Proyecto Backend C#

**Por Carlos Arturo Castro Castro**

**Paso 1: Configuración Inicial del Proyecto**

1. **Abre Visual Studio Code**:
   * Inicia Visual Studio Code en tu computadora.
2. **Crea una Nueva Carpeta para el Proyecto**:
   * En el explorador de archivos, crea una nueva carpeta en la ubicación que prefieras. Nombrarla algo como ProyectoBackend.
3. **Abre la Carpeta en VS Code**:
   * En VS Code, selecciona File > Open Folder... y elige la carpeta que acabas de crear.
4. **Inicializa un Nuevo Proyecto .NET**:
   * Abre la terminal integrada en VS Code (Terminal > New Terminal) y asegúrate de estar en la carpeta del proyecto.
   * Ejecuta el siguiente comando para crear una nueva aplicación de consola .NET:

|  |
| --- |
| dotnet new webapi -n ProyectoBackend |

*  Esto creará la estructura básica del proyecto en la carpeta ProyectoBackend.

 **Navega a la Carpeta del Proyecto**:

* Cambia a la carpeta del proyecto

|  |
| --- |
| cd ProyectoBackend |

**Paso 2: Inicializar un Repositorio Git**

1. **Inicializa Git en el Proyecto**:
   * Dentro de la carpeta ProyectoBackend, inicializa un repositorio Git:

|  |
| --- |
| git init |

**Crea un archivo .gitignore**:

* Ejecuta el siguiente comando para crear un archivo .gitignore adecuado para proyectos .NET:

|  |
| --- |
| dotnet new gitignore |

**Agrega los Archivos al Repositorio**:

* Agrega todos los archivos del proyecto al área de preparación de Git:

|  |
| --- |
| git add . |

**Haz el Primer Commit**:

* Realiza el primer commit para guardar el estado inicial del proyecto:

|  |
| --- |
| git commit -m "Initial commit - setup .NET Web API project" |

**Paso 3: Crear el Primer Archivo del Modelo**

1. **Crea las Carpetas Necesarias**:
   * En la carpeta ProyectoBackend, crea una carpeta llamada Models.
2. **Crear el Archivo del Modelo**:
   * Dentro de la carpeta Models, crea un nuevo archivo llamado Entidad.cs.
3. **Define el Modelo**:
   * Agrega el siguiente código en Entidad.cs para definir una entidad para el proyecto

|  |
| --- |
| #nullable enable  using System.Collections.Generic;  namespace ProyectoBackend.Models  {  public class Entidad  {  private Dictionary<string, object?> propiedades;  public Entidad()  {  propiedades = new Dictionary<string, object?>();  }  public Entidad(Dictionary<string, object?> initialProperties)  {  propiedades = initialProperties ?? new Dictionary<string, object?>();  }  public object? this[string nombre]  {  get  {  if (propiedades.TryGetValue(nombre, out var value))  {  return value;  }  return null;  }  set  {  propiedades[nombre] = value;  }  }  public Dictionary<string, object?> ObtenerPropiedades()  {  return new Dictionary<string, object?>(propiedades);  }  }  } |

**Paso 4: Realizar un Commit con el Modelo**

1. **Agrega el Archivo al Área de Preparación**:
   * Agrega el nuevo archivo Entidad.cs al área de preparación de Git

|  |
| --- |
| git add Models/Entidad.cs |

**Haz un Commit con el Modelo**:

* Realiza un commit para guardar el archivo del modelo en el repositorio:

|  |
| --- |
| git commit -m "Add Entidad model" |

**Paso 5: Subir el Proyecto a GitHub**

1. **Crea un Nuevo Repositorio en GitHub**:
   * Ve a [GitHub](https://github.com/) y crea un nuevo repositorio. Dale un nombre, por ejemplo, ProyectoBackend.
2. **Vincula tu Repositorio Local con GitHub**:
   * En la terminal de VS Code, vincula tu repositorio local con el repositorio remoto en GitHub:

(Reemplaza tu-usuario con tu nombre de usuario de GitHub y ProyectoBackend con el nombre de tu repositorio).

|  |
| --- |
| git remote add origin https://github.com/tu-usuario/ProyectoBackend.git |

**Sube el Proyecto a GitHub**:

* Sube tus commits al repositorio de GitHub:

|  |
| --- |
| git push -u origin master |

**Actualizar el Código en ControlConexion.cs**

1. **Crea o abre el archivo ControlConexion.cs en la carpeta Services**:
   * Si no lo has hecho ya, crea una carpeta llamada Services en tu proyecto y dentro de esta, crea el archivo ControlConexion.cs.
2. **Agrega el código simplificado**:
   * Copia y pega el código proporcionado en el archivo ControlConexion.cs.

**Paso 2: Realizar un Commit**

1. **Verifica el estado del repositorio**:
   * Abre la terminal integrada de VS Code y navega a la carpeta del proyecto si no lo estás ya:

|  |
| --- |
| cd ProyectoBackend |

Verifica el estado de tu repositorio para ver los cambios pendientes:

|  |
| --- |
| git status |

**Agrega el archivo al área de preparación**:

* Agrega el archivo ControlConexion.cs al área de preparación:

|  |
| --- |
| git add Services/ControlConexion.cs |

**Realiza un Commit**:

* Haz un commit con un mensaje que describa claramente el cambio:

|  |
| --- |
| git commit -m "Simplify ControlConexion for LocalDB and SQL Server support only" |

**Sube el Cambio a GitHub**:

* Sube el commit al repositorio remoto en GitHub:

|  |
| --- |
| git push origin master |

**Agregar la referencia al paquete Microsoft.Data.SqlClient**:

* Abre la terminal en Visual Studio Code y asegúrate de estar en la carpeta del proyecto (ProyectoBackend).
* Ejecuta el siguiente comando para instalar el paquete Microsoft.Data.SqlClient:

|  |
| --- |
| dotnet add package Microsoft.Data.SqlClient |

**Restaurar los paquetes**:

* Luego de agregar el paquete, restaura los paquetes del proyecto para asegurarte de que todo esté correctamente instalado:

|  |
| --- |
| dotnet restore |

**Verifica el Código**:

* Después de instalar el paquete y restaurar los paquetes, verifica si los errores desaparecen. Las clases como SqlConnection, SqlCommand, y SqlParameter deberían estar disponibles ahora.

|  |
| --- |
|  |

**Realizar un Commit**:

* Si todo funciona correctamente, realiza un commit para guardar este cambio

|  |
| --- |
| git add ProyectoBackend.csproj  git commit -m "Add Microsoft.Data.SqlClient package for SQL Server support"  git push origin master |

EntidadesController.cs

|  |
| --- |
| #nullable enable  using System;  using System.Collections.Generic;  using System.Data;  using System.Data.Common;  using Microsoft.AspNetCore.Authorization;  using Microsoft.AspNetCore.Mvc;  using Microsoft.Extensions.Configuration;  using Microsoft.Data.SqlClient;  using System.Linq;  using System.Text.Json;  using ProyectoBackend.Models;  using ProyectoBackend.Services;  using BCrypt.Net;  namespace ProyectoBackend.Controllers  {      [Route("api/{projectName}/{tableName}")]      [ApiController]      [Authorize]      public class EntidadesController : ControllerBase      {          private readonly ControlConexion controlConexion;          private readonly IConfiguration \_configuration;          public EntidadesController(ControlConexion controlConexion, IConfiguration configuration)          {              this.controlConexion = controlConexion ?? throw new ArgumentNullException(nameof(controlConexion));              \_configuration = configuration ?? throw new ArgumentNullException(nameof(configuration));          }          [AllowAnonymous]          [HttpGet]          public IActionResult Listar(string projectName, string tableName)          {              if (string.IsNullOrWhiteSpace(tableName))                  return BadRequest("El nombre de la tabla no puede estar vacío.");              try              {                  var lista = new List<Dictionary<string, object?>>();                  string comandoSQL = $"SELECT \* FROM {tableName}";                  controlConexion.AbrirBd();                  var tabla = controlConexion.EjecutarConsultaSql(comandoSQL, null);                  controlConexion.CerrarBd();                  foreach (DataRow fila in tabla.Rows)                  {                      var propiedades = fila.Table.Columns.Cast<DataColumn>()                                          .ToDictionary(col => col.ColumnName, col => fila[col] == DBNull.Value ? null : fila[col]);                      lista.Add(propiedades);                  }                  return Ok(lista);              }              catch (Exception ex)              {                  return StatusCode(500, $"Error interno del servidor: {ex.Message}");              }          }          [AllowAnonymous]          [HttpGet("{keyName}/{value}")]          public IActionResult GetByKey(string projectName, string tableName, string keyName, string value)          {              if (string.IsNullOrWhiteSpace(tableName) || string.IsNullOrWhiteSpace(keyName) || string.IsNullOrWhiteSpace(value))                  return BadRequest("El nombre de la tabla, el nombre de la clave y el valor no pueden estar vacíos.");              controlConexion.AbrirBd();              try              {                  string query = "SELECT data\_type FROM information\_schema.columns WHERE table\_name = @tableName AND column\_name = @columnName";                  var parameters = new DbParameter[]                  {                      CreateParameter("@tableName", tableName),                      CreateParameter("@columnName", keyName)                  };                  var dataTypeResult = controlConexion.EjecutarConsultaSql(query, parameters);                  if (dataTypeResult == null || dataTypeResult.Rows.Count == 0 || dataTypeResult.Rows[0]["data\_type"] == DBNull.Value)                      return NotFound("No se pudo determinar el tipo de dato.");                  string dataType = dataTypeResult.Rows[0]["data\_type"]?.ToString() ?? "";                  object convertedValue;                  string comandoSQL;                  switch (dataType.ToLower())                  {                      case "int":                      case "bigint":                      case "integer":                          if (int.TryParse(value, out int intValue))                          {                              convertedValue = intValue;                              comandoSQL = $"SELECT \* FROM {tableName} WHERE {keyName} = @Value";                          }                          else                          {                              return BadRequest("El valor proporcionado no es válido para el tipo de datos entero.");                          }                          break;                      case "decimal":                      case "numeric":                          if (decimal.TryParse(value, out decimal decimalValue))                          {                              convertedValue = decimalValue;                              comandoSQL = $"SELECT \* FROM {tableName} WHERE {keyName} = @Value";                          }                          else                          {                              return BadRequest("El valor proporcionado no es válido para el tipo de datos número.");                          }                          break;                      case "bit":                      case "boolean":                          if (bool.TryParse(value, out bool boolValue))                          {                              convertedValue = boolValue;                              comandoSQL = $"SELECT \* FROM {tableName} WHERE {keyName} = @Value";                          }                          else                          {                              return BadRequest("El valor proporcionado no es válido para el tipo de datos booleano.");                          }                          break;                      case "float":                      case "real":                          if (double.TryParse(value, out double doubleValue))                          {                              convertedValue = doubleValue;                              comandoSQL = $"SELECT \* FROM {tableName} WHERE {keyName} = @Value";                          }                          else                          {                              return BadRequest("El valor proporcionado no es válido para el tipo de datos doble.");                          }                          break;                      case "nvarchar":                      case "varchar":                      case "character varying":                      case "text":                      case "char":                          convertedValue = value;                          comandoSQL = $"SELECT \* FROM {tableName} WHERE {keyName} = @Value";                          break;                      case "date":                      case "datetime":                          if (DateTime.TryParse(value, out DateTime dateValue))                          {                              comandoSQL = $"SELECT \* FROM {tableName} WHERE CAST({keyName} AS DATE) = @Value";                              convertedValue = dateValue.Date;                          }                          else                          {                              return BadRequest("El valor proporcionado no es válido para el tipo de datos fecha.");                          }                          break;                      default:                          return BadRequest($"Tipo de dato no soportado: {dataType}");                  }                  var parametro = CreateParameter("@Value", convertedValue);                  var resultado = controlConexion.EjecutarConsultaSql(comandoSQL, new DbParameter[] { parametro });                  if (resultado.Rows.Count > 0)                  {                      var lista = new List<Dictionary<string, object?>>();                      foreach (DataRow fila in resultado.Rows)                      {                          var propiedades = resultado.Columns.Cast<DataColumn>()                                             .ToDictionary(col => col.ColumnName, col => fila[col] == DBNull.Value ? null : fila[col]);                          lista.Add(propiedades);                      }                      return Ok(lista);                  }                  return NotFound();              }              catch (Exception ex)              {                  return StatusCode(500, $"Error interno del servidor: {ex.Message}");              }              finally              {                  controlConexion.CerrarBd();              }          }          [AllowAnonymous]          [HttpPost]          public IActionResult Crear(string projectName, string tableName, [FromBody] Dictionary<string, object?> entidadData)          {              if (string.IsNullOrWhiteSpace(tableName) || entidadData == null || !entidadData.Any())                  return BadRequest("El nombre de la tabla y los datos de la entidad no pueden estar vacíos.");              try              {                  var propiedades = entidadData.ToDictionary(                      kvp => kvp.Key,                      kvp => kvp.Value is JsonElement jsonElement ? ConvertJsonElement(jsonElement) : kvp.Value);                  // Encriptación de contraseñas si se detectan                  var passwordKeys = new[] { "password", "contrasena", "passw" };                  var passwordKey = propiedades.Keys.FirstOrDefault(k => passwordKeys.Any(pk => k.IndexOf(pk, StringComparison.OrdinalIgnoreCase) >= 0));                    if (passwordKey != null)                  {                      var plainPassword = propiedades[passwordKey]?.ToString();                      if (!string.IsNullOrEmpty(plainPassword))                      {                          propiedades[passwordKey] = BCrypt.Net.BCrypt.HashPassword(plainPassword);                      }                  }                  var columnas = string.Join(",", propiedades.Keys);                  var valores = string.Join(",", propiedades.Keys.Select(k => $"@{k}"));                  string comandoSQL = $"INSERT INTO {tableName} ({columnas}) VALUES ({valores})";                  var parametros = propiedades.Select(p => CreateParameter($"@{p.Key}", p.Value)).ToArray();                  controlConexion.AbrirBd();                  controlConexion.EjecutarComandoSql(comandoSQL, parametros);                  controlConexion.CerrarBd();                  return Ok("Entidad creada exitosamente.");              }              catch (Exception ex)              {                  return StatusCode(500, $"Error interno del servidor: {ex.Message}");              }          }          [AllowAnonymous]          [HttpPut("{keyName}/{keyValue}")]          public IActionResult Actualizar(string projectName, string tableName, string keyName, string keyValue, [FromBody] Dictionary<string, object?> entidadData)          {              if (string.IsNullOrWhiteSpace(tableName) || string.IsNullOrWhiteSpace(keyName) || entidadData == null || !entidadData.Any())                  return BadRequest("El nombre de la tabla, el nombre de la clave y los datos de la entidad no pueden estar vacíos.");              try              {                  var propiedades = entidadData.ToDictionary(                      kvp => kvp.Key,                      kvp => kvp.Value is JsonElement jsonElement ? ConvertJsonElement(jsonElement) : kvp.Value);                  // Encriptación de contraseñas si se detectan                  var passwordKeys = new[] { "password", "contrasena", "passw" };                  var passwordKey = propiedades.Keys.FirstOrDefault(k => passwordKeys.Any(pk => k.IndexOf(pk, StringComparison.OrdinalIgnoreCase) >= 0));                    if (passwordKey != null)                  {                      var plainPassword = propiedades[passwordKey]?.ToString();                      if (!string.IsNullOrEmpty(plainPassword))                      {                          propiedades[passwordKey] = BCrypt.Net.BCrypt.HashPassword(plainPassword);                      }                  }                  var actualizaciones = string.Join(",", propiedades.Select(p => $"{p.Key}=@{p.Key}"));                  string comandoSQL = $"UPDATE {tableName} SET {actualizaciones} WHERE {keyName}=@KeyValue";                  var parametros = propiedades.Select(p => CreateParameter($"@{p.Key}", p.Value)).ToList();                  parametros.Add(CreateParameter("@KeyValue", keyValue));                  controlConexion.AbrirBd();                  controlConexion.EjecutarComandoSql(comandoSQL, parametros.ToArray());                  controlConexion.CerrarBd();                  return Ok("Entidad actualizada exitosamente.");              }              catch (Exception ex)              {                  return StatusCode(500, $"Error interno del servidor: {ex.Message}");              }          }          [AllowAnonymous]          [HttpDelete("{keyName}/{keyValue}")]          public IActionResult Eliminar(string projectName, string tableName, string keyName, string keyValue)          {              if (string.IsNullOrWhiteSpace(tableName) || string.IsNullOrWhiteSpace(keyName))                  return BadRequest("El nombre de la tabla o el nombre de la clave no pueden estar vacíos.");              try              {                  string comandoSQL = $"DELETE FROM {tableName} WHERE {keyName}=@KeyValue";                  var parametro = CreateParameter("@KeyValue", keyValue);                  controlConexion.AbrirBd();                  controlConexion.EjecutarComandoSql(comandoSQL, new[] { parametro });                  controlConexion.CerrarBd();                  return Ok("Entidad eliminada exitosamente.");              }              catch (Exception ex)              {                  return StatusCode(500, $"Error interno del servidor: {ex.Message}");              }          }          [AllowAnonymous]          [HttpPost("verificar-contrasena")]          public IActionResult VerificarContrasena(string projectName, string tableName, [FromBody] Dictionary<string, string> datos)          {              if (string.IsNullOrWhiteSpace(tableName) || datos == null || !datos.ContainsKey("userField") || !datos.ContainsKey("passwordField") || !datos.ContainsKey("userValue") || !datos.ContainsKey("passwordValue"))                  return BadRequest("El nombre de la tabla, el campo de usuario, el campo de contraseña, el valor de usuario y el valor de contraseña no pueden estar vacíos.");              try              {                  string userField = datos["userField"];                  string passwordField = datos["passwordField"];                  string userValue = datos["userValue"];                  string passwordValue = datos["passwordValue"];                  string comandoSQL = $"SELECT {passwordField} FROM {tableName} WHERE {userField} = @UserValue";                  var parametro = CreateParameter("@UserValue", userValue);                  controlConexion.AbrirBd();                  var resultado = controlConexion.EjecutarConsultaSql(comandoSQL, new DbParameter[] { parametro });                  controlConexion.CerrarBd();                  if (resultado.Rows.Count == 0)                      return NotFound("Usuario no encontrado.");                  string hashedPassword = resultado.Rows[0][passwordField]?.ToString() ?? string.Empty;                  if (!hashedPassword.StartsWith("$2"))                      throw new InvalidOperationException("Stored password hash is not a valid BCrypt hash.");                  bool isPasswordValid = BCrypt.Net.BCrypt.Verify(passwordValue, hashedPassword);                  if (isPasswordValid)                      return Ok("Contraseña verificada exitosamente.");                  else                      return Unauthorized("Contraseña incorrecta.");              }              catch (Exception ex)              {                  return StatusCode(500, $"Error interno del servidor: {ex.Message}");              }          }          public DbParameter CreateParameter(string name, object? value)          {              return new SqlParameter(name, value ?? DBNull.Value);          }          private object? ConvertJsonElement(JsonElement jsonElement)          {              if (jsonElement.ValueKind == JsonValueKind.Null)                  return null;              switch (jsonElement.ValueKind)              {                  case JsonValueKind.String:                      return DateTime.TryParse(jsonElement.GetString(), out DateTime dateValue) ? (object)dateValue : jsonElement.GetString();                  case JsonValueKind.Number:                      return jsonElement.TryGetInt32(out var intValue) ? (object)intValue : jsonElement.GetDouble();                  case JsonValueKind.True:                      return true;                  case JsonValueKind.False:                      return false;                  case JsonValueKind.Object:                      return jsonElement.GetRawText();                  case JsonValueKind.Array:                      return jsonElement.GetRawText();                  default:                      throw new InvalidOperationException($"Unsupported JsonValueKind: {jsonElement.ValueKind}");              }          }      }  } |

**ntegración del Código**

1. **Crear o Modificar el Archivo**:
   * Crea un archivo EntidadesController.cs dentro de la carpeta Controllers en tu proyecto.
   * Copia y pega el código simplificado en este archivo.

**Paso 2: Realizar un Commit**

1. **Agrega el archivo al área de preparación**:
   * Abre la terminal en VS Code y asegúrate de estar en la carpeta del proyecto.
   * Luego, agrega el archivo EntidadesController.cs al área de preparación:

|  |
| --- |
| git add Controllers/EntidadesController.cs |

**Realiza un Commit**:

* Haz un commit con un mensaje descriptivo

|  |
| --- |
| git commit -m "Add EntidadesController for LocalDB and SQL Server support only" |

**Sube el Cambio a GitHub**:

* Sube el commit al repositorio remoto en GitHub:

|  |
| --- |
| git push origin master |

**Referencia a BCrypt**

El error indica que BCrypt no se encuentra. Esto suele deberse a que falta el paquete correspondiente.

1. **Instala el Paquete BCrypt.Net-Next**:
   * Abre la terminal integrada en Visual Studio Code y ejecuta el siguiente comando para instalar el paquete:

|  |
| --- |
| dotnet add package BCrypt.Net-Next |

**Restaura los Paquetes**:

* Luego, ejecuta:

|  |
| --- |
| dotnet restore |

**Asegúrate de que el Namespace se Esté Usando**:

* En el archivo EntidadesController.cs, asegúrate de que estás usando el namespace correcto:

|  |
| --- |
| using BCrypt.Net; |

**Recompila y Verifica**

1. **Recompila el Proyecto**:
   * Después de realizar las correcciones, recompila el proyecto para asegurarte de que todos los errores se han solucionado.
2. **Realiza un Commit**:
   * Si todo funciona correctamente, realiza un commit para guardar estos cambios:

|  |
| --- |
| git add .  git commit -m "Fix namespaces and add BCrypt.Net package"  git push origin master |

**Commit de los Cambios:**

Una vez que hayas verificado y corregido el problema, realiza un commit para guardar los cambios en Git:

1. **Agregar los Cambios**:

|  |
| --- |
| git add . |

**Realizar el Commit**:

En caso de corrección de errores por ejemplo

|  |
| --- |
| git commit -m "Fix namespace issue for Models in EntidadesController" |

**Subir los Cambios a GitHub**:

|  |
| --- |
| git push origin master |

**Recompila y Verifica**

Después de asegurarte de que los namespaces sean correctos, recompila el proyecto:

1. **Compila el proyecto** desde la terminal de Visual Studio Code:

dotnet build

**4. Realiza un Commit de los Cambios**

Si todo está funcionando bien, guarda los cambios en Git:

1. **Agrega los cambios**:

git add .

1. **Realiza un commit**:

git commit -m "Fix namespaces to match project name ProyectoBackend"

1. **Sube los cambios a GitHub**:

git push origin master

|  |
| --- |
|  |

 **ecompila el Proyecto**:

* Después de realizar estos cambios, recompila el proyecto para asegurarte de que todo esté en orden:

dotnet build

 **Realiza un Commit de los Cambios**:

* Si todo está funcionando correctamente, realiza un commit para guardar los cambios:

git add .

git commit -m "Fix namespace issues for Models in ProyectoBackend"

git push origin master

launchsettings.json

|  |
| --- |
| {    "$schema": "http://json.schemastore.org/launchsettings.json",    "iisSettings": {      "windowsAuthentication": false,      "anonymousAuthentication": true,      "iisExpress": {        "applicationUrl": "http://localhost:63330",        "sslPort": 44371      }    },    "profiles": {      "http": {        "commandName": "Project",        "dotnetRunMessages": true,        "launchBrowser": true,        "launchUrl": "swagger",        "applicationUrl": "http://localhost:5000",        "environmentVariables": {          "ASPNETCORE\_ENVIRONMENT": "Development"        }      },      "https": {        "commandName": "Project",        "dotnetRunMessages": true,        "launchBrowser": true,        "launchUrl": "swagger",        "applicationUrl": "https://localhost:5001",        "environmentVariables": {          "ASPNETCORE\_ENVIRONMENT": "Development"        }      },      "IIS Express": {        "commandName": "IISExpress",        "launchBrowser": true,        "launchUrl": "swagger",        "environmentVariables": {          "ASPNETCORE\_ENVIRONMENT": "Development"        }      }    }  } |

1. **Si este archivo ya está bien configurado** para tus necesidades, no necesitas hacer más cambios. Simplemente puedes utilizar este archivo launchsettings.json tal como está.
2. **Si realizaste algún cambio**, no olvides hacer un commit para guardar estos cambios:

git add Properties/launchSettings.json

git commit -m "Update launchsettings.json for generic development configuration"

git push origin master

Este archivo de configuración debería permitirte iniciar la aplicación fácilmente en diferentes entornos con configuraciones que son tanto seguras como flexibles para el desarrollo local.

appsettings.json

|  |
| --- |
| {    "Jwt": {      "Key": "MySuperSecretKey1234567890!@#$%^&\*()\_+",      "Issuer": "MyApp",      "Audience": "MyAppUsers"    },    "ConnectionStrings": {      "LocalDb": "Data Source=(localdb)\\MSSQLLocalDB;AttachDbFilename=|DataDirectory|\\bd\_indicadores\_1330.mdf;Integrated Security=True",      "SqlServer": "Server=myServerAddress;Database=myDataBase;User Id=myUsername;Password=myPassword;"    },    "DatabaseProvider": "SqlServer"  } |

1. **Clave JWT**:
   * La clave para JWT ("Key") y los valores para "Issuer" y "Audience" permanecen, ya que son esenciales para la autenticación y seguridad en tu API.

**Próximos Pasos:**

1. **Asegúrate de que este archivo cumpla con tus requisitos** y realiza las modificaciones adicionales si es necesario.
2. **Si realizaste cambios en el archivo**, guarda y haz un commit:

git add appsettings.json

git commit -m "Simplify appsettings.json for LocalDB and SQL Server only"

git push origin master

Este archivo de configuración ahora está alineado para trabajar exclusivamente con LocalDB y SQL Server, con una configuración de JWT básica que puedes ajustar según sea necesario.

Program.cs

|  |
| --- |
| var builder = WebApplication.CreateBuilder(args);  // Add services to the container.  builder.Services.AddControllers();  builder.Services.AddEndpointsApiExplorer();  builder.Services.AddSwaggerGen();  string jwtKey = builder.Configuration["Jwt:Key"] ?? throw new ArgumentNullException("Jwt:Key", "JWT Key cannot be null");  string jwtIssuer = builder.Configuration["Jwt:Issuer"] ?? throw new ArgumentNullException("Jwt:Issuer", "JWT Issuer cannot be null");  string jwtAudience = builder.Configuration["Jwt:Audience"] ?? throw new ArgumentNullException("Jwt:Audience", "JWT Audience cannot be null");  builder.Services.AddAuthentication("Bearer")      .AddJwtBearer("Bearer", options =>      {          options.TokenValidationParameters = new Microsoft.IdentityModel.Tokens.TokenValidationParameters          {              ValidateIssuer = true,              ValidateAudience = true,              ValidateLifetime = true,              ValidateIssuerSigningKey = true,              ValidIssuer = jwtIssuer,              ValidAudience = jwtAudience,              IssuerSigningKey = new Microsoft.IdentityModel.Tokens.SymmetricSecurityKey(System.Text.Encoding.UTF8.GetBytes(jwtKey))          };      });  builder.Services.AddAuthorization();  var app = builder.Build();  if (app.Environment.IsDevelopment())  {      app.UseSwagger();      app.UseSwaggerUI();  }  app.UseHttpsRedirection();  app.UseAuthentication();  app.UseAuthorization();  app.MapControllers();  // Agrega un endpoint en la raíz para manejar solicitudes a "/"  app.MapGet("/", () => "API is running!");  app.Run(); |

**Próximos Pasos:**

1. **Si estás satisfecho con esta configuración** para Program.cs, guárdalo y realiza un commit:

git add Program.cs

git commit -m "Simplify and enhance Program.cs for API with LocalDB and SQL Server"

git push origin master

1. **Asegúrate de que todo esté bien configurado**:
   * Verifica que las dependencias necesarias estén instaladas (como Microsoft.IdentityModel.Tokens para JWT).

**Instalar el Paquete Necesario para JWT**:

* Abre la terminal en Visual Studio Code y ejecuta el siguiente comando para instalar el paquete necesario:

|  |
| --- |
| dotnet add package Microsoft.AspNetCore.Authentication.JwtBearer |

**Recompila el Proyecto**:

* + Una vez instaladas las dependencias y corregidos los valores, recompila tu proyecto:

dotnet build

**Finalizar y Realizar Commit:**

Si todo está funcionando después de realizar estos cambios:

**Realiza un Commit**:

* + Agrega los cambios a Git y realiza un commit:

git add .

git commit -m "Fix JWT configuration and add necessary packages"

git push origin master