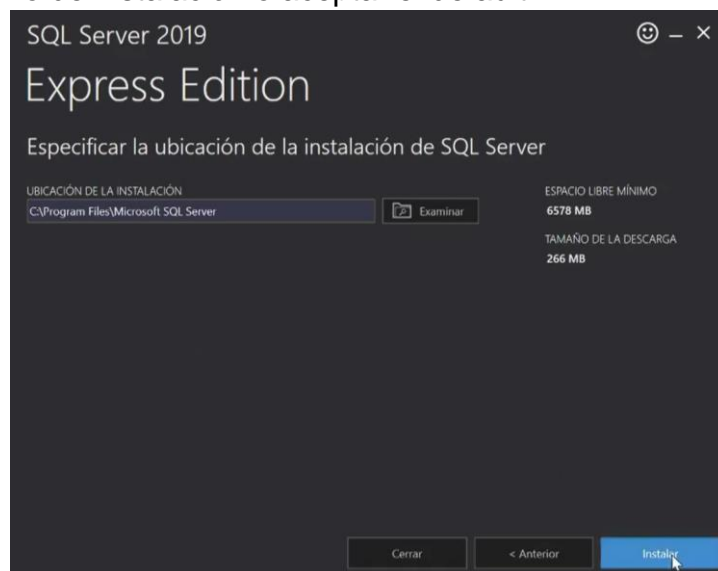


Ilustración 6: Selección del directorio de instalación

7. Esperar por la finalización de la instalación. Si la instalación es exitosa se indicará en un mensaje y se pedirá por instalar SSMS.
8. Seleccionar “Instalar SSMS” para descargar el instalador de SQL Server Management Studio

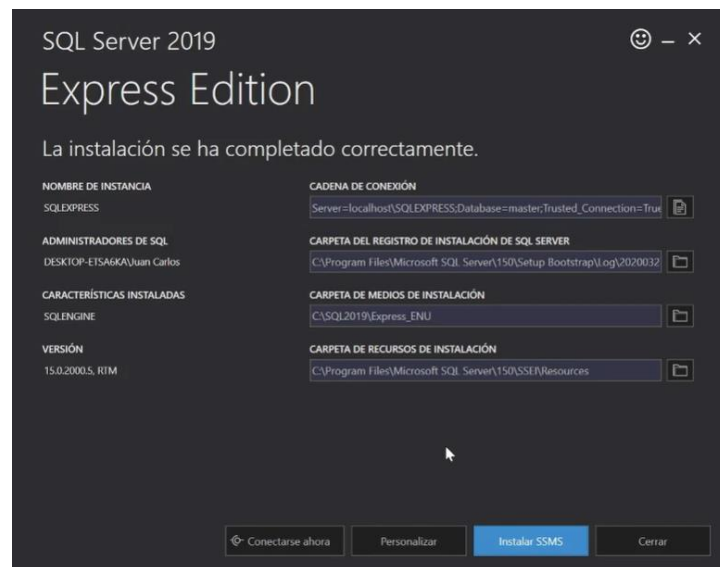


Ilustración 7: Se solicita instalar SSMS para administrar SQL Server

9. Seremos redirigidos a la página: <https://docs.microsoft.com/en-us/sql/ssms/download-sql-server-management-studio-ssms?view=sql-server-ver15>, donde descargaremos el instalador:

## Download SSMS

 [Download SQL Server Management Studio \(SSMS\) 18.9.1](#)

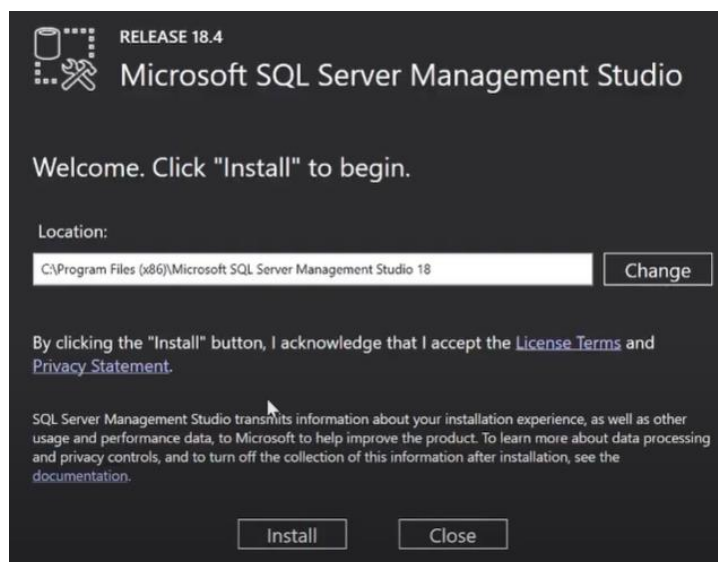
Ilustración 8: Vista de descarga en la página web

10. Procedemos a ejecutar el archivo descargado



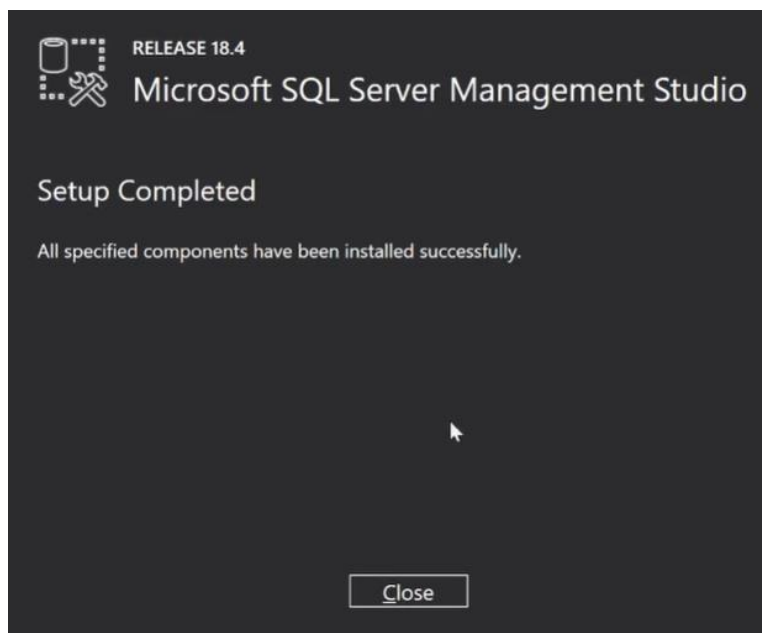
Ilustración 9: Instalador en el explorador de archivos

11. Seleccionar directorio de instalación o aceptar el default. Y seleccionar “Install”



*Ilustración 10: Escribir directorio de instalación para SSMS*

12. Si la instalación fue completada exitosamente se mostrará un mensaje y seremos capaces de encontrar SSMS entre nuestras aplicaciones



*Ilustración 11: Instalación completada con éxito*



### 13. Instalación completa. Software listo para usar.

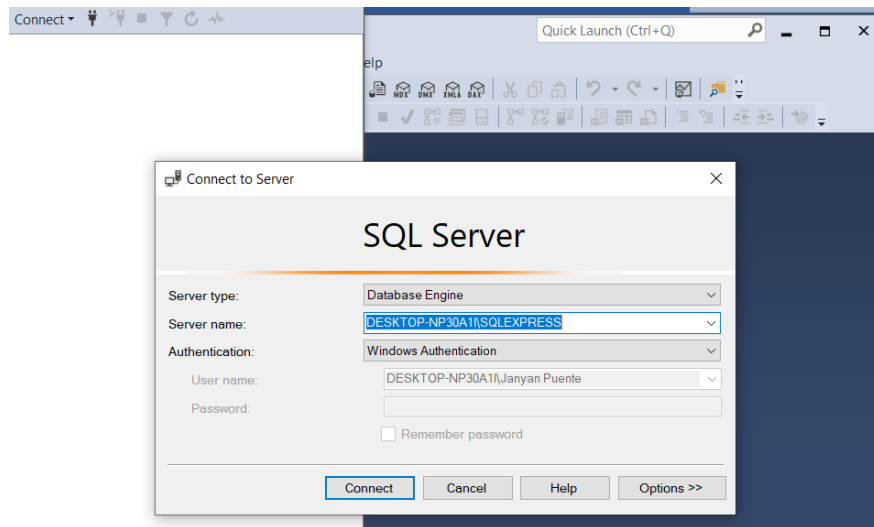


Ilustración 12: SSMS en ejecución

## VISUAL STUDIO

1. Ingresar a <https://visualstudio.microsoft.com/es/downloads/> y descargar instalador de Visual Studio Community



Ilustración 13: Vista de la opción de descarga en la página web

2. Ejecutar el archivo .exe descargado para empezar la instalación. Aceptar cambios en el sistema.

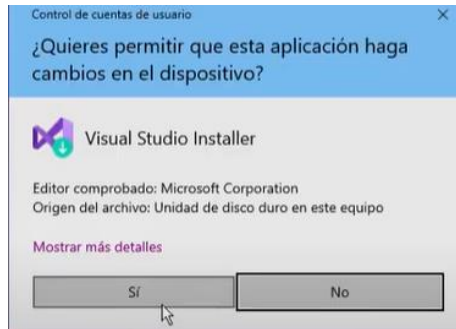


Ilustración 14: Advertencia del equipo



Ilustración 15: Advertencia del instalador

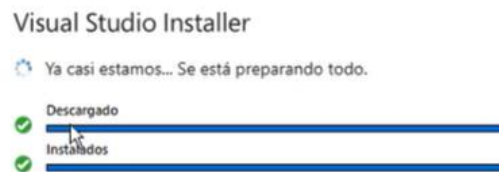


Ilustración 16: Progreso de la descarga e instalación

3. Seleccionar los componentes con los que se quiera trabajar. La aplicación está desarrollada con componentes .NET/ASP.NET). Elegir idioma.

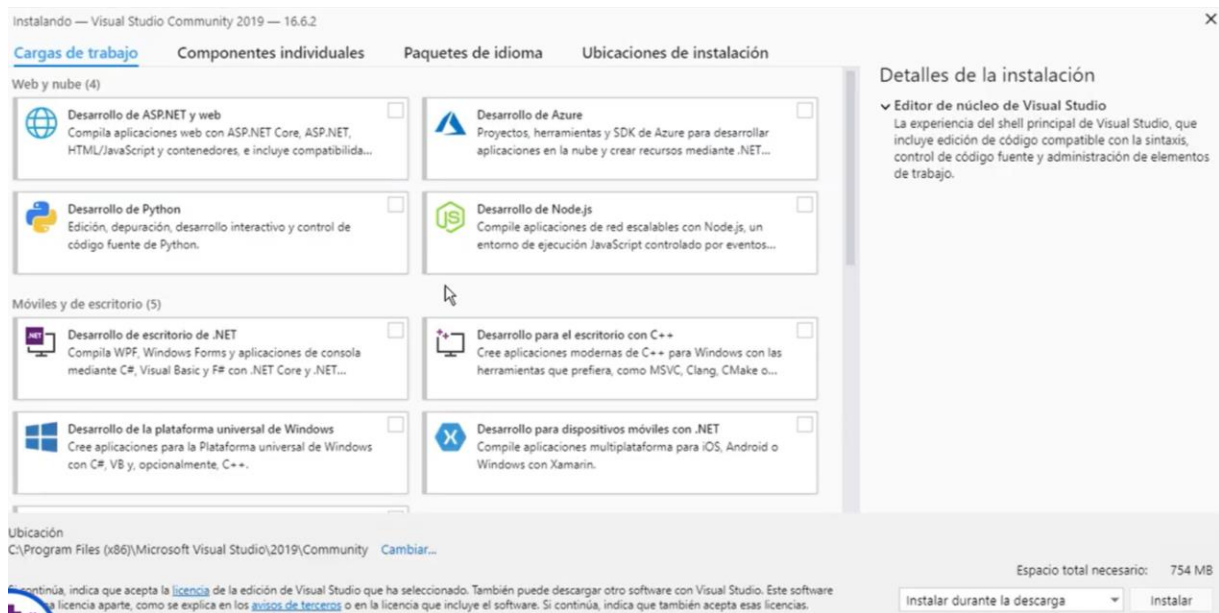
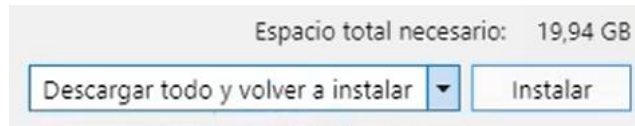


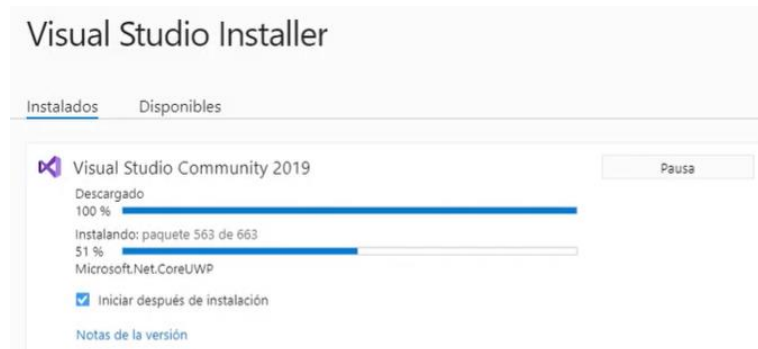
Ilustración 17: Elección de componentes

4. Al finalizar las configuraciones, determinamos el tipo de instalación como “Descargar todo y volver a instalar”. Seleccionar “Instalar”



*Ilustración 18: El tipo de instalación se cambia para que en caso de que la instalación se vea interrumpida, los componentes ya estén descargados y se pueda reanudar la instalación sin problemas*

Esperar a la descarga e instalación de los componentes



*Ilustración 19: Progreso de instalación de los componentes*

5. Iniciar sesión si se cuenta con una cuenta Microsoft u omitir

## Visual Studio

¡Hola!

Conéctese desde aquí a todos sus servicios de desarrollo.

Inicie sesión para empezar a usar los créditos de Azure, publicar código en un repositorio Git privado, sincronizar la configuración y desbloquear el IDE.

[¿Por qué debo iniciar sesión en Visual Studio?](#)

☒ Autenticar en todas las instancias de Azure Active Directory al iniciar sesión

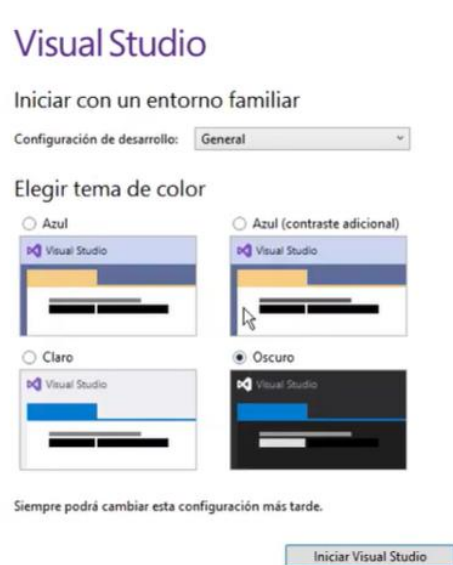
Iniciar sesión

[¿No hay ninguna cuenta? ¡Créela!](#)

[De momento, no; quizás más tarde.](#)

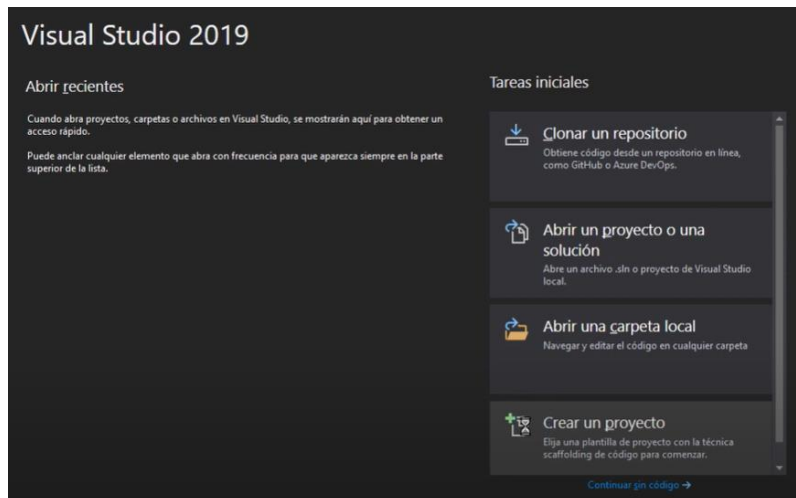
*Ilustración 20: Pantalla de inicio de sesión*

6. A continuación elegir tema de la aplicación y entorno de desarrollo para empezar. Selecciona “Iniciar”



*Ilustración 21: Configuración final*

7. La instalación ha terminado. Listo para empezar a desarrollar.



*Ilustración 22: Visual Studio instalado y en ejecución*

## POSTMAN

1. Descargar instalador de la página: <https://www.postman.com/downloads/>

### The Postman app

The ever-improving Postman app (a new release every two weeks) gives you a full-featured Postman experience.

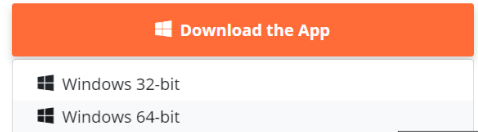


Ilustración 23: Descargar instalador compatible con el equipo

2. Ejecutar instalador .exe para iniciar con la instalación

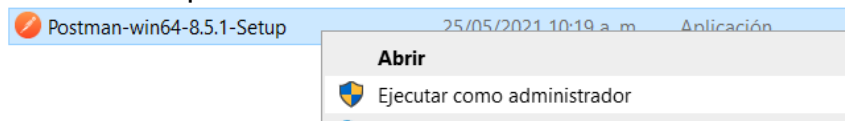


Ilustración 24: Instalador



Ilustración 25: Postman está instalándose

3. Fin de la instalación.

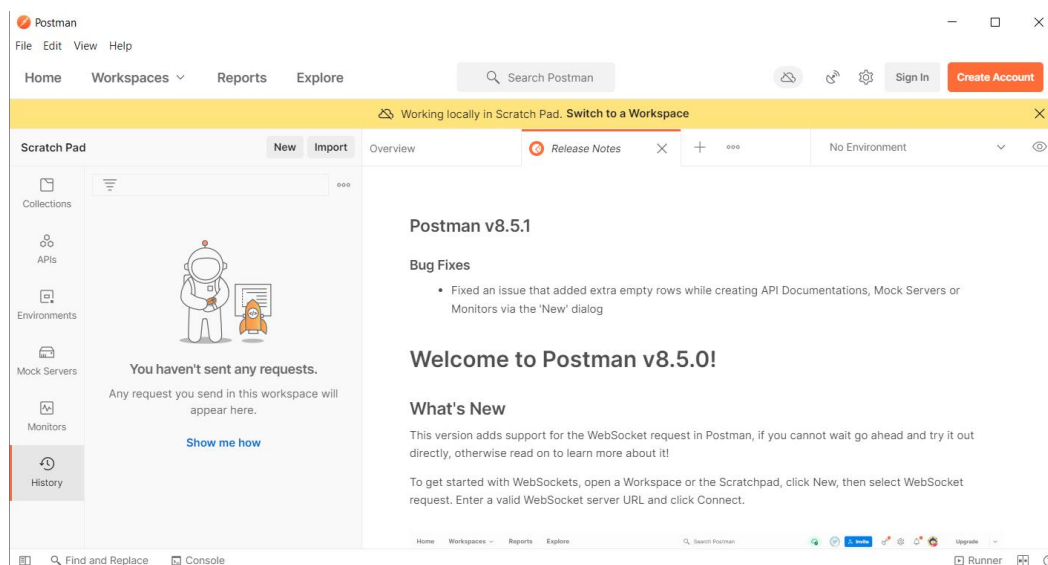
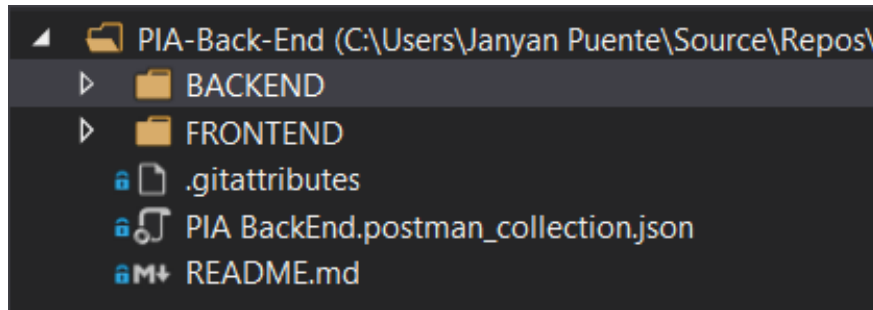


Ilustración 26: Postman listo para ser utilizado

## ESTRUCTURA

La aplicación se divide tajantemente en Front-End y Back-End.

Debido a que la aplicación se enfoca de manera especial en la correcta aplicación de las operaciones básicas CRUD, se le prestará más atención a la descripción del Back-End. El Front-End será explicado enfocándonos en su función.



*Ilustración 27: Estructura principal de la aplicación*

## TABLAS POR TRABAJAR, MODELOS

En la base de datos NORTHWIND se tienen registros de un negocio, entre ellos se encuentran tablas que describen los datos de distribuidores y órdenes de productos. En esta aplicación se trabajarán tres de sus tablas: Employees, Customers y Products.

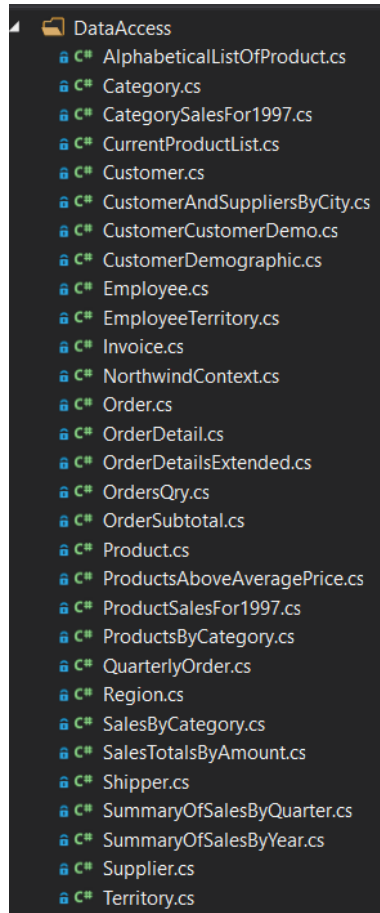


Ilustración 29: La aplicación tiene acceso a todas las tablas y vistas de Northwind, pero la aplicación solo usa las tres tablas ya mencionadas.

Para manejar los datos de los registros desde la aplicación se crearon clases/objetos (uno por tabla) llamados modelos, cuyas variables representan los datos que manejan dichas tablas y tienen propiedades get y set.

Estos modelos serán posteriormente usados por los Controllers y clases donde se definan las funciones que tendrán (archivos SC.cs).

Los archivos que definen los modelos se encuentran en la ruta: BACKEND/DatabaseFirstDWB/DatabaseFirstDWB/ Models

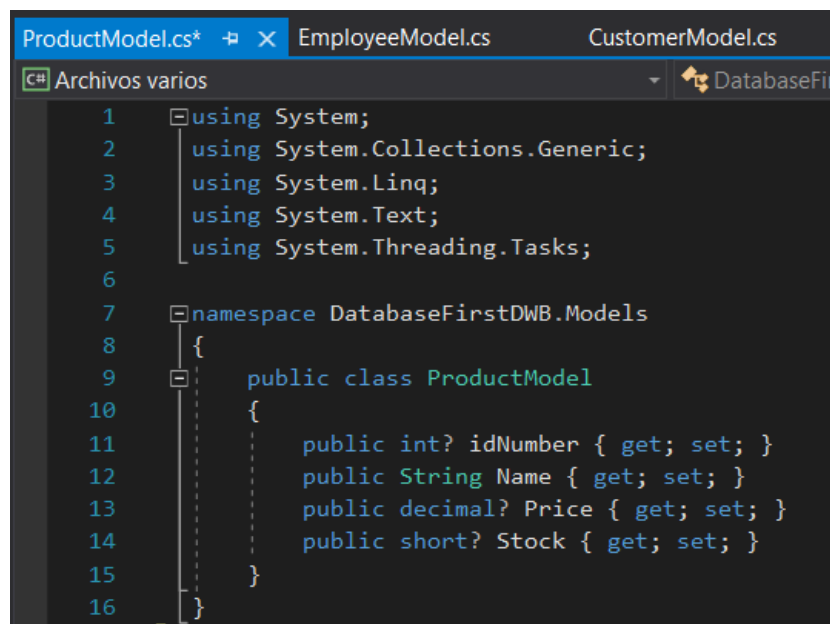


Ilustración 28: Modelo de la tabla Products

## SERVICE COMPONENT

Los controllers se ubican en el directorio: PIA-Back-End\BACKEND\ DatabaseFirstDWB\ DatabaseFirstDWB\ Backend

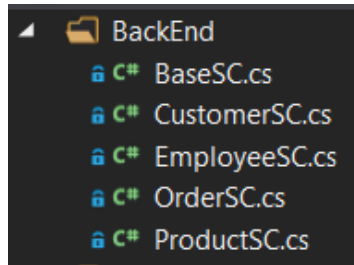


Ilustración 30: Archivos localizados en BACKEND /DatabaseFirstDWB/ DatabaseFirstDWB/ Backend

En los archivos SC.cs se definen las funciones que los modelos de Employee, Customer y Order usarán para obtener, añadir, actualizar o eliminar registros correctamente.

BaseSC es la base de los demás archivos SC.cs, esto para que los demás heredaran de BaseSC la clase DbContext, la cual se encarga de interactuar con la base de datos. Por lo que los demás archivos contienen una clase del tipo BaseScC.

ProductSC, CustomerSC y EmployeeSC tienen funciones para:

- Obtener: todos los registros, registro por ID, todas las ordenes, ordenes por ID, detalles de órdenes, detalles de órdenes por ID

```
public IQueryable<Employee> GetAllEmployees()...
public IQueryable<EmployeeTerritory> GetAllEmployeesTerritories()...
public ICollection<Order> GetOrders()...
public ICollection<OrderDetail> GetOrdersDetail()...
public ICollection<OrderDetail> GetOrdersDetailByOrderId(int orderId)...
public ICollection<Order> GetOrdersByEmployeeId(int employeeid)...
public ICollection<EmployeeTerritory> GetEmployeeTerritoriesByEmployeeId(int employeeid)...
public Employee GetEmployeeById(int id)...
public ICollection<Employee> GetEmployeesReporsto(int id)...
```

Ilustración 31: Funciones Get del Service Component para Employee

- Eliminar: donde el registro a eliminar se elimina por ID. Si tiene datos compartidos con alguna otra tabla, primero se borran los datos y despues el registro.

```
public void RemoveProduct (int id)
{
    //SE GUARDA EN UNA LISTA TODAS LAS ORDENES DE DETALLE MEDIANTE EL ID
    var orderdetailist = GetOrderDetailsByID(id);
    foreach(OrderDetail od in orderdetailist)
    {
        //SE BORRA CADA DETALLE DE ORDEN DE CADA PRODUCTO
        dbContext.OrderDetails.Remove(od);
    }
    dbContext.SaveChanges();
    //SE BUSCA EL ID DEL PRODUCTO EN LA LISTA
    var removedProduct = GetProductById(id);
    //SE ELIMINA EL PRODUCTO
    dbContext.Products.Remove(removedProduct);
    dbContext.SaveChanges();
}
```

Ilustración 32: Metodología de la función Remove



- Actualizar: Se busca el registro con el ID correspondiente y se reemplazan los datos del registro por los nuevamente ingresados. Los nuevos datos se manejan por medio de un modelo.

```
public void UpdateProduct (int id, ProductModel product)
{
    Product currentProduct = GetProductById(id);
    if (currentProduct == null)
    {
        throw new Exception("No se encontró el producto con el ID proporcionado");
    }

    currentProduct.ProductName = product.Name;
    currentProduct.UnitPrice = product.Price;
    currentProduct.UnitsInStock = product.Stock;

    dbContext.SaveChanges();
}
```

*Ilustración 33: Metodología de la función Update*

- Agregar: Se crea un nuevo modelo y se le asignan los datos ingresados para posteriormente añadirlos a la base de datos.

```
public void AddProduct(ProductModel newProduct)
{
    var newProductRegister = new Product()
    {
        ProductName = newProduct.Name,
        UnitPrice = newProduct.Price,
        UnitsInStock = newProduct.Stock
    };

    dbContext.Products.Add(newProductRegister);
    dbContext.SaveChanges();
}
```

*Ilustración 34: Metodología de la función Add*

Después de la operación realizada confirmamos a la base de datos los cambios hechos por medio de: `dbContext.SaveChanges()`.

A diferencia de Product, Customer y Employee, OrderSC.cs solo tiene dos funciones: Obtener órdenes por ID y obtener todas las órdenes.

## CONTROLLERS

Los controllers se ubican en el directorio: PIA-Back-End\BACKEND\ DatabaseFirstDWB\ ApiRestNorthwind\ Controllers

Los controllers se encargan de manejar las peticiones HTTP (GET, POST, PUT, DELETE). Los controllers no son de la clase ApiController, si no de ControllerBase, de donde se obtiene la estructura básica de las funciones, sin embargo, el funcionamiento de estas está determinado por el desarrollador.

Los controllers utilizan los Service Components y modelos para manejar los datos solicitados en las peticiones (haciendo uso de las funciones/propiedades que poseen).

Hay dos API's definidas para GET, la diferencia entre ambas es que una obtiene la lista completa de los registros, mientras la otra obtiene el registro cuya ID coincide con el buscado.

El HTTP que obtiene la lista completa no tiene necesidad de solicitar ningún parámetro, mientras que el otro está en necesidad de recibir el ID del registro buscado.

```
// GET: api/<EmployeeController>
[HttpGet]
public List<EmployeeModel> Get() //Obtiene una lista de los empleados
{
    var employees = new EmployeeSC().GetAllEmployees().Select(s => new EmployeeModel
    {
        IdNumber = s.EmployeeId,
        Name = s.FirstName,
        FamilyName = s.LastName
    }).ToList();
    return employees;
}
```

*Ilustración 35: Metodología de API GET para obtener todos los registros*

```
// GET api/<EmployeeController>/5
// Busca una instancia de la tabla empleados por un id
[HttpGet("{id}")]
public EmployeeModel Get(int id)
{
    var employee = new EmployeeSC().GetEmployeeById(id);
    var employeeModel = new EmployeeModel();
    try // Como el metodo GetEmployeeById devuelve un null si no lo encuentra se puso en un try catch
    { // ya que el EmployeeModel no acepta datos nulos
        employeeModel.IdNumber = employee.EmployeeId;
        employeeModel.Name = employee.FirstName;
        employeeModel.FamilyName = employee.LastName;
    }
    catch
    {
        employeeModel.IdNumber = null;
        employeeModel.Name = "";
        employeeModel.FamilyName = "";
    }
    return employeeModel;
}
```

*Ilustración 36: Metodología de API GET para obtener el registro por ID.*

Las API's para POST, PUT y DELETE se limitan a hacer uso de las funciones declaradas en los Service Components.

POST recibe como parámetro un objeto del tipo Model. PUT necesita recibir un ID, y un objeto Model. Mientras que DELETE necesita recibir solo el ID.

```
// POST api/<EmployeeController>
// Agrega una instancia de empleado a la tala Employee
[HttpPost]
public void Post([FromBody] EmployeeModel newEmployee)
{
    .....
    var employee = new EmployeeSC();
    employee.AddEmployee(newEmployee);
}

// PUT api/<EmployeeController>/5
// Actualiza una instancia de la tabla Employee
[HttpPut("{id}")]
public void Put(int id, [FromBody] EmployeeModel employee)
{
    .....
    new EmployeeSC().UpdateEmployee(id, employee);
}

// DELETE api/<EmployeeController>/5
// Elimina una instancia de la tabla Employee por id
[HttpDelete("{id}")]
public void Delete(int id)
{
    .....
    new EmployeeSC().FireEmployee(id);
}
```

*Ilustración 37: API's POST, PUT, DELETE aplicadas a Employee*

## LOGIN CONTROLLER

LoginController es el controlador de un JWT, el cual se encarga de autenticar el inicio de sesión de los usuarios. Para lograr esto hacemos uso de cuatro métodos.

1. LoginController: Recibiendo una variable del tipo IConfiguration, funciona como constructor para \_config, variable que usaremos en los demás métodos.

```
public LoginController(IConfiguration config)
{
    _config = config;
}
```

*Ilustración 38: Constructor para \_config*

2. AuthenticateUser: recibe como parámetro un LoginModel y revisa que los datos del modelo coincidan con los del usuario que tiene acceso a la aplicación, si coinciden, LoginModel pasa por un proceso de decodificación, en éste se decide si se cargan los datos del inicio de sesión o no.

```
private LoginModel AuthenticateUser(LoginModel login)
{
    //Username: Juan 123
    //Password: Password123
    //Hashed Password: LVJc4+gPYqzez3o10o2eD9QNEEN/foR+uTaGG93BpVyjTOSmE
    //Si el usuario es Juan, se cargan sus datos
    if (login.Username == "Juan 123")
    {
        string savedPasswordHash = "LVJc4+gPYqzez3o10o2eD9QNEEN/foR+uTaGG93BpVyjTOSmE";
        byte[] hashBytes = Convert.FromBase64String(savedPasswordHash);
        byte[] salt = new byte[16];
        Array.Copy(hashBytes, 0, salt, 0, 16);
        var pbkdf2 = new Rfc2898DeriveBytes(login.HashedPassword, salt, 100000);
        byte[] hash = pbkdf2.GetBytes(20);
        for (int i = 0; i < 20; i++)
            if (hashBytes[i + 16] != hash[i])
                return null;
        return login;
    }
    return null;
}
```

*Ilustración 39: Autenticador y decodificador*

3. GenerateJSONWebToken: Recibe un LoginModel y se crea un nuevo Token con expiración de dos horas.

```
private string GenerateJSONWebToken(LoginModel loginInfo)
{
    var securityKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(_config["Jwt:Key"]));
    var credentials = new SigningCredentials(securityKey, SecurityAlgorithms.HmacSha256);

    var claims = new[] {
        new Claim(JwtRegisteredClaimNames.Sub, loginInfo.Username),
    };

    var token = new JwtSecurityToken(_config["Jwt:Issuer"],
        _config["Jwt:Issuer"],
        claims,
        expires: DateTime.Now.AddMinutes(120), //Expiración de dos horas
        signingCredentials: credentials);

    return new JwtSecurityTokenHandler().WriteToken(token); //Se devuelve el Token generado
}
```

*Ilustración 40: Generador de un nuevo Token*

4. Login: En Login se implementan los métodos anteriores. Recibe un LoginModel y lo pasa por AuthenticateUser, el LoginModel que regresa es pasado como parámetro a GenerateJSONWebToken (si no es nulo) para generar un nuevo Token y se inicia la sesión

```
public IActionResult Login(LoginModel login)
{
    IActionResult response = Unauthorized();
    var user = AuthenticateUser(login);

    if (user != null)
    {
        var tokenString = GenerateJSONWebToken(user);
        response = Ok(new { token = tokenString });
    }

    return response;
}
```

*Ilustración 41: Login*

## CORS

Controllers, Cors y el esquema de autenticación JWT son agregados o inicializados en el método `ConfigureServices` del archivo `Startup.cs`

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddControllers();

    //SE AGREGAN LOS COORS PARA QUE NOS DE ACCESO A LA INFORMACION EN NUESTRO FRONTEND
    services.AddCors(options => options.AddPolicy("AllowWebApp", builder => builder.AllowAnyOrigin().AllowAnyMethod().AllowAnyHeader()));
    services.AddSwaggerGen(c =>
    {
        c.SwaggerDoc("v1", new OpenApiInfo { Title = "ApiRestNorthwind", Version = "v1" });
    });
    //Configuración del esquema de autenticación para JWT
    services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)
        .AddJwtBearer(options =>
        {
            options.TokenValidationParameters = new TokenValidationParameters
            {
                ValidateIssuer = true,
                ValidateAudience = true,
                ValidateLifetime = true,
                ValidateIssuerSigningKey = true,
                ValidIssuer = Configuration["Jwt:Issuer"],
                ValidAudience = Configuration["Jwt:Issuer"],
                IssuerSigningKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(Configuration["Jwt:Key"]))
            };
        });
    services.AddMvc();
}
```

*Ilustración 42: ConfigureServices*

```
// This method gets called by the runtime. Use this method to configure the HTTP request pipeline.
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
        app.UseSwagger();
        app.UseSwaggerUI(c => c.SwaggerEndpoint("/swagger/v1/swagger.json", "ApiRestNorthwind v1"));
    }

    app.UseHttpsRedirection();

    app.UseRouting();

    //Métodos para que el servicio de autenticación está disponible para la aplicación
    app.UseAuthentication();
    //app.UseMvc();

    app.UseCors("AllowWebApp");

    app.UseAuthorization();

    app.UseEndpoints(endpoints =>
    {
        endpoints.MapControllers();
    });
}
```

*Ilustración 43: En el método Configure se encuentran los métodos que abren paso al uso de los autenticadores para el login*

## PRUEBAS POSTMAN

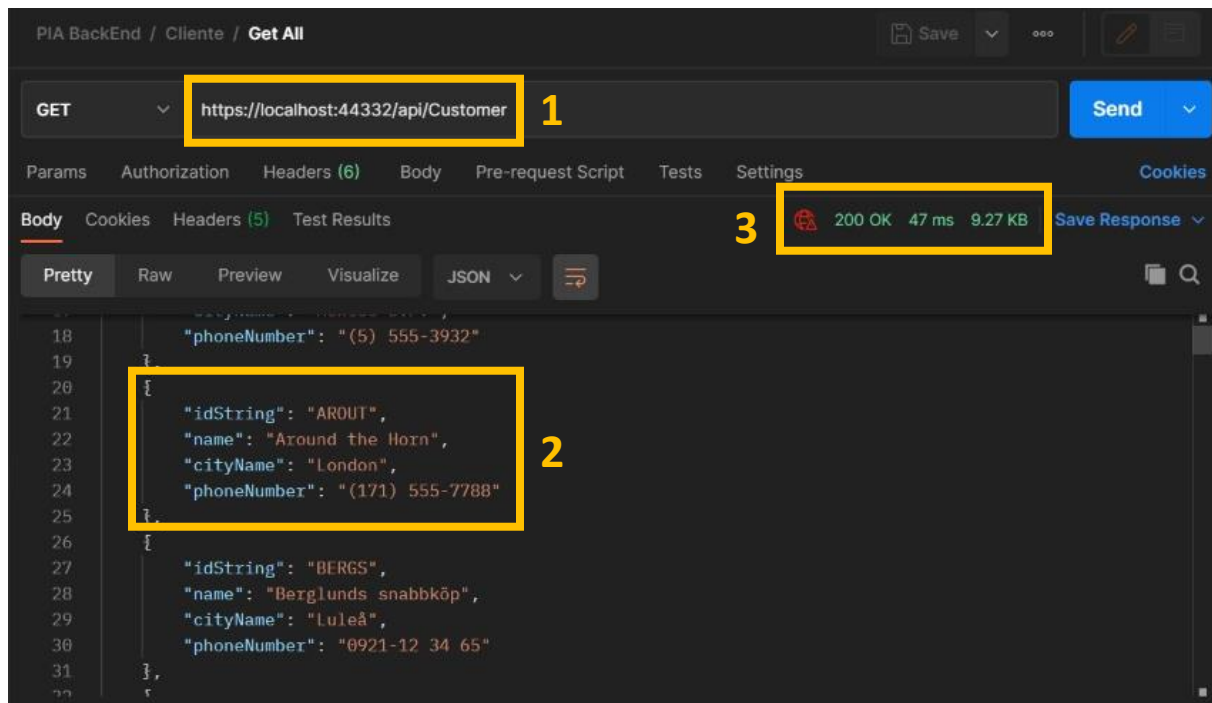


Ilustración 44: 1) URL: Se compone de `https://localhost:44332/api/.../...`, siendo el primer "..." el nombre de la tabla, y el segundo "..." el parámetro que se le da a la API para realizar la operación. 2) Los parámetros son los mismos que los campos de las tablas o registros. 3) La clave "200 OK", nos indica que la operación/petición fue realizada con resultados exitosas, de no haber sido así se muestra el mensaje "500 Internal Server Error"

## CUSTOMER

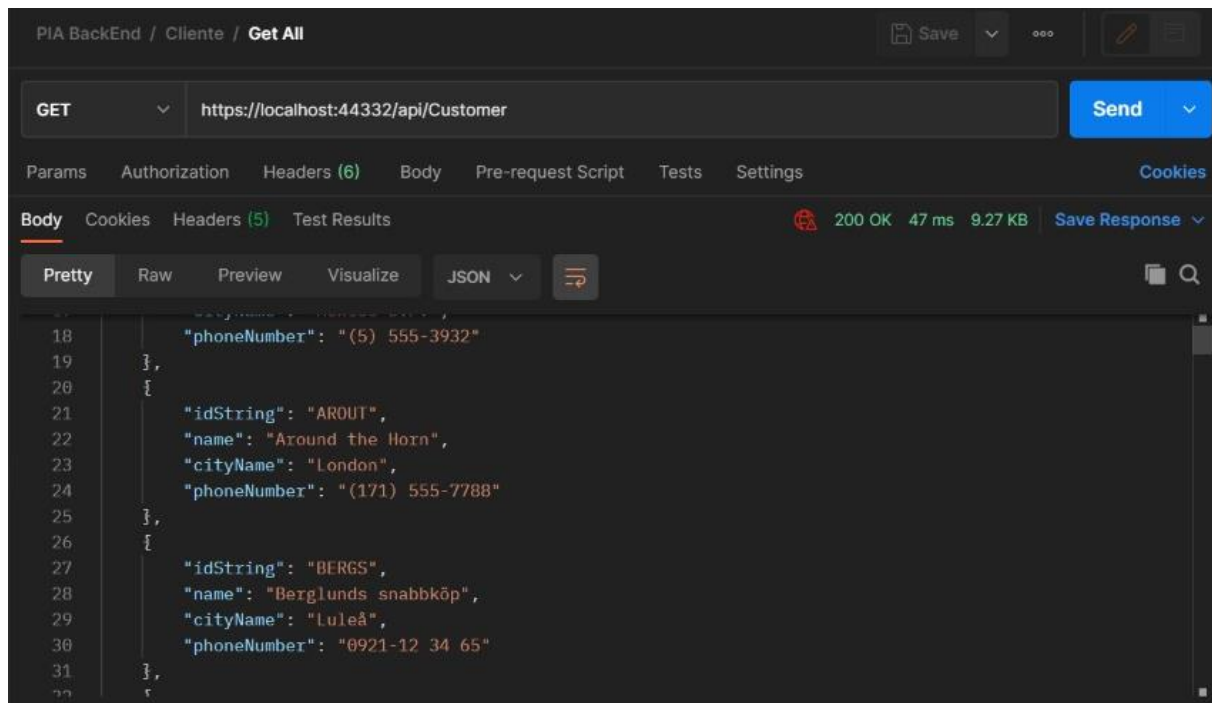


Ilustración 45: Get All – Customer

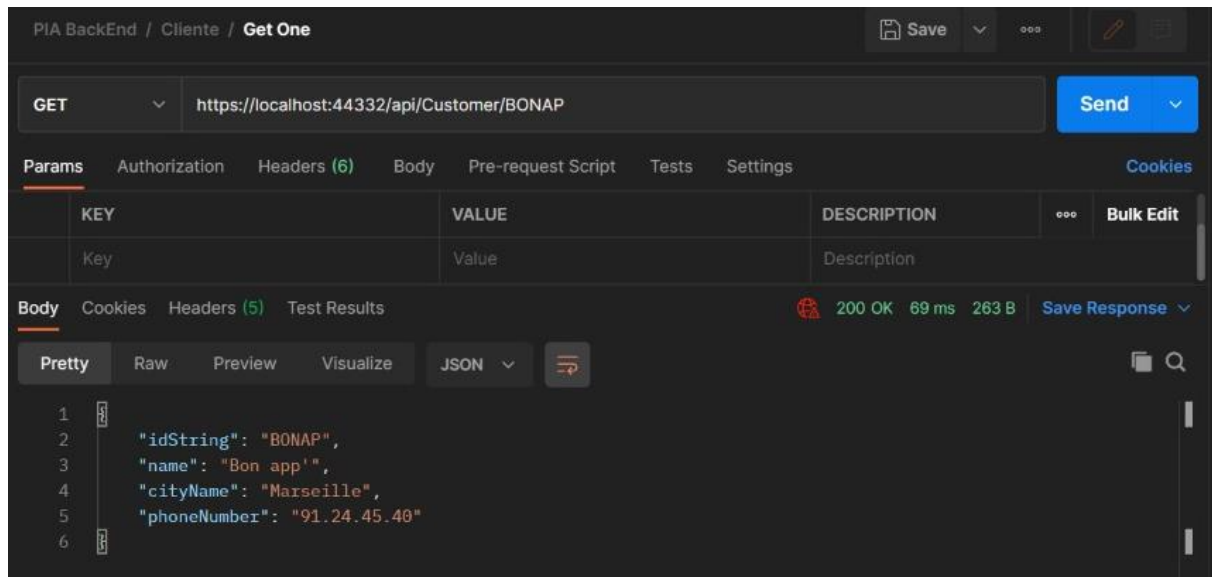


Ilustración 46: Get One – Customer. ID=BONAP

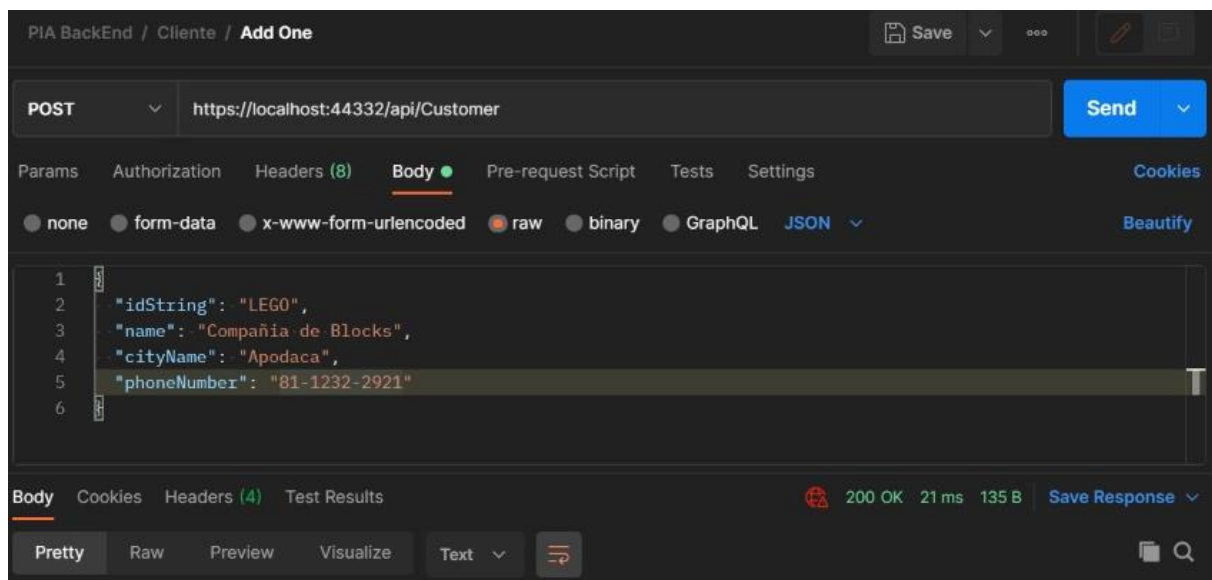


Ilustración 47: Add – Customer



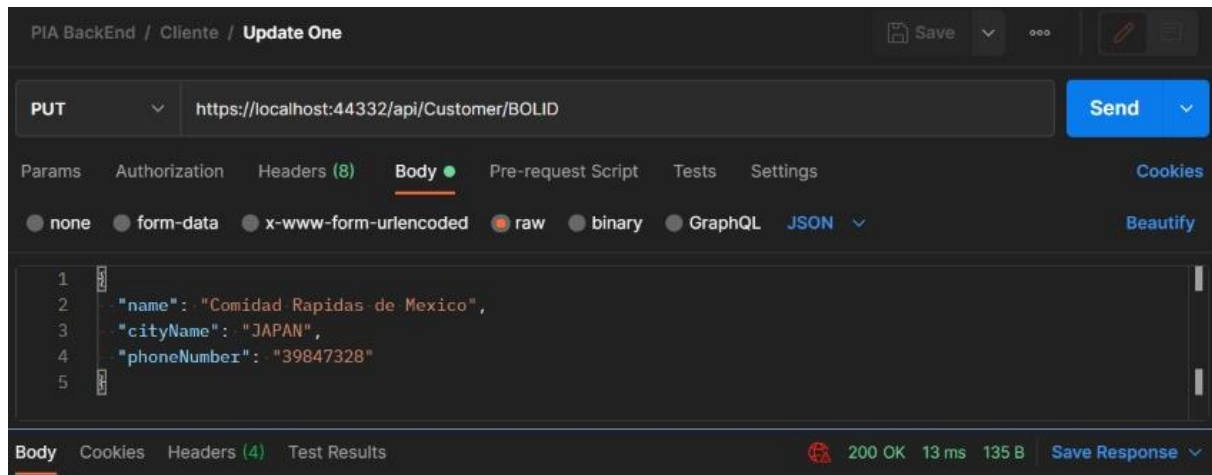


Ilustración 48: Update – Customer. ID=BOLID

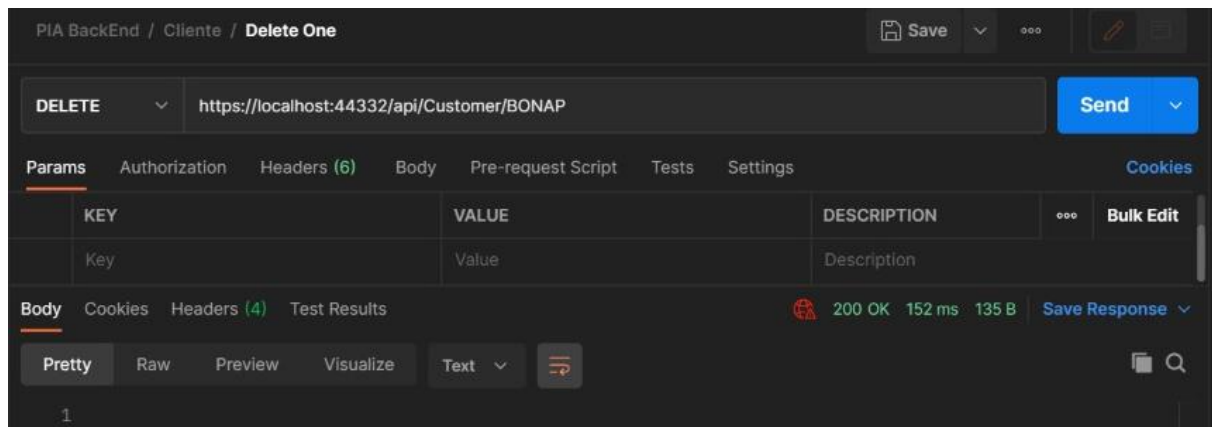


Ilustración 49: Delete – Customer. ID=BONAP

# EMPLOYEE

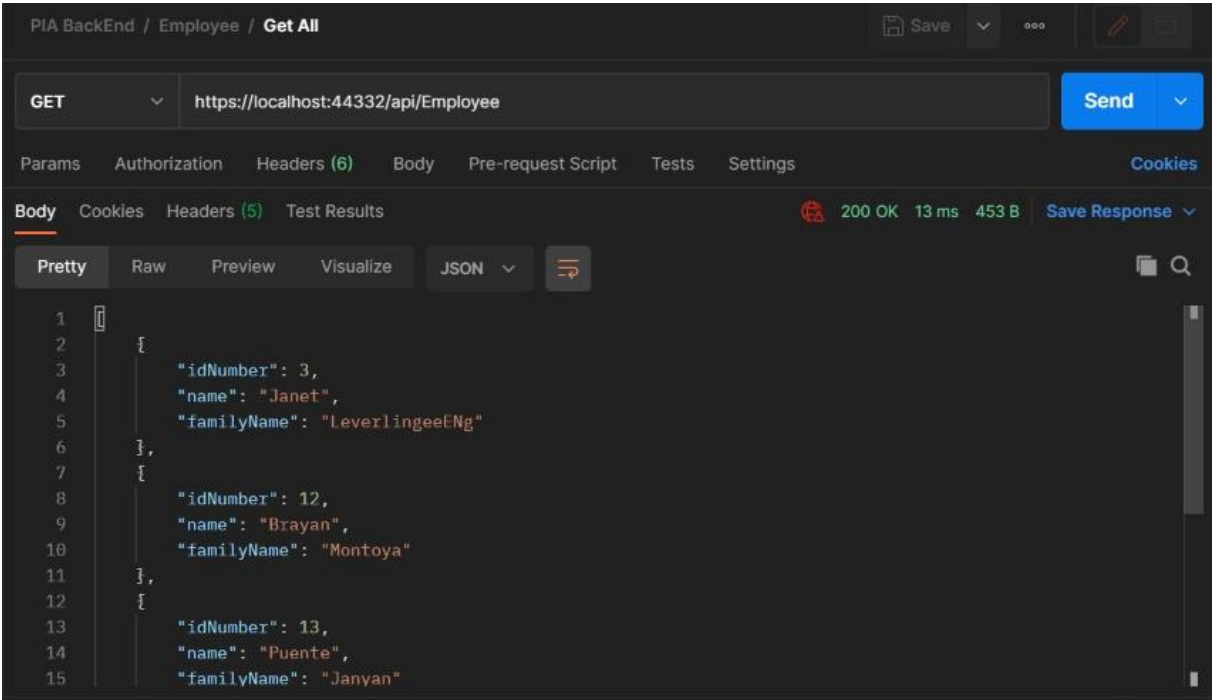


Ilustración 50: Get All – Employee

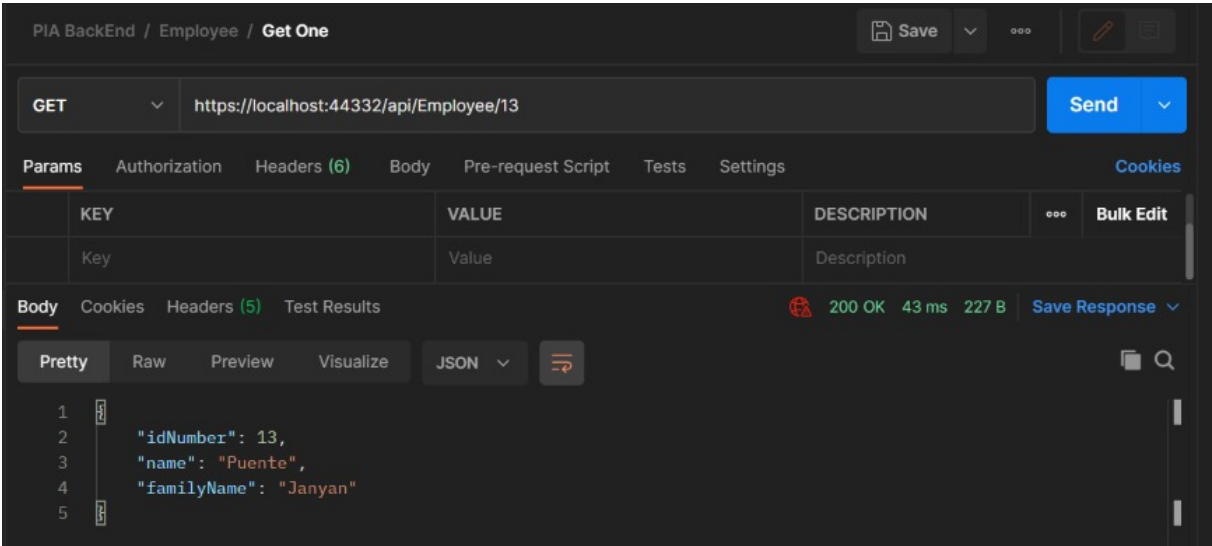


Ilustración 51: Get One - Employee. ID=13

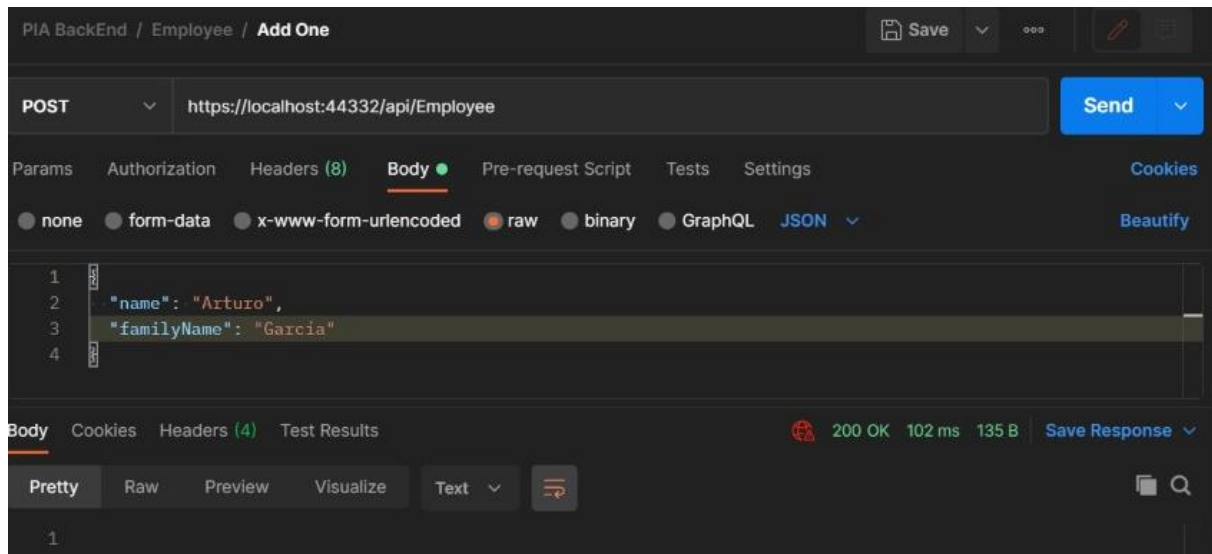


Ilustración 52: Add - Employee

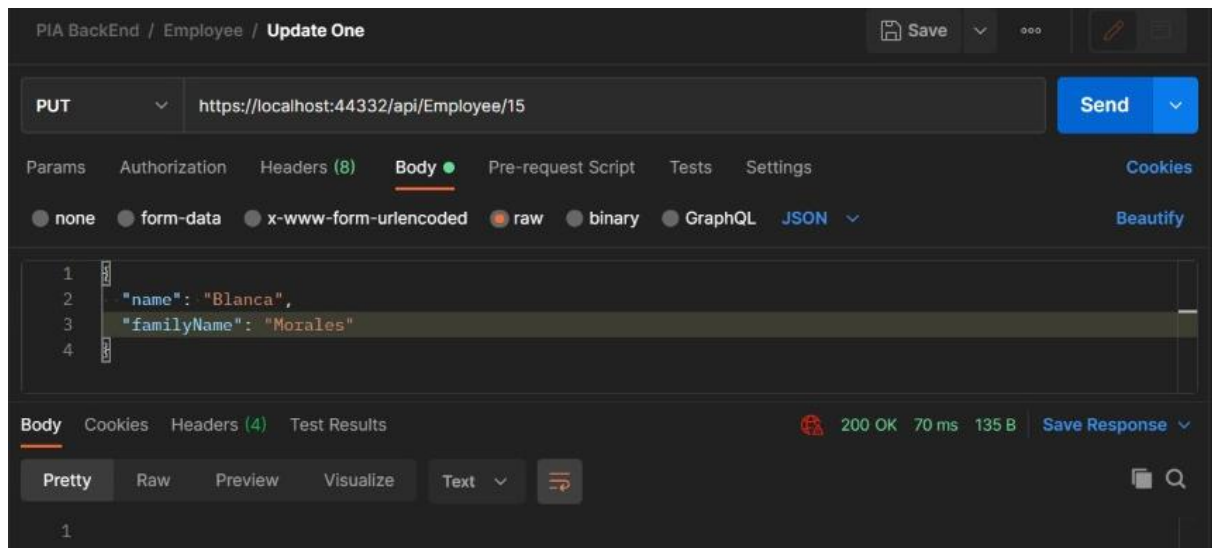


Ilustración 53: Update - Employee. ID=15

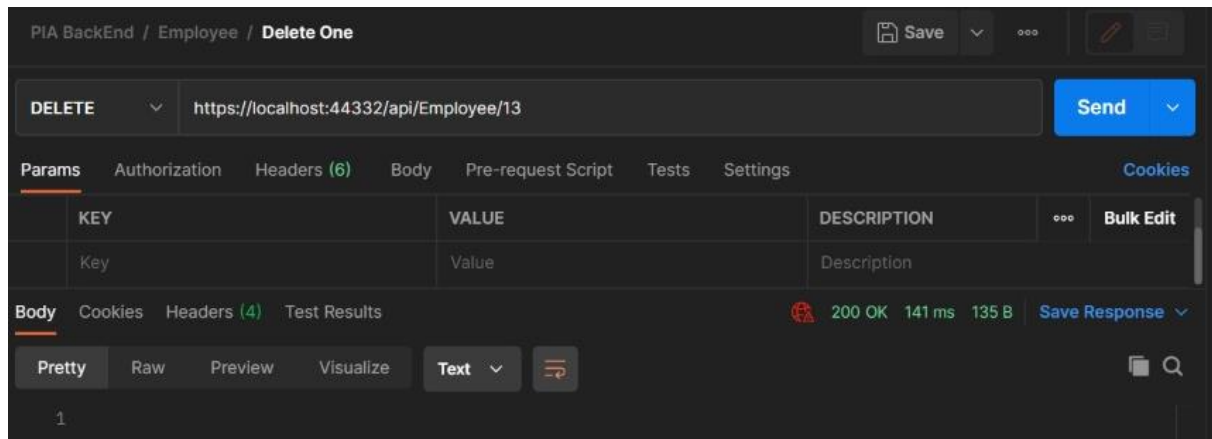


Ilustración 54: Delete - Employee. ID=13

## PRODUCT

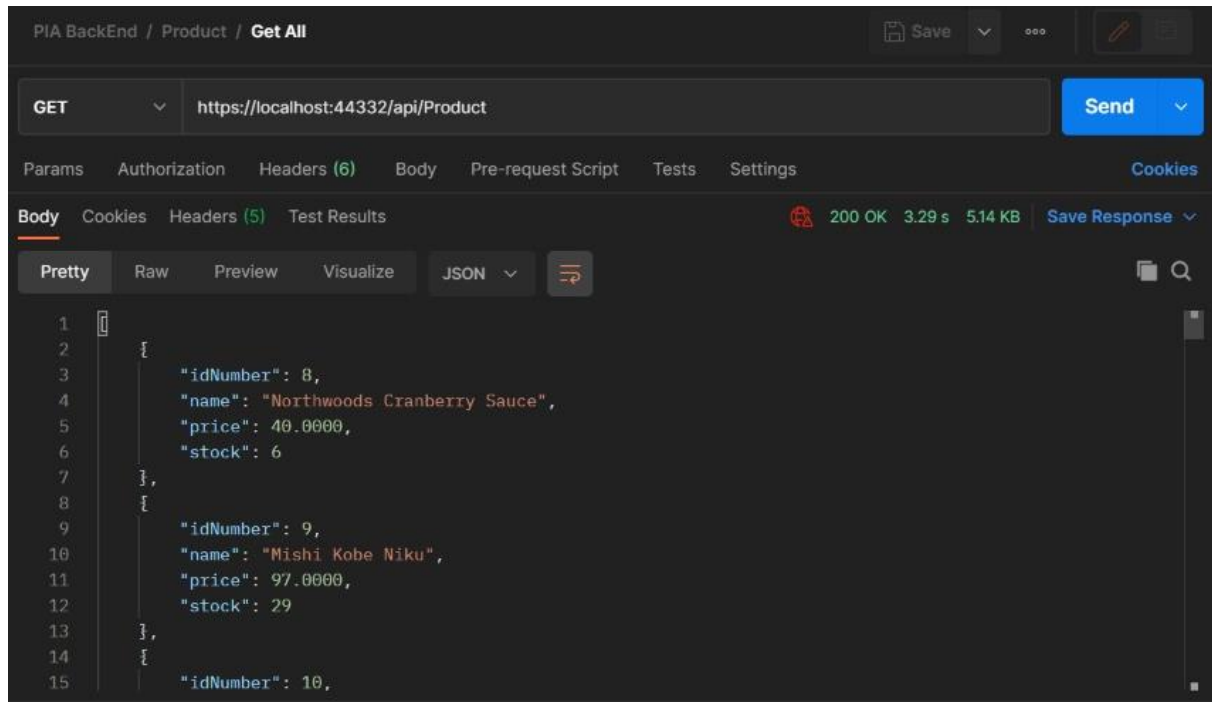


Ilustración 55: Get All – Product

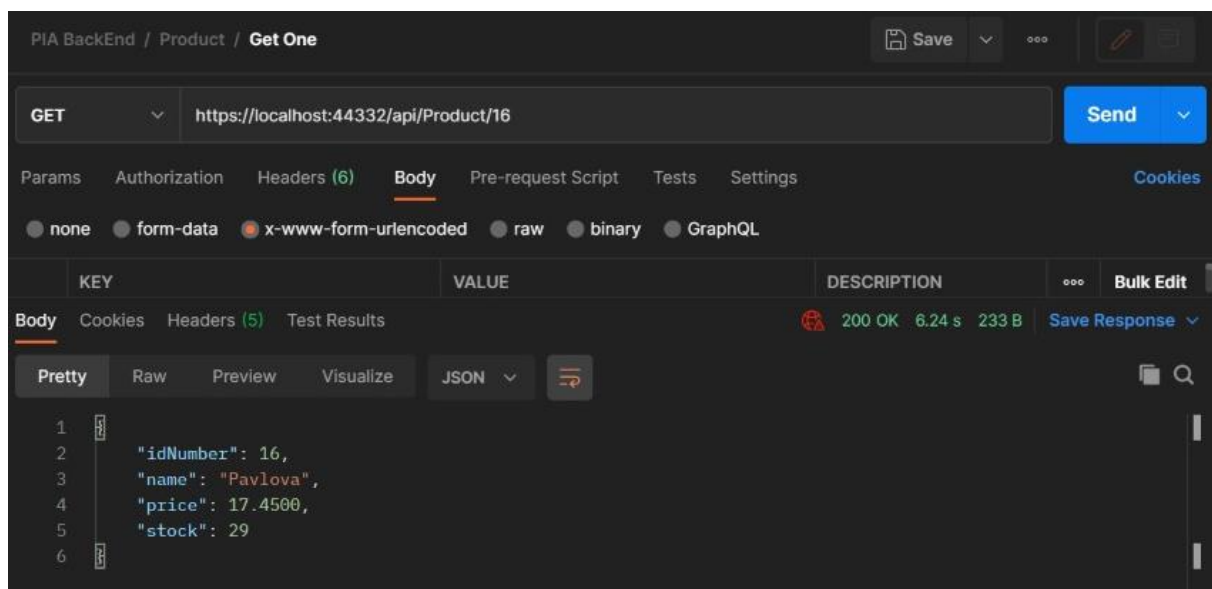


Ilustración 56: Get One - Product. ID=16

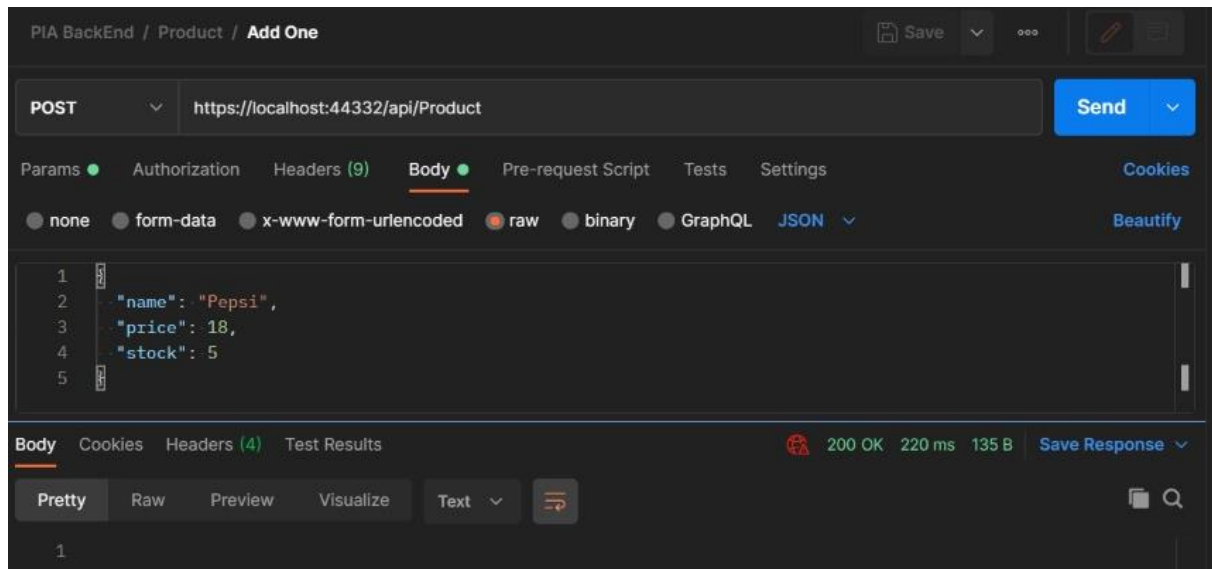


Ilustración 57: Add - Product

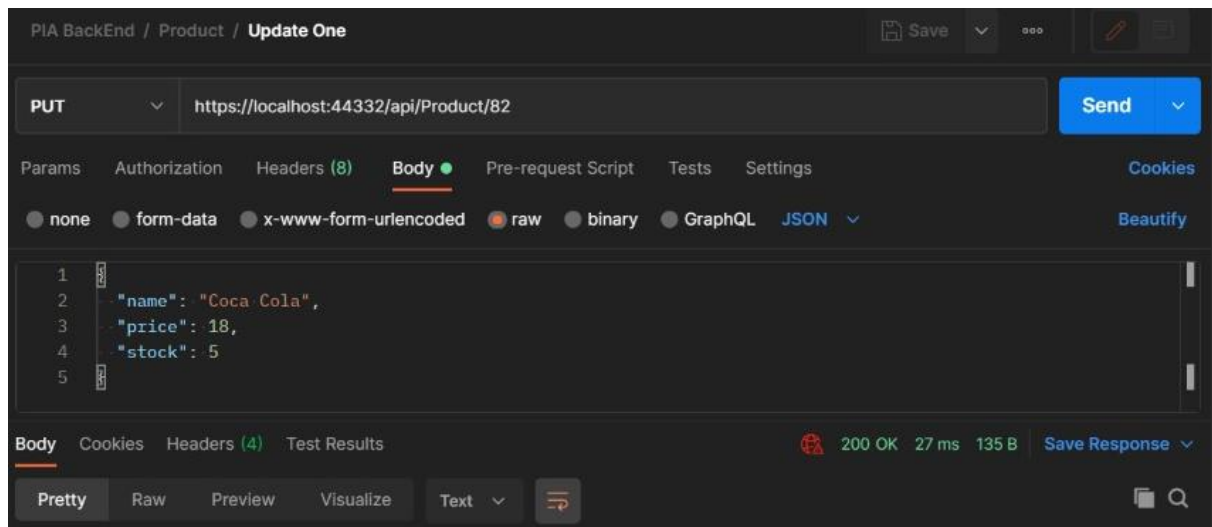


Ilustración 58: Update - Product. ID=82

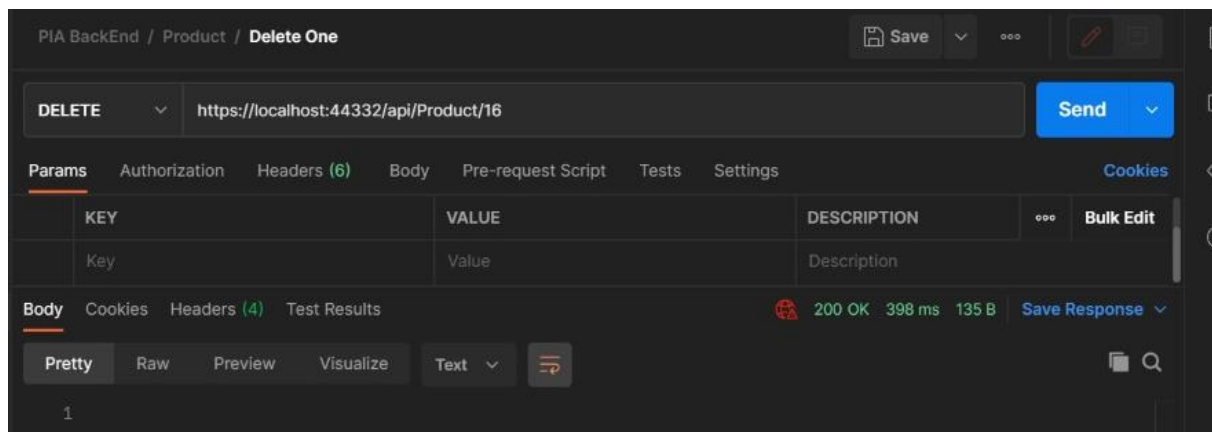


Ilustración 59: Delete - Product. ID=16

## FRONT-END

La interfaz del cliente, es decir el front end de la aplicación, se hizo utilizando el framework de Angular. Para iniciar el proyecto nos ubicamos en la carpeta donde guardamos el proyecto en una consola y hacemos la instalación de los paquetes de Angular utilizando el comando **npm install**

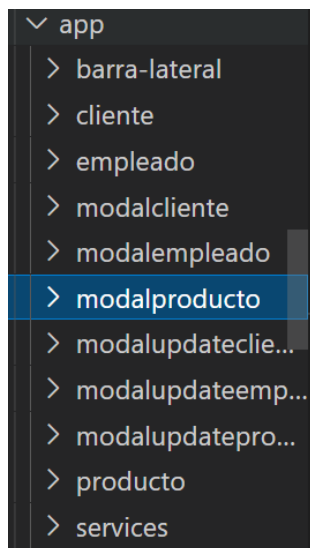
```
C:\Users\Abigail\Documents\GitHub\PIA-Back-End\FRONTEND>npm install
```

*Ilustración 60: Vista cmd*

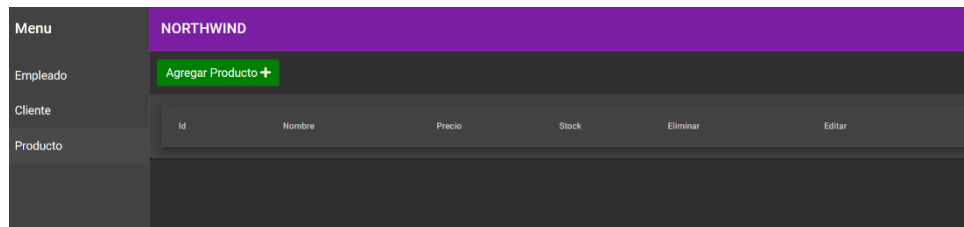
Después ejecutamos el comando **ng serve** para tener el proyecto andando en nuestro servidor local por default.

Una vez iniciado se ve de la siguiente forma:

Consta de 11 componentes que trabajando en conjunto hacen posible la interfaz visual



*Ilustración 61: Componentes Front-End*



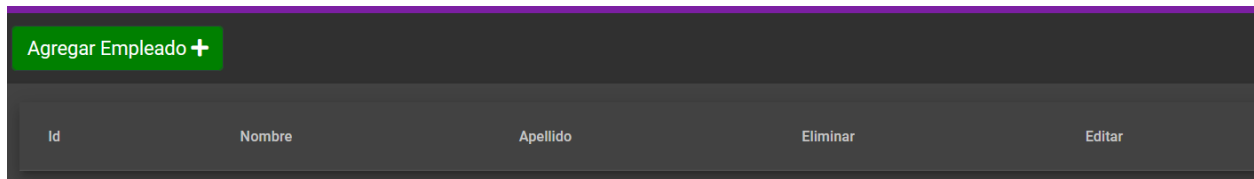
*Ilustración 62: Vista Front-End*

Los componentes empleados, cliente y producto cuentan con métodos que agregan, editan o eliminan datos. Estos a su vez, mandan a llamar a un modal correspondiente que aquí es donde está la lógica de guardar la información o mandar a llamar un servicio POST. Es decir, la lógica del modal se manda a llamar cuando se va a agregar a un cliente, empleado o producto, y además, si se va a actualizar su información. Por la parte del .ts

del componente, este se suscribe a la respuesta del modal para esperar su respuesta de la acción.

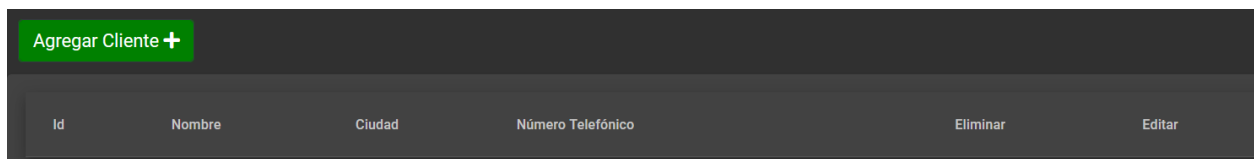
En caso que no se ocupe el modal, también se encuentra la función para eliminar algún registro, que se hace por medio de su ID.

En la parte del html de los componentes está el diseño de las tablas donde se va almacenar la información.



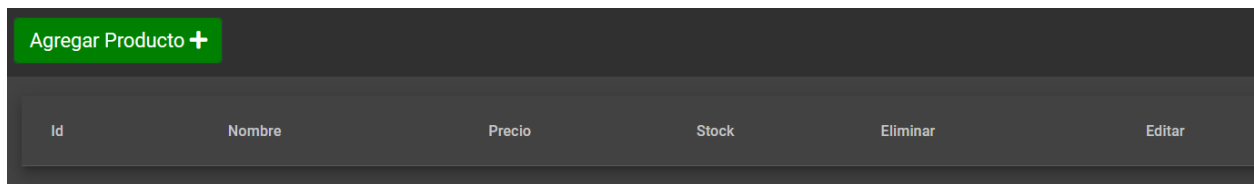
Id	Nombre	Apellido	Eliminar	Editar
----	--------	----------	----------	--------

*Ilustración 63: Formulario para Agregar Empleado y lista de registros en tabla*



Id	Nombre	Ciudad	Número Telefónico	Eliminar	Editar
----	--------	--------	-------------------	----------	--------

*Ilustración 64: Formulario para Agregar Cliente y lista de registros en tabla*

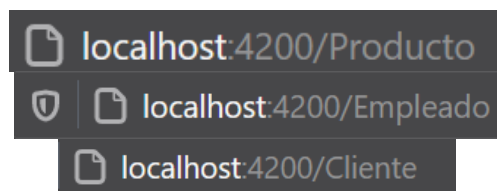


Id	Nombre	Precio	Stock	Eliminar	Editar
----	--------	--------	-------	----------	--------

*Ilustración 65: Formulario para Agregar Producto y lista de registros en tabla*

El archivo de routing tiene configuraciones para que al dar clic en las diferentes pestañas nos lleve a las diferentes vistas.

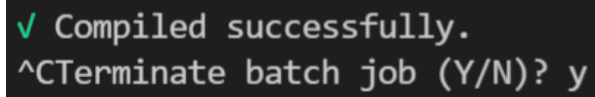
Por ello, dependiendo en cuál nos encontremos, veremos que la url se modifica respectivamente:



*Ilustración 66: URL's de las tablas*

Cada uno de los empleados, clientes y productos cuentan con sus tablas y su respectiva información.

Para detener la aplicación, abrimos la consola y presionamos CTRL+C, escribimos Y y se detendrá.



```
✓ Compiled successfully.  
^CTerminate batch job (Y/N)? y
```

*Ilustración 67: Ejecución de la aplicación finalizada*

Si se desea volver a correr, escribimos de nuevo ng serve.