Actividad 2: Integrando servicios REST en .NET

Recuerda que esta actividad deberá ser completada en la plataforma. Solo tendrás un intento. Una vez que la envíes, se dará por cerrada. Tu puntuación y las respuestas correctas se mostrarán cuando finalice el periodo de entrega de la tarea.

**Descripción de la actividad y pautas de elaboración**

En esta actividad vamos a aprender a integrar servicios REST de terceros utilizando diversos métodos desde el entorno de .NET. Para ello, supondremos que vamos a tener que consumir un servicio REST de un backend externo como es el sistema Pet‑Clinic. Se trata de un proyecto clásico para aprender a programar con el framework Spring de Java y que tiene diversas implementaciones en su [comunidad](https://spring-petclinic.github.io/). Para esta práctica vamos a utilizar la implementación sobre una arquitectura cliente servidor.

Accede al backend en el siguiente enlace:   
<https://github.com/spring-petclinic/spring-petclinic-rest>

Accede al frontend en el siguiente enlace:   
<https://github.com/spring-petclinic/spring-petclinic-angular>

Puedes descargar su código fuente (haciendo un git clone) o utilizar la copia adjunta al enunciado. Para poder tener todo el entorno en local necesitarás:

* Una versión de la máquina virtual de Java, por ejemplo, [OpenJDK11](https://adoptopenjdk.net/).
* [Maven](https://maven.apache.org/download.cgi).
* [Node.js](https://nodejs.org/es/download/).

Primero lanzamos el backend con la instrucción *./mvnw spring-boot:run* (mvnw.cmd en Windows) y comprobamos que los servicios están activos mediante la visualización con swagger en <http://localhost:9966/petclinic/swagger-ui/index.html>.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

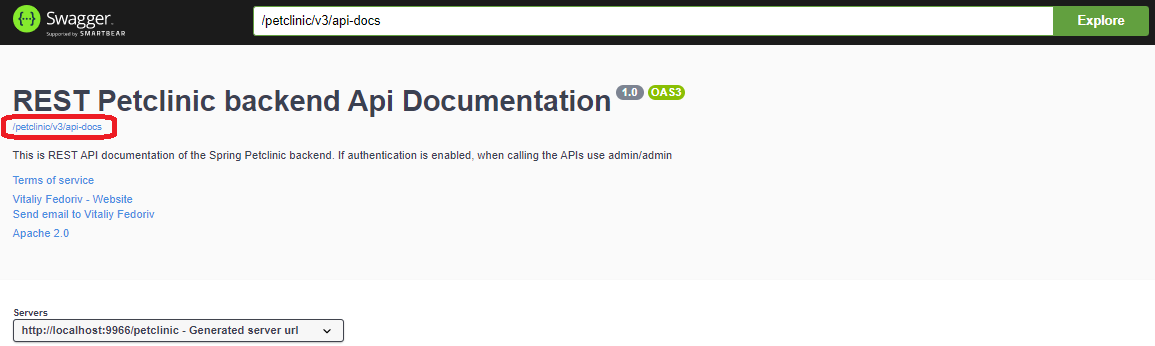
Descripción generada automáticamente

De cara a entender el funcionamiento del sistema puedes lanzar el frontend con *ng serve* y comprobamos que podemos acceder al interfaz gráfico desde <http://localhost:4200/petclinic/welcome>.

Interfaz de usuario gráfica, Sitio web

Descripción generada automáticamente

En esta actividad vamos a intentar consumir esos servicios REST que nos deja accesibles el backend y que puedes acceder desde <http://localhost:9966/petclinic/v3/api-docs> o bien pulsando donde se señala con un recuadro rojo en la siguiente figura, en el que sale el enlace a /petclinic/v3/api-docs:



En este caso, el API está definido conforme al estándar [openAPI](https://swagger.io/specification/). Podemos visualizar su contenido, especificado en formato json, y tendremos el detalle de los servicios disponibles:

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

En esta actividad vamos a ir poniendo en práctica distintas formas de interaccionar con este API REST del backend (el frontend no es necesario tenerlo lanzado) mediante desarrollos en .NET.

1. Crea un proyecto en .NET de tipo consola con el siguiente código:

using System;

using System.Net.Http;

using System.Threading.Tasks;

namespace ConsoleApp

{

internal class Pregunta1

{

static readonly HttpClient client = new HttpClient();

static async Task Main()

{

try

{

using HttpResponseMessage response = await client.GetAsync("http://localhost:9966/petclinic/api/vets");

if (response.IsSuccessStatusCode)

{

string responseBody = await response.Content.ReadAsStringAsync();

Console.WriteLine(responseBody);

}

}

catch (Exception e)

{

Console.WriteLine(e.Message);

}

Console.ReadLine();

}

}

}

Indica cuál es el resultado esperado tras la ejecución:

1. Muestra una excepción.
2. No compila porque hay errores en el código.
3. Muestra en pantalla un json que empieza por [{"firstName":"James","lastName":"Carter".
4. Muestra en pantalla un XML que empieza por <vetList>.
5. Modificamos el programa anterior para poder mapear la respuesta en un modelo de datos propio, de esta forma:

using System;

using System.Net.Http;

using System.Threading.Tasks;

using System.Collections.Generic;

using System.Text.Json;

namespace ConsoleApp

{

internal class Pregunta2

{

static readonly HttpClient client = new HttpClient();

public class VetSpeciality

{

public int id { get; set; }

public string name { get; set; }

}

public class Vet

{

public int id { get; set; }

public string firstName { get; set; }

public string lastName { get; set; }

public List<VetSpeciality> specialties { get; set; }

}

static async Task Main()

{

try

{

using HttpResponseMessage response = await client.GetAsync("http://localhost:9966/petclinic/api/vets");

if (response.IsSuccessStatusCode)

{

string jsonString = await response.Content.ReadAsStringAsync();

List<Vet> vets = JsonSerializer.Deserialize <List<Vet>> (jsonString);

Console.WriteLine(vets.Count);

}

}

catch (Exception e)

{

Console.WriteLine(e.Message);

}

Console.ReadLine();

}

}

}

Indica cuál es el resultado esperado tras la ejecución:

1. Muestra un 4.
2. Muestra el mismo json anterior.
3. Sale una excepción.
4. Muestra un 6.
5. Vamos a hacer lo mismo para las visitas, genera este programa:

using System;

using System.Net.Http;

using System.Threading.Tasks;

using System.Collections.Generic;

using System.Text.Json;

namespace ConsoleApp

{

internal class Pregunta3

{

static readonly HttpClient client = new HttpClient();

public class Visit

{

public int id { get; set; }

public string description { get; set; }

public int petId { get; set; }

public DateTime date { get; set; }

public override string ToString()

{

return "id: " + id+ " Description " + description + " date " + date + " petId " + petId;

}

}

static async Task Main()

{

try

{

using HttpResponseMessage response = await client.GetAsync("http://localhost:9966/petclinic/api/vets");

if (response.IsSuccessStatusCode)

{

string jsonString = await response.Content.ReadAsStringAsync();

List<Visit> visits = JsonSerializer.Deserialize <List<Visit>> (jsonString);

int counter = 0;

foreach (Visit v in visits)

{

Console.WriteLine("Visita " + counter++ + " -> " + v);

}

}

}

catch (Exception e)

{

Console.WriteLine(e.Message);

}

Console.ReadLine();

}

}

}

Indica cuál es el resultado esperado tras la ejecución:

1. Muestra 6 visitas con sus datos.
2. Da un error de compilación.
3. Da un error de ejecución.
4. Muestra las visitas con datos erróneos.
5. Indica qué habría que cambiar en el programa para invocar al servicio correcto:
6. using HttpResponseMessage response = await client.GetAsync("http://localhost:9966/petclinic/api/vets");
7. using HttpResponseMessage response = await client.GetAsync("http://localhost:9966/petclinic/api/visits");
8. using HttpResponseMessage response = await client.GetAsync("vets");
9. using HttpResponseMessage response = await client.GetAsync("visits");
10. Indica cuál es el resultado esperado tras la ejecución:
11. Muestra las cuatro visitas con datos correctos.
12. Muestra las seis visitas con datos correctos.
13. Muestra las cuatro visitas con datos nulos.
14. Da un error de ejecución.
15. Vamos a realizar operaciones con métodos POST que nos permitan insertar o modificar datos. Genera el siguiente programa:

using System;

using System.Net.Http;

using System.Net.Http.Headers;

using System.Net.Http.Json;

using System.Threading.Tasks;

namespace ConsoleApp

{

internal class Pregunta6

{

static readonly HttpClient client = new HttpClient();

public class Speciality

{

public int id { get; set; }

public string name { get; set; }

public override string ToString()

{

return "id: " + id+ " name " + name;

}

}

static async Task<Uri> CreateSpecialityAsync(Speciality speciality)

{

HttpResponseMessage response = await client.PostAsJsonAsync(

"api/specialties", speciality);

response.EnsureSuccessStatusCode();

// return URI of the created resource.

return response.Headers.Location;

}

static async Task Main()

{

try

{

client.BaseAddress = new Uri("http://localhost:9966/petclinic/");

client.DefaultRequestHeaders.Accept.Clear();

client.DefaultRequestHeaders.Accept.Add(

new MediaTypeWithQualityHeaderValue("application/json"));

Speciality speciality = new Speciality()

{

id = 100,

name = "MySpec100"

};

var url = await CreateSpecialityAsync(speciality);

Console.WriteLine($"Created at {url}");

}

catch (Exception e)

{

Console.WriteLine(e.Message);

Console.WriteLine(e.ToString());

}

Console.ReadLine();

}

}

}

Indica cuál es el resultado de su ejecución:

1. Crea una nueva especialidad con código 100 y mostrando un «Created at /api/specialties/100».
2. Da un error de ejecución.
3. Dice que crea una nueva especialidad pero no muestra nada.
4. Nos muestra un «Created at /api/specialties/100», pero en realidad ha creado una nueva especialidad pero le ha asignado un código diferente.
5. Seguimos ampliando las funcionalidades, escribe este código:

using System;

using System.Collections.Generic;

using System.Net.Http;

using System.Net.Http.Headers;

using System.Net.Http.Json;

using System.Text.Json;

using System.Threading.Tasks;

namespace ConsoleApp

{

internal class Pregunta7

{

static readonly HttpClient client = new HttpClient();

public class Speciality

{

public int id { get; set; }

public string name { get; set; }

public override string ToString()

{

return "id: " + id+ " name " + name;

}

}

static async Task<Speciality> CreateSpecialityAsync(Speciality speciality)

{

HttpResponseMessage response = await client.PostAsJsonAsync(

"api/specialties", speciality);

response.EnsureSuccessStatusCode();

string jsonContent = response.Content.ReadAsStringAsync().Result;

Speciality dev = JsonSerializer.Deserialize<Speciality>(jsonContent);

return dev;

}

static async Task<List<Speciality>> GetSpecialitiesAsync()

{

HttpResponseMessage response = await client.GetAsync(

"api/specialties");

response.EnsureSuccessStatusCode();

string jsonString = await response.Content.ReadAsStringAsync();

List<Speciality> specialities = JsonSerializer.Deserialize<List<Speciality>>(jsonString);

return specialities;

}

static async Task<Speciality> GetSpecialityAsync(int id)

{

HttpResponseMessage response = await client.GetAsync(

"api/specialties" + "/" + id);

response.EnsureSuccessStatusCode();

string jsonString = await response.Content.ReadAsStringAsync();

Speciality dev = JsonSerializer.Deserialize<Speciality>(jsonString);

return dev;

}

static Speciality findByName (List<Speciality> specialityList, String name)

{

foreach (Speciality s in specialityList)

{

if (s.name.Equals(name))

return s;

}

return null;

}

static async Task Main()

{

try

{

client.BaseAddress = new Uri("http://localhost:9966/petclinic/");

client.DefaultRequestHeaders.Accept.Clear();

client.DefaultRequestHeaders.Accept.Add(

new MediaTypeWithQualityHeaderValue("application/json"));

Speciality specialityToBeCreated = new Speciality()

{

id = 123,

name = "MySpecialityDemo"

};

// Añadir

Speciality specialityCreated = await CreateSpecialityAsync(specialityToBeCreated);

Console.WriteLine("specialityCreated -> " + specialityCreated);

// Buscar todos

List<Speciality> specialityList = await GetSpecialitiesAsync();

Console.WriteLine("specialityList size -> " + specialityList.Count);

Speciality specialityFounded = findByName(specialityList, "MySpecialityDemo");

Console.WriteLine("specialityFounded -> " + specialityFounded);

// Buscar por id

Speciality specialityFoundedIndividual = await GetSpecialityAsync(specialityFounded.id);

Console.WriteLine("specialityFoundedIndividual -> " + specialityFoundedIndividual);

}

catch (Exception e)

{

Console.WriteLine(e.Message);

Console.WriteLine(e.ToString());

}

Console.ReadLine();

}

}

}

Durante una ejecución muestra este mensaje:

Texto

Descripción generada automáticamente

Indica qué es lo que hace el programa:

1. Da de alta una especialidad con id 123 y descripción MySpecialityDemo. Busca las que hay registradas que son 25. Busca alguna que tenga esa descripción y coincide que es la 29 y la devuelve. Hay dos con el mismo nombre.
2. Da de alta una especialidad con id 123 y descripción MySpecialityDemo. Busca las que hay registradas que son 25. Busca alguna que tenga esa descripción y le ha dado el código 29 que es la que encuentra. Implica que el añadir aunque le pongamos el identificador y lo devuelva no es correcta la implementación del servidor.
3. Añade y busca una especialidad, pero esa traza de ejecución no es posible.
4. Hay un error porque no se pueden tener 25 especialidades guardadas.
5. Ampliamos la funcionalidad para incluir una actualización con este código:

using System;

using System.Collections.Generic;

using System.Net.Http;

using System.Net.Http.Headers;

using System.Net.Http.Json;

using System.Text.Json;

using System.Threading.Tasks;

namespace ConsoleApp

{

internal class Pregunta8

{

static readonly HttpClient client = new HttpClient();

public class Speciality

{

public int id { get; set; }

public string name { get; set; }

public override string ToString()

{

return "id: " + id+ " name " + name;

}

}

static async Task<Speciality> CreateSpecialityAsync(Speciality speciality)

{

HttpResponseMessage response = await client.PostAsJsonAsync(

"api/specialties", speciality);

response.EnsureSuccessStatusCode();

string jsonContent = response.Content.ReadAsStringAsync().Result;

Speciality dev = JsonSerializer.Deserialize<Speciality>(jsonContent);

return dev;

}

static async Task<List<Speciality>> GetSpecialitiesAsync()

{

HttpResponseMessage response = await client.GetAsync(

"api/specialties");

response.EnsureSuccessStatusCode();

string jsonString = await response.Content.ReadAsStringAsync();

List<Speciality> specialities = JsonSerializer.Deserialize<List<Speciality>>(jsonString);

return specialities;

}

static async Task<Speciality> GetSpecialityAsync(int id)

{

HttpResponseMessage response = await client.GetAsync(

"api/specialties" + "/" + id);

response.EnsureSuccessStatusCode();

string jsonString = await response.Content.ReadAsStringAsync();

Speciality dev = JsonSerializer.Deserialize<Speciality>(jsonString);

return dev;

}

static Speciality findByName (List<Speciality> specialityList, String name)

{

foreach (Speciality s in specialityList)

{

if (s.name.Equals(name))

return s;

}

return null;

}

static async Task<Speciality> UpdateSpecialityAsync(Speciality speciality)

{

HttpResponseMessage response = await client.PutAsJsonAsync(

$"api/specialties/{speciality.id}", speciality);

response.EnsureSuccessStatusCode();

string jsonString = await response.Content.ReadAsStringAsync();

Speciality dev = JsonSerializer.Deserialize<Speciality>(jsonString);

return dev;

}

static async Task Main()

{

try

{

client.BaseAddress = new Uri("http://localhost:9966/petclinic/");

client.DefaultRequestHeaders.Accept.Clear();

client.DefaultRequestHeaders.Accept.Add(

new MediaTypeWithQualityHeaderValue("application/json"));

Speciality specialityToBeCreated = new Speciality()

{

name = "MySpecialityDemo-Pregunta8"

};

// Añadir

Speciality specialityCreated = await CreateSpecialityAsync(specialityToBeCreated);

Console.WriteLine("specialityCreated -> " + specialityCreated);

// Buscar todos

List<Speciality> specialityList = await GetSpecialitiesAsync();

Console.WriteLine("specialityList size -> " + specialityList.Count);

Speciality specialityFounded = findByName(specialityList, "MySpecialityDemo-Pregunta8");

Console.WriteLine("specialityFounded -> " + specialityFounded);

// Buscar por id

Speciality specialityFoundedIndividual = await GetSpecialityAsync(specialityFounded.id);

Console.WriteLine("specialityFoundedIndividual -> " + specialityFoundedIndividual);

// Actualizar

specialityFoundedIndividual.name = "MySpecialityDemo-Changed";

Speciality specialityChanged = await UpdateSpecialityAsync(specialityFoundedIndividual);

Console.WriteLine("specialityChanged -> " + specialityChanged);

// Buscar por id de nuevo

specialityFoundedIndividual = await GetSpecialityAsync(specialityFounded.id);

Console.WriteLine("specialityFoundedIndividual tras cambio -> " + specialityFoundedIndividual);

}

catch (Exception e)

{

Console.WriteLine(e.Message);

Console.WriteLine(e.ToString());

}

Console.ReadLine();

}

}

}

Indica qué es lo que hace el programa:

1. Da de alta especialidad, la busca, la actualiza y la vuelve a buscar y en cada caso va mostrando los valores.
2. Da de alta especialidad, la busca, la actualiza y la vuelve a buscar, pero al buscarla al final vemos que la ha borrado.
3. Da de alta especialidad, la busca, la actualiza y la vuelve a buscar, pero al buscarla al final vemos que la ha duplicado.
4. Hay un error en el servidor y, cuando intentamos actualizar, falla.
5. Modificamos ligeramente el programa para que asuma el error del servidor y realice un borrado:

using System;

using System.Collections.Generic;

using System.Net;

using System.Net.Http;

using System.Net.Http.Headers;

using System.Net.Http.Json;

using System.Text.Json;

using System.Threading.Tasks;

namespace ConsoleApp

{

internal class Pregunta9

{

static readonly HttpClient client = new HttpClient();

public class Speciality

{

public int id { get; set; }

public string name { get; set; }

public override string ToString()

{

return "id: " + id+ " name " + name;

}

}

static async Task<Speciality> CreateSpecialityAsync(Speciality speciality)

{

HttpResponseMessage response = await client.PostAsJsonAsync(

"api/specialties", speciality);

response.EnsureSuccessStatusCode();

string jsonContent = response.Content.ReadAsStringAsync().Result;

Speciality dev = JsonSerializer.Deserialize<Speciality>(jsonContent);

return dev;

}

static async Task<List<Speciality>> GetSpecialitiesAsync()

{

HttpResponseMessage response = await client.GetAsync(

"api/specialties");

response.EnsureSuccessStatusCode();

string jsonString = await response.Content.ReadAsStringAsync();

List<Speciality> specialities = JsonSerializer.Deserialize<List<Speciality>>(jsonString);

return specialities;

}

static async Task<Speciality> GetSpecialityAsync(int id)

{

HttpResponseMessage response = await client.GetAsync(

"api/specialties" + "/" + id);

response.EnsureSuccessStatusCode();

string jsonString = await response.Content.ReadAsStringAsync();

Speciality dev = JsonSerializer.Deserialize<Speciality>(jsonString);

return dev;

}

static Speciality findByName (List<Speciality> specialityList, String name)

{

foreach (Speciality s in specialityList)

{

if (s.name.Equals(name))

return s;

}

return null;

}

static async Task<String> UpdateSpecialityAsync(Speciality speciality)

{

HttpResponseMessage response = await client.PutAsJsonAsync(

$"api/specialties/{speciality.id}", speciality);

response.EnsureSuccessStatusCode();

string jsonString = await response.Content.ReadAsStringAsync();

return jsonString;

}

static async Task<HttpStatusCode> DeleteSpecialityAsync(int id)

{

HttpResponseMessage response = await client.DeleteAsync(

"api/specialties/" + id);

return response.StatusCode;

}

static async Task Main()

{

try

{

client.BaseAddress = new Uri("http://localhost:9966/petclinic/");

client.DefaultRequestHeaders.Accept.Clear();

client.DefaultRequestHeaders.Accept.Add(

new MediaTypeWithQualityHeaderValue("application/json"));

Speciality specialityToBeCreated = new Speciality()

{

name = "MySpecialityDemo-Pregunta9"

};

// Añadir

Speciality specialityCreated = await CreateSpecialityAsync(specialityToBeCreated);

Console.WriteLine("specialityCreated -> " + specialityCreated);

// Buscar todos

List<Speciality> specialityList = await GetSpecialitiesAsync();

Console.WriteLine("specialityList size -> " + specialityList.Count);

Speciality specialityFounded = findByName(specialityList, "MySpecialityDemo-Pregunta9");

Console.WriteLine("specialityFounded -> " + specialityFounded);

// Buscar por id

Speciality specialityFoundedIndividual = await GetSpecialityAsync(specialityFounded.id);

Console.WriteLine("specialityFoundedIndividual -> " + specialityFoundedIndividual);

// Actualizar

specialityFoundedIndividual.name = "MySpecialityDemo-PRegunta9-Changed";

String json = await UpdateSpecialityAsync(specialityFoundedIndividual);

Console.WriteLine("specialityChanged -> " + json);

// Buscar por id de nuevo

specialityFoundedIndividual = await GetSpecialityAsync(specialityFounded.id);

Console.WriteLine("specialityFoundedIndividual tras cambio -> " + specialityFoundedIndividual);

// Borrar

var statusCode = await DeleteSpecialityAsync(specialityFounded.id);

Console.WriteLine($"Deleted (HTTP Status = {(int)statusCode})");

}

catch (Exception e)

{

Console.WriteLine(e.Message);

Console.WriteLine(e.ToString());

}

Console.ReadLine();

}

}

}

Indica cuál es el resultado de su ejecución:

1. Sigue dando una excepción al actualizar.
2. Aparentemente crea, busca, actualiza y borra, pero el borrado no se realiza bien.
3. Crea, busca, actualiza y borra una especialidad correctamente.
4. Da una nueva excepción al borrar.
5. Vamos a realizar la integración con un servidor que tiene seguridad.

Para el servicio REST de Pet-Clinic y edita el fichero application.properties que tienes en la ruta pet-clinic\spring-petclinic-rest\src\main\resources. Debes poner en la parte de abajo petclinic.security.enable=true.

Si lo prefieres, tienes un fichero application.properties.conseguridad que puedes renombrar a application.properties.

Vuelve a lanzar el servidor con mvn spring-boot:run.

Recuerda que usa una base de datos en memoria, por lo que todos los datos que tuvieras de pruebas anteriores se perderán.

Si intentas acceder al swagger te pedirá credenciales, por defecto puedes usar admin-admin para usuario y contraseña:

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Intenta ejecutar este programa:

using System;

using System.Collections.Generic;

using System.Net.Http;

using System.Net.Http.Headers;

using System.Text.Json;

using System.Threading.Tasks;

namespace ConsoleApp

{

internal class Pregunta9

{

static readonly HttpClient client = new HttpClient();

public class Speciality

{

public int id { get; set; }

public string name { get; set; }

public override string ToString()

{

return "id: " + id+ " name " + name;

}

}

static async Task<List<Speciality>> GetSpecialitiesAsync()

{

HttpResponseMessage response = await client.GetAsync(

"api/specialties");

Console.WriteLine("response -> " + response);

response.EnsureSuccessStatusCode();

string jsonString = await response.Content.ReadAsStringAsync();

List<Speciality> specialities = JsonSerializer.Deserialize<List<Speciality>>(jsonString);

return specialities;

}

static async Task Main()

{

try

{

client.BaseAddress = new Uri("http://localhost:9966/petclinic/");

client.DefaultRequestHeaders.Accept.Clear();

client.DefaultRequestHeaders.Accept.Add(

new MediaTypeWithQualityHeaderValue("application/json"));

// Buscar todos

List<Speciality> specialityList = await GetSpecialitiesAsync();

if (specialityList != null)

{

Console.WriteLine("specialityList size -> " + specialityList.Count);

foreach (Speciality s in specialityList)

Console.WriteLine("Speciality -> " + s);

}

}

catch (Exception e)

{

Console.WriteLine(e.Message);

Console.WriteLine(e.ToString());

}

Console.ReadLine();

}

}

}

Y verás que muestra un error de autenticación:

Captura de pantalla de computadora

Descripción generada automáticamente

Si probamos desde un cliente como POSTMAN, añadiendo la autorización básica de usuario y contraseña veremos que ya funciona:

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Modifica el programa para incluir la autenticación en la cabecera de las peticiones:

static async Task Main()

{

try

{

client.BaseAddress = new Uri("http://localhost:9966/petclinic/");

client.DefaultRequestHeaders.Accept.Clear();

client.DefaultRequestHeaders.Accept.Add(

new MediaTypeWithQualityHeaderValue("application/json"));

var clientId = "admin";

var clientSecret = "admin";

var authenticationString = $"{clientId}:{clientSecret}";

var base64EncodedAuthenticationString = Convert.ToBase64String(System.Text.ASCIIEncoding.UTF8.GetBytes(authenticationString));

client.DefaultRequestHeaders.Add("Authorization", "Basic " + base64EncodedAuthenticationString);

// Buscar todos

List<Speciality> specialityList = await GetSpecialitiesAsync();

if (specialityList != null)

{

Console.WriteLine("specialityList size -> " + specialityList.Count);

foreach (Speciality s in specialityList)

Console.WriteLine("Speciality -> " + s);

}

}

catch (Exception e)

{

Console.WriteLine(e.Message);

Console.WriteLine(e.ToString());

}

Console.ReadLine();

}

}

Indica cuál es el resultado de su ejecución:

1. Sigue dando un error de autenticación.
2. Funciona correctamente y devuelve e imprime las tres especialidades nuevas.
3. Da un error de ejecución que no tiene nada que ver con autenticación.
4. Funciona correctamente aunque no imprime nada.