Se desea desarrollar una **API REST en ASP.NET Core** que permita consultar información de empleados y productos de la empresa.

Para ello, deberás crear un **controlador llamado EmpleadosController** que exponga distintos endpoints GET para consultar información almacenada en la base de datos mediante un repositorio (\_repository).

El sistema debe cumplir con los siguientes requisitos:

1. **Consultar empleados**
   * GET api/TodosLosEmpleados: Devuelve la lista completa de empleados.
   * GET api/CantidadEmpleados: Devuelve el número total de empleados en la empresa.
   * GET api/EmpleadoPorID?empleadoID=5: Devuelve la información de un empleado a partir de su ID.
   * GET api/EmpleadosPorNombre?nombreEmpleado=””: Devuelve el empleado cuyo nombre coincida con el valor ingresado.
   * GET api/IDempleadoPorTitulo?titulo=Manager: Devuelve el empleado que ocupa el puesto indicado.
   * GET api/EmpleadoPorPais?country=””: Devuelve un empleado que viva en el país especificado.
   * GET api/TodosLosEmpleadosPorPais?country=””: Devuelve todos los empleados de un país determinado.
   * GET api/ElEmpleadoMasGrande: Devuelve el empleado de mayor edad.
2. **Estadísticas de empleados**
   * GET api/CantidadEmpleadosPorTitulos: Devuelve una lista con la cantidad de empleados agrupados por cada título (cargo/puesto).
3. **Productos**
   * GET api/ObtenerProductosConCategoria: Devuelve una lista de productos junto con la categoría a la que pertenecen.
   * GET api/ObtenerProductosQueContienen?palabra=””: Devuelve todos los productos cuyo nombre contenga la palabra indicada.

⚡ **Notas técnicas:**

* Todos los métodos deben ser asíncronos (async) y retornar los tipos adecuados (List<Employees>, Employees, int, etc.).
* Los parámetros deben recibirse mediante [FromQuery].
* Se utilizará un repositorio inyectado en el controlador (\_repository) para realizar las consultas.

# Apunte NorthwindRepository.cs

## Descripción general

Esta clase implementa el patrón **Repositorio**, encargándose de acceder a los datos del contexto DataContext (que representa la base de datos Northwind).  
Cada método usa **consultas LINQ (Language Integrated Query)** y los métodos **asíncronos de Entity Framework Core (EF Core)** para comunicarse con SQL Server.

## ⚙️ Flujo general de cada método

### 1. ObtenerTodosLosEmpleados()

**Objetivo:** devuelve una lista con todos los empleados.  
**Flujo:**

* Accede al DbSet<Employee> (\_context.Employees)
* Convierte el resultado en una lista asíncrona con ToListAsync()

return await \_context.Employees.ToListAsync();

🔹 **Método usado:**

* ToListAsync() → ejecuta la consulta en la BD y devuelve una lista (List<T>)

### 2. ObtenerlaCantidadDeEmpleados()

**Objetivo:** devuelve el número total de empleados.  
**Flujo:**

* Usa CountAsync() sobre la tabla Employees para contar los registros.

return await \_context.Employees.CountAsync();

🔹 **Método usado:**

* CountAsync() → ejecuta un SELECT COUNT(\*) en SQL.

### 3. ObtenerEmpleadoporId(int id)

**Objetivo:** busca un empleado por su ID.  
**Flujo:**

* Aplica un filtro Where() para que coincida el EmployeeID.
* Usa FirstOrDefaultAsync() para traer el primero o null si no hay coincidencia.

return await \_context.Employees

.Where(e => e.EmployeeID == id)

.FirstOrDefaultAsync();

🔹 **Métodos usados:**

* Where() → filtra elementos que cumplen una condición (WHERE en SQL).
* FirstOrDefaultAsync() → devuelve el primer elemento o null.

### 4. ObtenerEmpleadoporNombre(string nombre)

**Objetivo:** busca un empleado por su nombre.  
**Flujo:**

* Usa directamente FirstOrDefaultAsync() sin Where() (equivalente a WHERE + TOP 1).

return await \_context.Employees

.FirstOrDefaultAsync(e => e.FirstName == nombre);

### 5. ObtenerEmpleadoporTitulo(string titulo)

**Objetivo:** devuelve el ID del primer empleado que tenga ese título.  
**Flujo:**

* Usa consulta LINQ (from … in … where … select …)
* Devuelve el primer EmployeeID encontrado.

var result = from emp in \_context.Employees

where emp.Title == titulo

select emp.EmployeeID;

return await result.FirstAsync();

🔹 **Método usado:**

* FirstAsync() → igual que FirstOrDefaultAsync() pero lanza excepción si no hay resultados.

### 6. ObtenerEmpleadoporCountry(string country)

**Objetivo:** obtener el primer empleado de un país específico.  
**Flujo:**

* Filtra con where
* Proyecta solo algunos campos (FirstName, LastName, Country)
* Devuelve el primer resultado

var result = from emp in \_context.Employees

where emp.Country == country

select new Employee

{

FirstName = emp.FirstName,

LastName = emp.LastName,

Country = emp.Country

};

return await result.FirstOrDefaultAsync();

🔹 **Concepto:**  
select new ... crea un nuevo objeto solo con los campos necesarios (proyección).

### 7. ObtenerEmpleadosPorTitulos(string titulo)

**Objetivo:** obtener todos los empleados con un título específico.  
**Flujo:**

* Filtra por Title
* Ordena por FirstName
* Devuelve lista

var result = from emp in \_context.Employees

where emp.Title == titulo

orderby emp.FirstName

select emp;

return await result.ToListAsync();

🔹 **Métodos usados:**

* orderby → ordena resultados (ORDER BY en SQL)
* ToListAsync() → ejecuta la consulta.

### 8. ObtenerEmpleadoMasGrande()

**Objetivo:** obtener el empleado de mayor edad.  
**Flujo:**

* Ordena por BirthDate descendente
* Devuelve el primero (el más antiguo)

var result = from emp in \_context.Employees

orderby emp.BirthDate descending

select emp;

return await result.FirstOrDefaultAsync();

### 9. ObtenerProductosQueContienen(string palabra)

**Objetivo:** buscar productos cuyo nombre contenga una palabra.  
**Flujo:**

* Usa EF.Functions.Like() con comodines %palabra%
* Devuelve lista

return await \_context.Products

.Where(p => EF.Functions.Like(p.ProductName, $"%{palabra}%"))

.ToListAsync();

🔹 **Método usado:**

* EF.Functions.Like() → permite usar la función SQL LIKE.

### 10. ObtenerProductosConCategoria()

**Objetivo:** devolver lista de productos junto con el nombre de su categoría.  
**Flujo:**

* Hace un JOIN entre Products y Categories
* Proyecta un DTO (ProductWithCategoryDTO)
* Devuelve lista

var result = await (from prod in \_context.Products

join cat in \_context.Categories

on prod.CategoryID equals cat.CategoryID

select new ProductWithCategoryDTO

{

ProductID = prod.ProductID,

ProductName = prod.ProductName,

CategoryName = cat.CategoryName

}).ToListAsync();

return result;

🔹 **Conceptos:**

* join ... on ... equals ... → une dos tablas (como INNER JOIN)
* select new → crea un nuevo objeto combinando campos de ambas tablas.

## Métodos clave de EF Core (resumen general)

| Método / Expresión | Descripción | Equivalente SQL |
| --- | --- | --- |
| Where() | Filtra registros por una condición | WHERE |
| FirstOrDefaultAsync() | Devuelve el primer elemento o null si no hay | TOP 1 + WHERE |
| FirstAsync() | Devuelve el primer elemento o lanza excepción si no hay | TOP 1 |
| CountAsync() | Devuelve el total de registros | COUNT(\*) |
| ToListAsync() | Ejecuta la consulta y devuelve una lista | Ejecuta SELECT |
| orderby | Ordena resultados | ORDER BY |
| join … equals … | Une dos tablas | INNER JOIN |
| EF.Functions.Like() | Busca coincidencias parciales | LIKE '%texto%' |