ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE INGENIERÍA ELECTRÍCA Y ELECTRÓNICA

DESARROLLO DE UN SISTEMA DISTRIBUIDO DE GESTIÓN DE FOTOGRAFÍAS

SUBSISTEMA DE CONSULTA

TRABAJO DE INTEGRACIÓN CURRICULAR PRESENTADO COMO REQUISITO PARA LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN TECNOLOGÍAS DE LA INFORMACIÓN

STEVEN LEONARDO RONDAL ENRÍQUEZ

steven.rondal@epn.edu.ec

DIRECTOR: RAÚL DAVID MEJÍA NAVARRETE

david.mejia@epn.edu.ec

DMQ, abril 2023

CERTIFICACIONES

Yo, STEVEN LEONARDO RONDAL ENRÍQUEZ declaro que el trabajo de integración curricular aquí descrito es de mi autoría; que no ha sido previamente presentado para ningún grado o calificación profesional; y, que he consultado las referencias bibliográficas que se incluyen en este documento.

STEVEN LEONARDO RONDAL ENRÍQUEZ

Certifico que el presente trabajo de integración curricular fue desarrollado por STEVEN LEONARDO RONDAL ENRÍQUEZ, bajo mi supervisión.

RAUL DAVID MEJÍA NAVARRETE
DIRECTOR

DECLARACIÓN DE AUTORÍA

A través de la presente declaración, afirmamos que el trabajo de integración curricular aquí descrito, así como el (los) producto(s) resultante(s) del mismo, son públicos y estarán a disposición de la comunidad a través del repositorio institucional de la Escuela Politécnica Nacional; sin embargo, la titularidad de los derechos patrimoniales nos corresponde a los autores que hemos contribuido en el desarrollo del presente trabajo; observando para el efecto las disposiciones establecidas por el órgano competente en propiedad intelectual, la normativa interna y demás normas.

STEVEN LEONARDO RONDAL ENRÍQUEZ

RAÚL DAVID MEJÍA NAVARRETE

DEDICATORIA

A mi familia.

ÍNDICE DE CONTENIDO

RESUMEN

El presente Trabajo de Integración Curricular se enfoca en el desarrollo de un subsistema de consulta, implementado usando la tecnología ASP.NET web API, que permita remitir fotografías previamente almacenadas con base en etiquetas definidas por el usuario. El subsistema se encarga de procesar las etiquetas ingresadas por el usuario a través de un subsistema de adquisición de fotografías, así como consumir información de un subsistema de almacenamiento, para posteriormente devolver la información filtrada con base en la petición realizada por el usuario, la cual incluye las etiquetas en las que está interesado el mencionado usuario para lo cual consume las fotografías obtenidas por medio del subsistema de clasificación.

Como parte de ese trabajo, para emular la funcionalidad de los subsistemas de almacenamiento, clasificación y adquisición de fotografías se usarán *stubs*.

El subsistema de consulta corresponde al *back-end* que requiere el sistema para su funcionamiento; sin embargo, para poder probar el subsistema se requiere emular el *front-end*. Dado que en este trabajo no se desarrolla el subsistema de adquisición de fotografías, se debe crear un *stub* para el mismo, y por tanto, para emular el *front-end* se desarrollará una aplicación web mediante el *framework* Angular. Para el desarrollo del *back-end* se hará uso del *framework* ASP.NET web API, que permite implementar un servicio web en el que se encuentran los *endpoints* necesarios para procesar las peticiones y respuestas provenientes del *front-end*.

Además, en el *back-end* se implementará el *stub* del subsistema de almacenamiento, el *stub* del subsistema de clasificación y la lógica del subsistema de consulta. Para el desarrollo del *stub* del subsistema de almacenamiento se hará uso de Entity Framework, que permite la interacción de una base de datos con el *back-end*. Para el desarrollo del *stub* del subsistema de clasificación se implementará e integrará un conjunto de directorios y fotografías iniciales. Para el desarrollo del subsistema de consulta se implementará un *stub* que emule la lógica de consulta de fotografías.

PALABRAS CLAVE: subsistema, consulta, *stub*, etiquetas, fotografías, ASP.NET.

ABSTRACT

This Curricular Integration Work is focused on the development of a query subsystem implemented through ASP.NET web API technology that allows sending previously stored photos from tags defined by a user. The subsystem processes the tags entered by a user through a photos acquisition subsystem. It consumes information from a storage subsystem to later return the filtered information based on the request made by the user, which includes the labels chosen by the user, for which it consumes the photos obtained through the classification subsystem.

As part of this work, to emulate the functionality of the storage, classification and photos acquisition subsystems, stubs will be used.

The query subsystem corresponds to the back-end that the system requires for its operation, however, to be able to test the subsystem it is required to emulate the front-end. Since in this work the photos acquisition subsystem is not developed, a stub must be created for it, and therefore, to emulate the front-end a web application will be developed using the Angular framework. For the development of the back-end, the ASP.NET web API framework will be used, which allows a web service implementation where the necessary endpoints are located to process the requests and responses that arrive from the front-end.

In addition, the storage subsystem stub, the classification subsystem stub, and the query subsystem logic will be implemented in the back-end. For the development of the storage subsystem stub, the Entity Framework will be used, which allows the interaction of a database with the back-end. For the development of the classification subsystem stub, a set of initial directories and photos will be implemented and integrated. For the development of the query subsystem, a stub will be implemented that emulates the photos query logic.

KEYWORDS: subsystem, query, stub, labels, photos, ASP.NET.

1. INTRODUCCIÓN

Como parte del componente se desarrolló un subsistema de consulta que permite a un usuario obtener fotografías filtradas por etiquetas. El subsistema consta de un servicio web desarrollado a través de ASP.NET web API, y será el encargado de procesar consultas y proporcionar a un usuario un conjunto de fotografías que coincidan con el criterio de búsqueda (una o más etiquetas) ingresado por el mismo.

El subsistema se encarga de procesar la información solicitada por el usuario, consumir datos de un subsistema de almacenamiento y un subsistema de clasificación, para así obtener y devolver al usuario el resultado de la consulta filtrada por valores de etiquetas ingresadas mediante un subsistema de adquisición de fotografía. Para emular la funcionalidad de los subsistemas de almacenamiento, clasificación y adquisición de fotografías se usarán *stubs*.

El subsistema será desarrollado con base en la metodología Kanban.

1.1. OBJETIVO GENERAL

Desarrollar un subsistema de consulta mediante ASP.NET web API que permita obtener fotografías con base en etiquetas definidas por el usuario.

1.2. OBJETIVOS ESPECÍFICOS

- Analizar los fundamentos teóricos y herramientas requeridos para desarrollar un servicio web mediante ASP.NET web API.
- Diseñar los elementos del subsistema que permitan obtener las fotografías.
- Implementar los elementos del subsistema, incluyendo los *stubs* que permitan emular el funcionamiento de otros subsistemas como almacenamiento, clasificación y adquisición de fotografías.

1.3. ALCANCE

El presente Trabajo de Integración Curricular plantea desarrollar un prototipo de subsistema de consulta de fotografías, que permita recuperar un conjunto de fotografías

que correspondan con el criterio de la consulta, es decir, que coincidan con la o las etiquetas ingresadas por el usuario.

El desarrollo del subsistema contará con las siguientes fases:

Fase teórica

Revisión de fundamentos teóricos relacionados con el desarrollo del subsistema de consulta, como: ASP.NET web API, lenguaje de programación C#, implementación de *endpoints*, procesamiento de imágenes, bases de datos, Angular y Entity Framework.

Fase de diseño

El diseño del subsistema de consulta y los *stubs* requiere de un análisis de requerimientos. Con base en los resultados se puede representar la estructura, características y comportamiento del subsistema de consulta y los *stubs* de forma más detallada y concreta.

Las etapas de diseño son:

- Diseño del subsistema de consulta.
- Diseño de *stub* de almacenamiento.
- Diseño de *stub* de adquisición de fotografías.
- Diseño de *stub* de clasificación.

Fase de implementación

Para la implementación del subsistema de consulta se requiere del levantamiento del ambiente de desarrollo (*back-end)*, luego se implementan los métodos y clases requeridos para el funcionamiento del subsistema, así como los *endpoints* requeridos para procesar peticiones provenientes del *stub* del subsistema de adquisición de fotografías.

Posteriormente se implementa el *stub* del subsistema de almacenamiento, en el que desarrolla la lógica para que el servicio web interactúe con una base de datos.

Luego se implementa el *stub* del subsistema de clasificación, que para este caso estará simplificado y conformado por un conjunto de directorios e imágenes, creados y editados, de forma manual.

Una vez implementados los *endpoints* necesarios y los *stubs* del subsistema de almacenamiento y clasificación, se realiza la implementación del *stub* del subsistema de adquisición de fotografías (*front-end*), donde se implementa la lógica de consulta y presentación de fotografías con base en etiquetas.

Finalmente, para la implementación del subsistema de consulta se desarrolla la lógica en el *front-end* que permita generar peticiones de consulta de fotografías al *back-end*, procesar las peticiones y consultar la información de las fotografías al *stub* del subsistema de almacenamiento. Se requiere dos tipos de consulta: una que permita obtener todas las fotografías existentes y otra que permita recuperar las fotografías con base en las etiquetas definidas por el usuario.

Fase de pruebas

Se realizarán pruebas unitarias para verificar el funcionamiento del prototipo. Adicionalmente, se coordinará con cinco usuarios quienes realizarán pruebas de la aplicación y evaluarán su funcionalidad a través de una encuesta. Se realizarán correcciones considerando los resultados de las pruebas.

1.4. MARCO TEÓRICO

En esta sección se detallan las herramientas y los fundamentos teóricos que permitirán diseñar e implementar el subsistema de consulta y los *stubs* de los subsistemas de almacenamiento, clasificación y adquisición de fotografías.

Arquitectura cliente – servidor: Es un modelo de diseño de software compuesto por dos componentes: los proveedores servicios, llamados servidores, y los consumidores, llamados clientes. El servidor es un dispositivo con gran capacidad que proporciona una serie de servicios o recursos. El cliente es un dispositivo que puede comunicarse con el servidor y acceder a sus servicios y recursos a través de Internet o una red interna. Esta arquitectura se puede utilizar en diferentes modelos informáticos, y su propósito es establecer y mantener una comunicación de información entre diferentes entidades de una red a través del uso de protocolos establecidos [1]. En la Figura 1.1 se incluye un ejemplo de la arquitectura cliente-servidor.

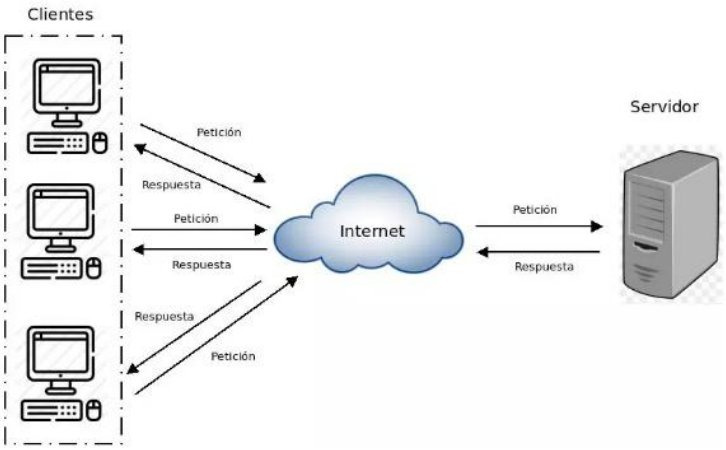

Figura 1.1. Arquitectura cliente - servidor [1]

MVC (*Model View Controller*): Es un patrón empleado en el desarrollo de software, que permite separar la lógica de la aplicación de la lógica de la vista en una aplicación. Se utiliza para separar código con base en sus distintas responsabilidades, manteniendo distintas capas que se encargan de hacer una tarea concreta, y considera tres capas. En la Figura 1.2 se detalla la interacción entre las capas y el usuario. Las capas y su funcionalidad son [2], [3]:

- **Modelo**: Contiene una representación de los datos y la lógica de negocio. Es la capa que se encarga de la interacción con los datos.
- **Vista**: Contiene los componentes que integran la interfaz de usuario. Es la capa que permite la interacción del usuario con una aplicación.
- **Controlador**: Contiene la lógica de procesamiento de peticiones. Esta capa sirve de enlace entre las capas Vista y Modelo, ya que consume información de la capa Modelo para compartirla con la capa Vista.

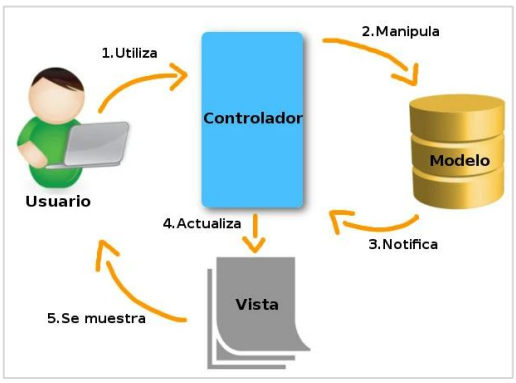

Figura 1.2. MVC: Interacción entre capas y usuario [3]

***Framework*:** Es una estructura o esquema de trabajo empleado por programadores para realizar el desarrollo de un proyecto. Ofrece un conjunto de módulos y herramientas que pueden ser reusados para múltiples proyectos. Emplear un *framework* permite agilizar y optimizar los procesos de desarrollo por que evita tener que desarrollar código de forma repetitiva, así también asegura la consistencia del código y buenas prácticas [4].

ASP.NET: Es un *framework* gratuito creado por Microsoft que sirve para crear páginas web dinámicas, servicios web y aplicaciones web mediante tecnologías como HTML[1], CSS[2] y JavaScript[3]. Es multiplataforma dado que permite ejecutar aplicaciones en diferentes sistemas operativos. La versión más reciente se denomina .NET Core, precediendo a las versiones ASP.NET y ASP [5], [6].

REST (*Representational State Transfer*): Es un tipo de arquitectura de software que sirve para establecer comunicación entre clientes y servidores, a través del protocolo HTTP[4]. Generalmente emplea JSON[5] o XML[6] como formato de intercambio de datos [7]. Emplea los métodos definidos en el protocolo HTTP, conocidos como verbos, los cuales son:

- **POST**: Permite crear recursos nuevos.
- **GET**: Permite obtener recursos.
- **PUT**: Permite modificar o actualizar recursos.
- **DELETE**: Permite borrar recursos.

Una API RESTful es una interfaz que emplea los principios de REST para permitir a un cliente comunicarse con el servidor, la cual funciona bajo tres conceptos [8]:

---

[1] HTML (*HyperText Markup Language*): Es un lenguaje de marcación que sirve para definir el contenido de páginas web.

[2] CSS (*Cascading Style Sheets*): Es un lenguaje de estilos usado para estilizar elementos HTML.

[3] JavaScript: Lenguaje de programación que sirve para desarrollar páginas web interactivas.

[4] HTTP (*Hypertext Transfer Protocol*): Es un protocolo de comunicación que permite la transferencia de información en la web.

[5] JSON (*Javascript Object Notation*): Es un formato basado en texto estándar utilizado para transmitir datos en aplicaciones web.

[6] XML (*Extensible Markup Language*): Es un lenguaje de marcado diseñado para extensible diseñado para almacenar y transportar datos.

1. Dentro de una API RESTful todo debe ser un recurso.
2. Los recursos en REST se manipulan mediante una URI[7].
3. Todas las peticiones deben estar asociadas a alguno de los verbos de HTTP.

ASP.NET web API: Es un *framework* que facilita la creación de servicios basados en HTTP, los cuales están disponibles para una amplia variedad de clientes, entre los que se incluyen navegadores web y dispositivos móviles. Es una plataforma ideal para implementar aplicaciones de tipo RESTful usando .NET Framework. Como características principales se pueden mencionar [9]:

- Es una plataforma útil para la implementación de servicios web RESTful.
- Se basa en ASP.NET y admite diferentes formatos de datos de respuesta como JSON o XML.
- Permite el desarrollo de servicios HTTP funcionales para clientes como navegadores o dispositivos móviles.
- Permite implementar páginas web dinámicas usando lenguajes de programación como C#[8] y Visual Basic.

Metodología Kanban: Es una metodología de gestión de proyectos centrada en la mejora continua que ayuda a optimizar y organizar el proceso de gestión de tareas. Es un método visual que permite conocer el estado de proyectos y agilizar la asignación de nuevas tareas. Esta metodología se puede implementar a través de un tablero Kanban, que es una herramienta ágil de gestión que permite visualizar flujos, distribuir carga de trabajo y maximizar la eficiencia. Un tablero Kanban de estructura básica describe un proyecto por medio de un tablero organizado por tres columnas que especifican el estado de una tarea: pendiente, en progreso y completada. Las tareas son representadas en el tablero por tarjetas visuales que avanzan a través de las diferentes columnas hasta que sean completadas [10].

[7] URI (*Uniform Resource Identifier*): Es una cadena de caracteres que sirve para identificar y localizar recursos de forma univoca en la web.

[8] C#: Lenguaje de programación orientado a objetos que permite crear diversos tipos de aplicaciones que se ejecutan en .NET.

Diagrama de clases: Es un tipo de diagrama que describe un sistema considerando los diferentes tipos de objetos que integran un sistema y los tipos de relaciones estáticas que existen entre ellos. Es un diagrama orientado al modelo de programación orientado a objetos. Es útil para describir diseños detallados, comprender los requisitos de programas informáticos y explorar los conceptos de dominio [11].

Diagrama entidad-relación: También conocido como modelo entidad relación, es un tipo de diagrama que describe cómo las entidades se relacionan entre sí dentro de un sistema. Este tipo de diagrama se usa comúnmente para diseñar bases de datos relacionales. Emplea un conjunto definido de símbolos para representar entidades, relaciones y atributos [12].

***Stub*:** En programación, un *stub* es un fragmento de código que se utiliza para reemplazar una funcionalidad. Permite simular el comportamiento de una funcionalidad existente o ser un sustituto temporal de una funcionalidad que esta por desarrollarse. Los *stubs* son útiles en el aspecto de portabilidad, desarrollo y pruebas de software [13].

DTO (*Data Transfer Object*): Es un tipo de objeto que sirve únicamente para transferir datos. Evita exponer las clases que se relacionan directamente con una base de datos. En este objeto se pueden definir atributos y propiedades que pueden originarse de una o más entidades de información con el fin de que el objeto se ajuste a un determinado requerimiento [14].

***Endpoint*:** Es un término empleado en programación para referirse a una ubicación específica en una red donde una aplicación puede enviar y recibir datos. Un *endpoint* puede ser una dirección especifica o una URL[9] que apunta a un sitio web o recurso en una red. Proporciona una forma de exponer la funcionalidad de una aplicación y permite que los clientes accedan a esa funcionalidad. Además, puede emplearse para crear conexiones seguras entre clientes y servidores [15].

[9] URL (*Uniform Resource Locator*): Es un subconjunto de URI, con la diferencia que solo permite establecer la ubicación del recurso.

AutoMapper: Es una herramienta que permite extraer y procesar campos de datos de un objeto origen y relacionarlos con campos de un objeto destino. Resulta útil para procesar información entre una clase gestora de entidad y una clase para realizar DTO. En Visual Studio se puede integrar esta herramienta a la solución a través del administrador de paquetes NuGet. Mediante NuGet se puede buscar e instalar la herramienta llamada `AutoMapper.Extensions.Microsoft.DependencyInjection` [16]. En la Figura 1.3 se presenta una captura de pantalla del administrador de paquetes NuGet.

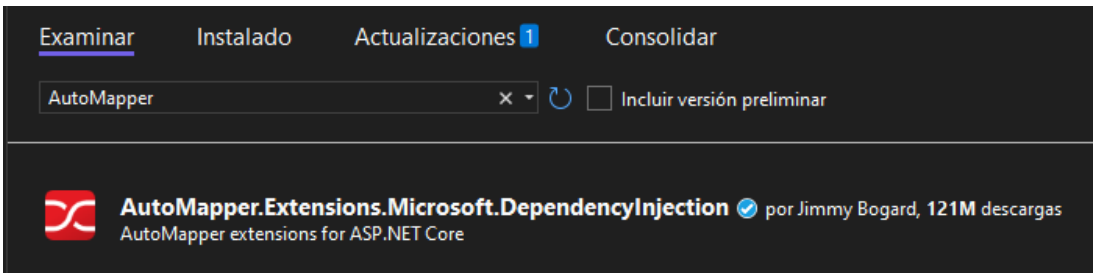

Figura 1.3. AutoMapper en el Administrador de paquetes NuGet

Entity Framework: Es un *framework* de código abierto para aplicaciones desarrolladas con .NET, permite a los desarrolladores trabajar con datos a través de clases específicas sin centrarse en las tablas y columnas de la base de datos. Este *framework* permite trabajar a un mayor nivel de abstracción con respecto al manejo de datos con menos código en comparación a otras tecnologías [17]. En Visual Studio se puede integrar esta herramienta en una solución a través del administrador de paquetes NuGet. Se puede buscar e instalar la herramienta denominada `Microsoft.EntityFrameworkCore.SqlServer` mediante NuGet, en la Figura 1.4 se presenta una captura de pantalla del administrador de paquetes NuGet.

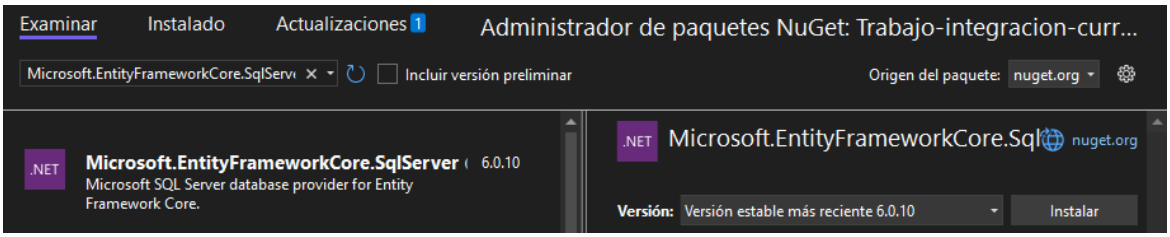

Figura 1.4. Entity Framework en el Administrador de paquetes NuGet

Además, resulta útil instalar la herramienta `Microsoft.EntityFrameworkCore.Tools` para integrar funcionalidades complementarias útiles que permiten a Entity Framework

generar una base de datos e interactuar con ella a través de comandos [18]. En la Figura 1.5 se presenta una captura de pantalla del administrador de paquetes NuGet.

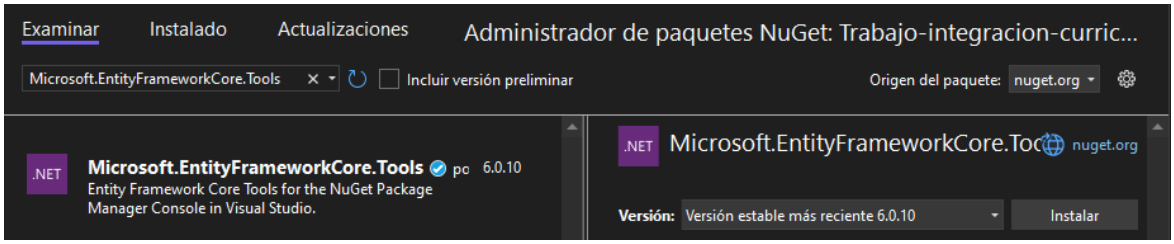

Figura 1.5. Herramienta adicional en el Administrador de paquetes NuGet

Algunos de los comandos más útiles se detallan en la Tabla 1.1.

Tabla 1.1. Comandos para la interacción con la base de datos [19]

Comando	Descripción
`Add-Migration "Nombre_migración"`	Crea o actualiza el esquema de la base de datos y le asigna el nombre indicado.
`Update-Database`	Actualiza y de ser necesario genera la estructura de la base de datos.
`drop-Database -context "Nombre_clase_contexto"`	Elimina la base de datos indicada.

Angular: Es un *framework* de JavaScript de código abierto escrito en TypeScript[10]. Su propósito consiste en el desarrollo de aplicaciones web de una sola página. Angular incluye [20], [21]:

- Un conjunto de componentes que permite crear aplicaciones web escalables.
- Una colección de bibliotecas integradas que contiene variedad de funcionalidades como el enrutamiento, comunicación cliente-servidor, gestión de formularios, entre otras.
- Un conjunto de herramientas que ayudan a desarrollar, compilar, probar y actualizar código.

[10] TypeScript: Es un lenguaje de programación fuertemente tipado basado en JavaScript.

Para crear un proyecto con Angular, se requiere de los siguientes prerrequisitos:

- Descargar e instalar node.js desde `https://nodejs.org/en/`
- Instalar Angular CLI[11].

Para instalar Angular CLI se puede ejecutar en una consola el comando:

```
npm install -g @angular/cli
```

Angular permite generar diferentes recursos a través de Angular CLI. En la Tabla 1.2 se detallan algunos de los comandos más importantes.

Tabla 1.2. Comandos Angular [20]

Comando	Descripción
`ng new "nombre_proyecto"`	Crea un nuevo proyecto.
`npm start`	Ejecuta o inicia la aplicación.
`ng g c "ubicación/nombre_componente"`	Crea un componente en la ubicación dada.
`ng g s "ubicación/nombre_servicio"`	Crea un servicio en la ubicación dada.

En Angular se tienen componentes, que son los bloques fundamentales que permiten implementar una aplicación en Angular. El modelo de componentes de Angular permite generar una estructura de aplicación intuitiva y brinda una fuerte encapsulación.

Un componente está integrado por un archivo HTML, un archivo CSS y un archivo Typescript. En la Figura 1.6 se presenta una captura de pantalla de la estructura fundamental del componente en el archivo Typescript, en el cual se definen los archivos asociados al componente y el nombre del selector por medio del decorador `@Component()`.

```
@Component({
  selector: 'app-listado-fotos',
  templateUrl: './listado-fotos.component.html',
  styleUrls: ['./listado-fotos.component.css']
})
```

Figura 1.6. Elementos de un componente

[11] CLI (*Command Line Interface*): Es una interfaz de usuario que permite definir instrucciones a través de una línea de texto simple.

En el decorador se debe especificar [22]: **`selector`**, permite definir el nombre que identifica e instancia al componente; **`templateUrl`**, permite definir el archivo o plantilla HTML que indica a Angular cómo representar el componente; y, **`styleUrls`**, permite definir un conjunto de estilos CSS que especifican la apariencia de los elementos HTML que se aplicarán a la plantilla.

Por otro lado, los servicios en Angular son clases que pueden acceder a los datos, para que puedan ser entregados a los componentes para su presentación. Un servicio es reutilizable y puede ser usado en distintos componentes [23].

2. METODOLOGÍA

En esta sección se presentan los detalles para el desarrollo del presente Trabajo de Integración Curricular. Se detallan los aspectos de diseño e implementación considerados para el desarrollo de un subsistema de consulta mediante ASP.NET web API que permita obtener fotografías con base en etiquetas definidas por el usuario.

2.1. DISEÑO

En esta sección se detallan los requerimientos, la arquitectura, las tecnologías empleadas y el proceso de diseño del subsistema de consulta y los *stubs* que emulan a los subsistemas de almacenamiento, clasificación y adquisición de fotografías.

2.2. REQUERIMIENTOS

Los requerimientos que debe cumplir el prototipo del subsistema de consulta fueron recolectados por medio de entrevistas realizadas al cliente.

Funcionales

- El subsistema de consulta implementará la lógica que permita proporcionar a un usuario un conjunto de fotografías que coincidan con el criterio de búsqueda (una o más etiquetas) ingresado por el mismo.
- El subsistema entregará la información solicitada por el usuario a través del subsistema de adquisición de fotografías, para lo cual consumirá datos de un subsistema de almacenamiento y un subsistema de clasificación, para así obtener el resultado de la información solicitada.
- Para emular el comportamiento de los subsistemas de almacenamiento, clasificación y adquisición de fotografías se implementarán *stubs*.

No funcionales

- El subsistema de consulta será construido usando ASP.NET web API.

- El subsistema será desarrollado con base en la metodología Kanban.
- El *stub* del subsistema de almacenamiento será implementado por medio de Entity Framework.
- El *stub* del subsistema de clasificación será implementando a través de un conjunto de directorios y fotografías integrados de forma manual.
- El *stub* del subsistema de adquisición de fotografías será implementado por medio del *framework* Angular.

Planteamiento de tablero Kanban

El presente trabajo se desarrollará con base en la metodología Kanban. En la Figura 2.1 se presenta el estado del tablero Kanban. Se puede apreciar que en esta fase ya se ha completado las tareas relacionadas con la teoría y se tiene cuatro tareas de diseño en progreso, cuya finalización permitirá avanzar con las tareas de implementación, mientras que las tareas de implementación y pruebas están pendientes.

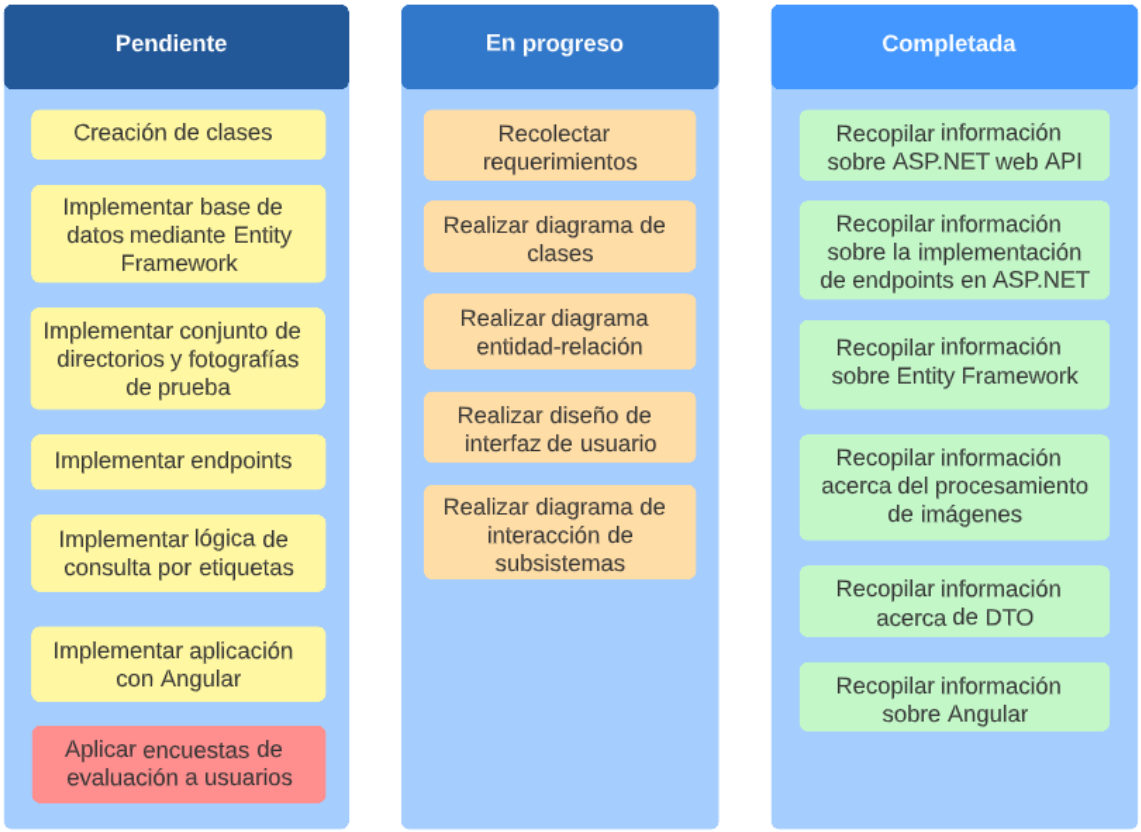

Figura 2.1. Tablero Kanban

2.3. ARQUITECTURA Y TECNOLOGÍAS

Para el desarrollo del componente se empleará una arquitectura cliente - servidor. En el lado del servidor se implementará un *back-end*, el cual estará conformado por un servicio web desarrollado con ASP.NET web API y una base de datos con SQL Server, así también para conectar la base de datos con el servicio web se emplea Entity Framework. Por otro lado, en el lado del cliente se dispone de un *stub* conformado por una aplicación web (*front-end*) desarrollada mediante Angular. El propósito del *stub* es permitir emular el comportamiento del cliente mediante la generación de peticiones de consulta solicitando fotografías para lo cual se proveerá de un conjunto de etiquetas que serán remitidas al servicio web y permitirá recibir las fotografías que cumplan con el criterio de búsqueda.

2.4. DISEÑO DEL SUBSISTEMA DE CONSULTA

Para el diseño del subsistema de consulta se consideran dos entidades fundamentales: `Foto` y `Etiqueta`. Ya que una foto puede corresponder a varias etiquetas y que varias etiquetas pueden estar presentes en una foto, se presenta una relación de muchos a muchos (`N:N`) entre las entidades. En la Figura 2.2 se presenta el diagrama de clases que modela el escenario descrito.

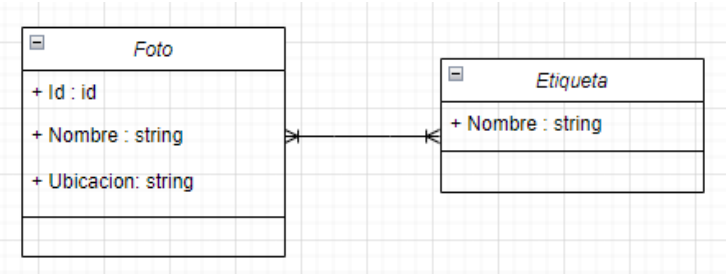

Figura 2.2. Diagrama de clases

En el diagrama de clases se aprecia que la entidad `Foto` posee los atributos `Id`, `Nombre` y `Ubicación`; y que la entidad `Etiqueta` posee el atributo `Nombre`. Con estas entidades se puede establecer la estructura necesaria para implementar la lógica de consulta de fotografías con base en etiquetas.

El comportamiento del subsistema de consulta y su interacción con los subsistemas de almacenamiento, clasificación y adquisición de fotografías se describe en la Figura 2.3. En el *front-end*, el *stub* que emula el subsistema de adquisición de fotografías permite al

usuario definir las etiquetas para la búsqueda de fotografías y envía una petición al *back-end*, luego de lo cual el subsistema de consulta procesa la petición e interactúa con el *stub* de almacenamiento para consultar la información de las fotografías que correspondan a las etiquetas indicadas por el cliente, el subsistema de consulta retorna la información al *stub* del subsistema de adquisición de fotografías para que pueda acceder a las fotografías almacenadas en la ubicación indicada por el *stub* del subsistema de clasificación.

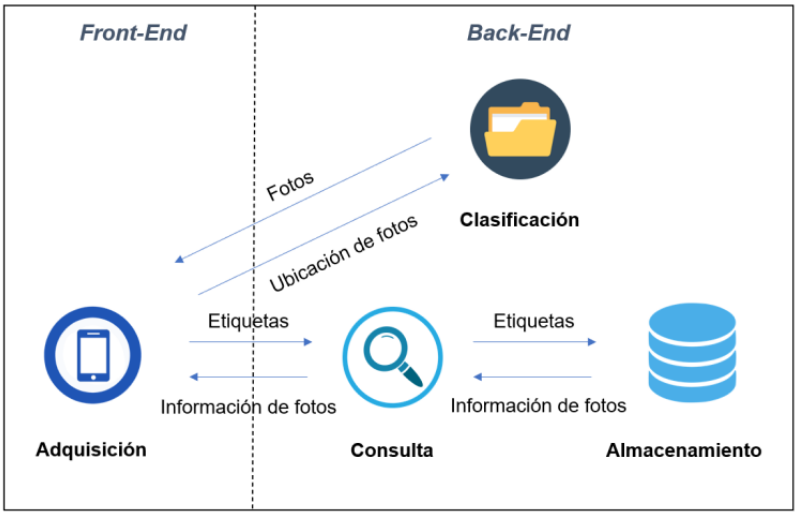

Figura 2.3. Diagrama de interacción entre subsistemas

El subsistema de consulta contará con una clase servicio que será la encargada de interactuar con la base de datos, es decir, contendrá toda la lógica de consulta de fotografías, además, contará con una clase que cumpla el rol de controlador, en la que se implementarán dos *endpoints* que permitirán atender los pedidos que los usuarios realicen mediante una petición HTTP GET para consultar todas las fotografías y para consultar fotografías con base en etiquetas. Tal interacción se detalla en Figura 2.4.

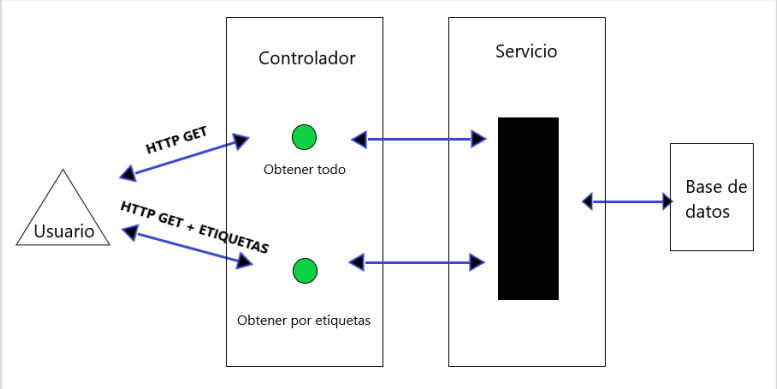

Figura 2.4. Diagrama de funcionamiento de peticiones

Stub que emula el subsistema de almacenamiento

El propósito del subsistema de almacenamiento es almacenar y procesar los registros relacionados con fotografías y etiquetas. Para su desarrollo se hará uso de una base de datos relacional que será integrada al servicio web por medio de Entity Framework.

Como parte del diseño de la base de datos se generó el diagrama entidad-relación con base en el diagrama de clases establecido en la Figura 2.2, en la cual se observan las clases `Foto` y `Etiqueta`, que corresponderán a una tabla para cada una en la base de datos. Adicionalmente, es necesario considerar que en una base de datos relacional la relación muchos a muchos generalmente se implementa mediante una tabla intermedia o de unión. En la Figura 2.5 se presenta el diagrama entidad-relación resultante.

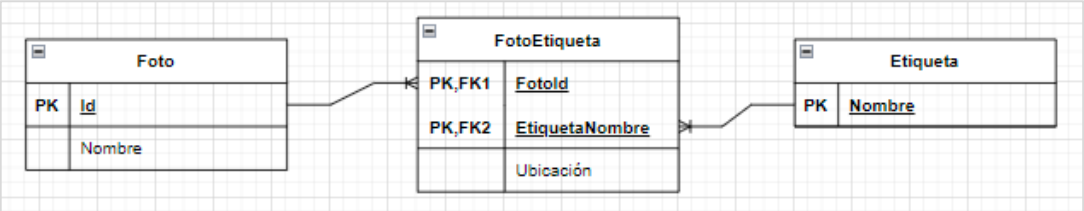

Figura 2.5. Diagrama entidad-relación

El diagrama entidad-relación está conformado por las entidades `Foto`, `Etiqueta` y `FotoEtiqueta`. La tabla `Foto` posee el atributo `Id` como llave primaria y un atributo `Nombre`, la tabla `Etiqueta` solo posee el atributo `Nombre` que a su vez es la llave primaria, y la tabla intermedia `FotoEtiqueta` está integrada por una llave primaria compuesta por los atributos `FotoId` y `EtiquetaNombre` que a su vez son llaves foráneas, además, posee un atributo `Ubicación` que se usa para almacenar la ruta de una fotografía en el servidor.

Para implementar el diseño de la base de datos mediante Entity Framework se requiere de una instancia de la clase `DbContext`, para esto se considera una clase llamada `AplicationDbContext` que derive de `DbContext`. Esta clase permitirá administrar la conexión con la base de datos, guardar instancias de entidades e interactuar con datos.

Stub que emula el subsistema de clasificación

El propósito del subsistema de clasificación es organizar y almacenar las fotografías en un conjunto de directorios. Los directorios serán creados con base en las etiquetas

predefinidas, los nombres de los directorios serán iguales a los nombres de las etiquetas. Además, se considerará una etiqueta general que esté presente en todas las fotografías, es decir, se creará un directorio llamado “General” donde se almacenarán todas las fotografías.

Stub que emula el subsistema de adquisición de fotografías

El propósito del subsistema de adquisición de fotografía es proporcionar una interfaz para que el usuario pueda interactuar con el subsistema de consulta. Esta interfaz permite la consulta y visualización de fotografías con base en etiquetas definidas por el usuario. En la Figura 2.6 se muestra el diseño de la interfaz de usuario.

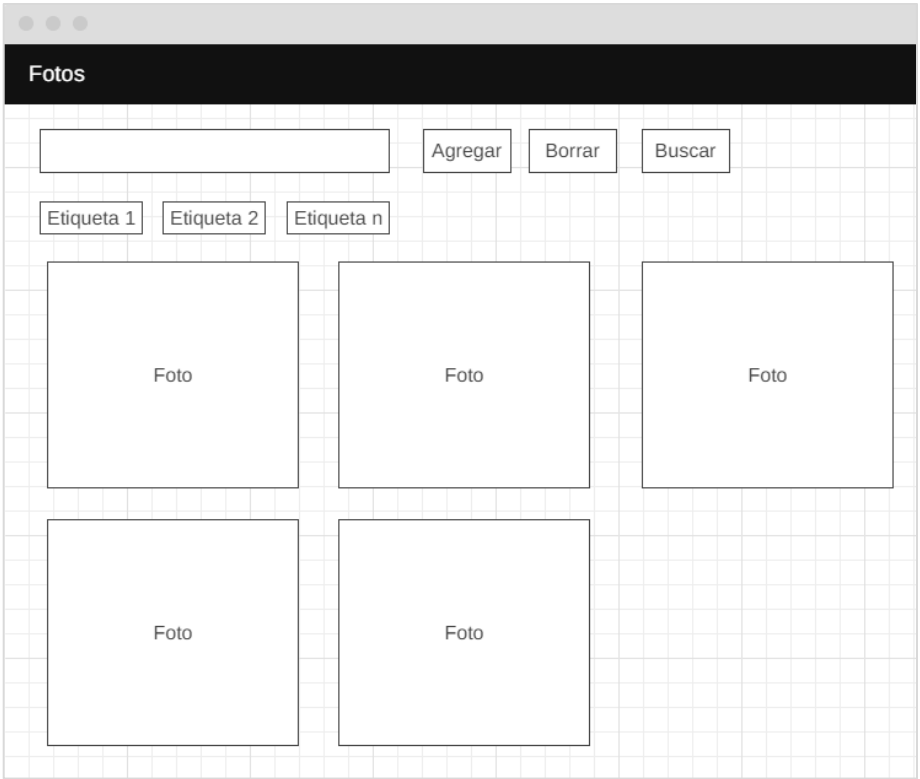

Figura 2.6. Diseño de interfaz de usuario

2.5. IMPLEMENTACIÓN

En esta sección se detalla la implementación del *back-end* y *front-end*. En el *back-end* se implementan los *stubs* que emulan los subsistemas de almacenamiento y clasificación, así como la lógica de consulta de fotografías. En el *front-end* se implementa el *stub* que emula el subsistema de adquisición de fotografías.

2.5.1. BACK-END

Creación del proyecto en Visual Studio

La creación del proyecto se realiza a través de la herramienta Microsoft Visual Studio (VS). Mediante el *wizard* de VS, y con base en una plantilla denominada ASP.NET Core web API se crea el proyecto (ver Figura 2.7). El *wizard* de forma automática creará una solución, y dentro de la solución un proyecto.

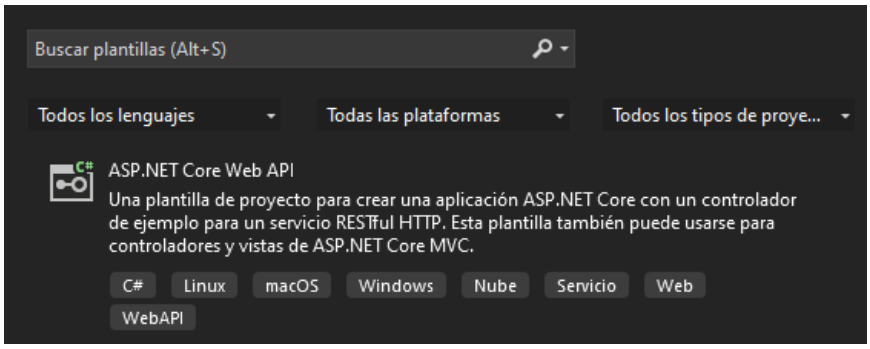

Figura 2.7. Plantilla ASP.NET Core web API

Posteriormente se asigna un nombre al proyecto y a la solución, y se selecciona la ubicación en la que se almacenarán los archivos del proyecto.

Clases para gestionar entidades

Para gestionar de forma adecuada a las entidades, se crea una nueva carpeta en el proyecto y se le asigna el nombre `Entidades`. En esta carpeta se almacenan las clases para gestionar las entidades requeridas. Con base en el diagrama entidad-relación (ver Figura 2.5), se generan las clases: `Foto`, `Etiqueta` y `FotoEtiqueta`.

En la Figura 2.8 se presenta la clase `Foto`.

```
public class Foto
{
    [Key]
    public int Id { get; set; }

    public string? Nombre { get; set; }

    public List<FotoEtiqueta>? FotoEtiquetas { get; set; }
}
```

Figura 2.8. Clase `Foto`

En la Figura 2.9 se presenta la clase `Etiqueta`.

```
public class Etiqueta
{
    [Key]
    public string? Nombre { get; set; }

    public List<FotoEtiqueta>? FotoEtiquetas { get; set; }
}
```

Figura 2.9. Clase `Etiqueta`

En la Figura 2.10 se presenta la clase `FotoEtiqueta`.

```
public class FotoEtiqueta
{
    [Key]
    public int FotoId { get; set; }

    [Key]
    public string? EtiquetaNombre { get; set; }

    public string? Ubicacion { get; set; }

    public Foto? Foto { get; set; }

    public Etiqueta? Etiqueta { get; set; }

}
```

Figura 2.10. Clase `FotoEtiqueta`

La etiqueta `Key` empleada en las clases presentadas en la Figura 2.8, Figura 2.9 y Figura 2.10, sirve para especificar que ese atributo corresponde a la clave primaria. La relación muchos a muchos se representa mediante instancias de las clases `Foto` y `Etiqueta` en la clase `FotoEtiqueta`, y mediante instancias de tipo `FotoEtiqueta` almacenadas en una lista en las clases `Foto` y `Etiqueta`.

Clase para realizar DTO

Se implementa la clase `FotoEtiquetaDTO` que tiene el propósito de almacenar la información de las fotografías provenientes de la consulta al *stub* del subsistema de almacenamiento y a su vez permitir transmitir tal información al *stub* del subsistema de adquisición de fotografías. En la Figura 2.11 se presenta la clase `FotoEtiquetaDTO`.

```
public class FotoEtiquetaDTO
{
    public int? FotoId { get; set; }

    public string? EtiquetaNombre { get; set; }

    public string? Ubicacion { get; set; }
}
```

Figura 2.11. Clase `FotoEtiquetaDTO`

Integración de AutoMapper

Para procesar la información entre una clase gestora de entidad y una clase para realizar DTO se emplea la librería AutoMapper. Para integrar AutoMapper a la web API es necesario agregar el código mostrado en la Figura 2.12 al archivo `Program.cs`.

```
builder.Services.AddAutoMapper(typeof(Program));
```

Figura 2.12. Integración de AutoMapper

Adicionalmente, se requiere crear en el proyecto una clase llamada `AutoMapperProfiles` que será la encargada de la configuración de AutoMapper. La clase debe derivarse de la interfaz `Profile` y definir las entidades involucradas en el procesamiento de información con la sentencia `CreateMap`, además se emplea el método `ReverseMap` que permite la conversión de información en los dos sentidos entre las entidades `FotoEtiqueta` y `FotoEtiquetaDTO`, tal y como se muestra en la Figura 2.13

```
public class AutoMapperProfiles : Profile
{
    public AutoMapperProfiles()
    {
        CreateMap<FotoEtiqueta, FotoEtiquetaDTO>().ReverseMap();
    }
}
```

Figura 2.13. Configuración de AutoMapper

Stub que emula el subsistema de almacenamiento

Para gestionar de forma adecuada la lógica de almacenamiento, se crea una nueva carpeta en el proyecto y se le asigna el nombre `Almacenamiento`. Dentro de la carpeta se crea

una clase llamada `ApplicacionDbContext`, la cual será la pieza central que emplee Entity Framework para gestionar la conexión e interacción con la base de datos.

La clase debe derivarse de la interfaz `DbContext` y se requiere de un constructor con parámetros de configuración, tal y como se muestra en la Figura 2.14.

```
public class ApplicationDbContext : DbContext
{
    public ApplicationDbContext(DbContextOptions options) : base(options)
    {
    }
}
```

Figura 2.14. Estructura básica de la clase `ApplicationDbContext`

Además, esta clase se puede usar para configurar la base de datos usando Entity Framework. Para gestionar las entidades a través de la clase `ApplicationDbContext`, se utiliza la clase `DbSet` en cada entidad (ver Figura 2.15)

```
public class ApplicationDbContext : DbContext
{
    public ApplicationDbContext(DbContextOptions options) : base(options)
    {
    }

    public DbSet<Foto> Fotos { get; set; }
    public DbSet<Etiqueta> Etiquetas { get; set; }
    public DbSet<FotoEtiqueta> FotoEtiquetas { get; set; }
}
```

Figura 2.15. Entidades en la clase `ApplicationDbContext`

Para construir e inicializar el modelo de la base de datos se puede implementar el método `OnModelCreating` de la clase `ApplicationDbContext`, en el cual se define que la llave primaria de la entidad `FotoEtiqueta` es la llave primaria compuesta por los atributos `FotoId` y `EtiquetaNombre`. En la Figura 2.16 se presenta el código implementado.

```
public class ApplicationDbContext : DbContext
{
    public ApplicationDbContext(DbContextOptions options) : base(options)
    {
    }

    public DbSet<Foto> Fotos { get; set; }
    public DbSet<Etiqueta> Etiquetas { get; set; }
    public DbSet<FotoEtiqueta> FotoEtiquetas { get; set; }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<FotoEtiqueta>()
            .HasKey(x => new { x.FotoId, x.EtiquetaNombre });
        base.OnModelCreating(modelBuilder);
    }
}
```

Figura 2.16. Configuración de llave primaria compuesta

Además, se requiere que la base de datos contenga información inicial. Para ello se carga en la base de datos las fotografías con los nombres `foto_1.jpg` y `foto_2.png`; y las etiquetas con los nombres `Etiqueta 1`, `Etiqueta 2` y `General`. La correspondencia entre fotos y etiquetas se detalla en la Tabla 2.1.

Tabla 2.1. Correspondencia entre fotos y etiquetas

Foto	Etiquetas
foto_1.jpg	General Etiqueta1
foto_2.png	General Etiqueta1 Etiqueta2

Se puede cargar la información inicial según lo descrito en la Tabla 2.1 a través del método `OnModelCreating`, en el cual se especifica para cada entidad la información que será ingresada a la base de datos. En la Figura 2.17 se presenta el código para insertar la información inicial en la base de datos para las entidades `Foto`, `Etiqueta` y `FotoEtiqueta` respectivamente a través de la creación de clases.

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<FotoEtiqueta>()
        .HasKey(x => new { x.FotoId, x.EtiquetaNombre });
    base.OnModelCreating(modelBuilder);

    // Foto
    modelBuilder.Entity<Foto>().HasData(
        new Foto { Id = 1, Nombre = "foto_1.jpg" },
        new Foto { Id = 2, Nombre = "foto_2.png" }
    );

    // Etiqueta
    modelBuilder.Entity<Etiqueta>().HasData(
        new Etiqueta { Nombre = "General" },
        new Etiqueta { Nombre = "Etiqueta1" },
        new Etiqueta { Nombre = "Etiqueta2" }
    );

    // FotoEtiqueta
    modelBuilder.Entity<FotoEtiqueta>().HasData(
        new FotoEtiqueta { FotoId = 1, EtiquetaNombre = "General", Ubicacion = "https://localhost:7041/General/foto_1.jpg" },
        new FotoEtiqueta { FotoId = 1, EtiquetaNombre = "Etiqueta1", Ubicacion = "https://localhost:7041/Etiqueta1/foto_1.jpg" },
        new FotoEtiqueta { FotoId = 2, EtiquetaNombre = "General", Ubicacion = "https://localhost:7041/General/foto_2.png" },
        new FotoEtiqueta { FotoId = 2, EtiquetaNombre = "Etiqueta1", Ubicacion = "https://localhost:7041/Etiqueta1/foto_2.png" },
        new FotoEtiqueta { FotoId = 2, EtiquetaNombre = "Etiqueta2", Ubicacion = "https://localhost:7041/Etiqueta2/foto_2.png" }
    );
}
```

Figura 2.17. Inserción inicial de información

Para poder establecer conexión con la base de datos, es necesario agregar en el archivo de configuración `Program.cs` el código de la Figura 2.18, mediante el cual se especifica

que se emplee la clase `ApplicationDbContext` para generar la base de datos denominada `TIC` con su estructura definida, y se especifica la cadena de conexión.

```
builder.Services.AddDbContext<ApplicationDbContext>(options =>
    options.UseSqlServer("Data Source=.;Initial Catalog=TIC;Integrated Security=True")
);
```

Figura 2.18. Configuración de conexión con una base de datos llamada `TIC`

Finalmente, se ejecuta el comando `Add-Migration inicial` para crear el esquema de la base de datos y asignarle el nombre `inicial`, luego se ejecuta el comando `Update-Database` para generar la estructura de la base de datos. Se puede comprobar la correcta creación de la base de datos a través de Microsoft SQL Server.

S*tub* que emula el subsistema de clasificación

Se crea una carpeta en la solución generada en Visual Studio llamada `wwwroot`, que será usada para almacenar el conjunto de directorios y las fotografías. En la Figura 2.19 se presentan las dos fotografías iniciales escogidas.

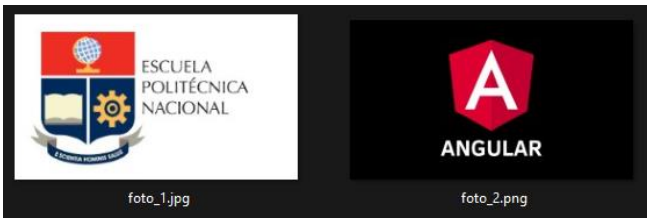

Figura 2.19. Fotografías para realizar pruebas

Dentro del directorio `wwwroot`, con base en lo descrito en la Tabla 2.1, se crean los directorios necesarios definidos en la Figura 2.20 y se guardan las fotografías en los directorios correspondientes.

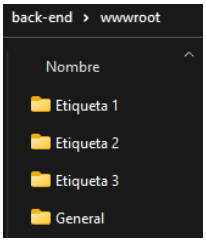

Figura 2.20. Directorios creados

Para que la web API trabaje con archivos estáticos (fotografías), se requiere editar el archivo `Program.cs` e introducir el código mostrado en la Figura 2.21.

```
app.UseStaticFiles();
```

Figura 2.21. Configuración de archivos estáticos

Servicio

Para gestionar de forma adecuada a los servicios, se crea una nueva carpeta en el proyecto y se le asigna el nombre `Servicios`. En la carpeta creada, se agrega una nueva clase llamada `FotosService`. Esta clase va a contener toda la lógica de consulta de fotografías. El controlador va a consumir los métodos de esta clase para procesar las peticiones de consulta. En la Figura 2.22 se presenta la clase `FotosService`.

```
public class FotosService
{
    public FotosService()
    {

    }
}
```

Figura 2.22. Estructura básica de la clase `FotosService`

En la clase `FotosService` se debe incluir la clase `ApplicationDbContext` así como la clase `IMapper` que permite emplear AutoMapper, y en el constructor se los debe inicializar, tal y como se muestra en la Figura 2.23.

```
public class FotosService
{
    private readonly ApplicationDbContext context;
    private readonly IMapper mapper;

    public FotosService(ApplicationDbContext context, IMapper mapper)
    {
        this.context = context;
        this.mapper = mapper;
    }

}
```

Figura 2.23. Clase `FotosService`

Controlador

Para almacenar los archivos de las clases que son controladores, se crea una nueva carpeta en el proyecto y se le asigna el nombre `Controladores`. Posteriormente en esta carpeta, se crea la clase `FotosController` en la que se implementarán los *endpoints* requeridos.

Para que la clase se comporte como un controlador y sea capaz de procesar peticiones, se requiere que la clase se derive de la interfaz `ControllerBase` y se añada la etiqueta `[Route(URL)]` a la clase para definir la URL que permitirá asociar dicha URL con el controlador, tal y como se muestra en la Figura 2.24.

```
[Route("api/fotos")]
public class FotosController : ControllerBase
{

}
```

Figura 2.24. Estructura básica de un controlador

El controlador debe implementar un objeto de la clase `FotoService`, y en su constructor debe inicializarse el mismo, como se muestra en la Figura 2.25.

```
[Route("api/fotos")]
public class FotosController : ControllerBase
{
    private readonly FotosService fotosService;

    public FotosController(FotosService fotosService)
    {
        this.fotosService = fotosService;
    }
}
```

Figura 2.25. Clase `FotosController`

Lógica de consulta de todas las fotografías

En la clase `FotosService` se crea un método que permita consultar a la base de datos la información de todas las fotografías almacenadas, mediante el código que se muestra en la Figura 2.26.

```
public async Task<List<FotoEtiquetaDTO>> ObtenerTodo()
{
    var fotoEtiquetas = await context.FotoEtiquetas.Where(fe => fe.EtiquetaNombre.Equals("General")).ToListAsync();
    return mapper.Map<List<FotoEtiquetaDTO>>(fotoEtiquetas);
}
```

Figura 2.26. Método para obtener todas las fotografías de la base de datos

El método `ObtenerTodo` interactúa con la base de datos a través del objeto `context`, para acceder a `FotoEtiquetas`, y mediante este realizar una consulta en la tabla `FotoEtiquetas` donde el atributo `EtiquetaNombre` corresponda al valor “General”. También, se realiza el procesamiento de información mediante AutoMapper entre las clases `FotoEtiqueta` y `FotoEtiquetaDTO`, retornando una lista de objetos de tipo `FotoEtiquetaDTO`. Además, el método emplea las palabras clave `async` y `await` para especificar que realiza una tarea asíncrona, es decir, una tarea que se va a ejecutar de forma paralela, por lo cual, para modelar la operación asíncrona, también es necesario emplear el objeto `Task<>` en el tipo de objeto que retorna el método.

A continuación, se debe definir el *endpoint* que permita atender los pedidos que los clientes realicen mediante el método HTTP GET que permita consultar todas las fotografías. Este *endpoint* llama al método `ObtenerTodo` a través de la instancia de la clase `FotosService`, como se muestra en la Figura 2.27.

```
[HttpGet]
public async Task<ActionResult<List<FotoEtiquetaDTO>>> ObtenerTodo()
{
    return await fotosService.ObtenerTodo();
}
```

Figura 2.27. *Endpoint* para consulta de todas las fotografías

Lógica de consulta de fotografías por etiquetas

En la clase `FotosService` se crea un método que permita consultar a la base de datos la información de las fotografías almacenadas con base en etiquetas definidas, mediante el código que se presenta en la Figura 2.28. El método `ObtenerPorEtiquetas` procesa las etiquetas remitidas en una variable de tipo `String` para que sean almacenadas en un arreglo de objetos de tipo `String`, para posteriormente iterar el arreglo y realizar la consulta de información de fotografías mediante el objeto `context` para cada una de las etiquetas definidas en la petición. Cada uno de los resultados de la consulta se almacena

en una lista de identificadores, la cual sirve para almacenar el atributo `FotoId` de las fotografías que correspondan a cierta etiqueta, luego se compara la lista de con cada resultado obtenido de la consulta a la base de datos, de forma que si un identificador está presente en todas las iteraciones de consulta implica que la fotografía con ese identificador corresponde a todas las etiquetas, por tanto, con cada iteración solo se conservan las fotografías que corresponde a todas la etiquetas. También realiza el procesamiento de información mediante AutoMapper entre las clases `FotoEtiqueta` y `FotoEtiquetaDTO`, retornando una lista de objetos de tipo `FotoEtiquetaDTO`. Además, el método emplea las palabras clave `async`, `await` y `Task<>` para especificar que realiza una tarea asíncrona.

```
public async Task<List<FotoEtiquetaDTO>> ObtenerPorEtiquetas(string etiquetas)
{
    var etiquetasArray = etiquetas.Split(",");

    List<FotoEtiqueta> listaFinal = new List<FotoEtiqueta>();

    for (int i = 0; i < etiquetasArray.Count(); i++)
    {
        var fotoEtiquetas = await context.FotoEtiquetas.Where(fe => fe.EtiquetaNombre.Equals(etiquetasArray[i])).ToListAsync();

        List<int> fotosIds = new List<int>();

        foreach (var f in fotoEtiquetas)
        {
            fotosIds.Add(f.FotoId);
        }

        if (i == 0)
        {
            listaFinal = fotoEtiquetas;
        }

        listaFinal = listaFinal.Where(f => fotosIds.Contains(f.FotoId)).ToList();

        fotosIds.Clear();

    }

    return mapper.Map<List<FotoEtiquetaDTO>>(listaFinal);
}
```

Figura 2.28. Método para consultar y procesar fotografías por etiquetas

A continuación, se debe definir el *endpoint* que permita atender los pedidos que los clientes realicen mediante el método HTTP GET que permita consultar las fotografías con base en etiquetas. Este *endpoint* llama al método `ObtenerPorEtiquetas` a través de la instancia de la clase `FotosService`, como se muestra en la Figura 2.29.

```
[HttpGet("{etiquetas}")]
public async Task<ActionResult<List<FotoEtiquetaDTO>>> ObtenerPorEtiquetas(string etiquetas)
{
    return await fotosService.ObtenerPorEtiquetas(etiquetas);
}
```

Figura 2.29. *Endpoint* para consultar fotografías por etiquetas

Configuración de permisos de dominio

Para que exista comunicación entre el *front-end* y el *back-end*, es necesario configurar permisos de dominio en la web API. Se puede conceder los permisos de dominio en el archivo `Program.cs` considerando la URL del *front-end*, tal y como se muestra en la Figura 2.30.

```
builder.Services.AddCors(options =>
{
    var frontendURL = "http://localhost:4200";
    options.AddDefaultPolicy(builder =>
    {
        builder.WithOrigins(frontendURL).AllowAnyMethod().AllowAnyHeader();
    });
});
```

Figura 2.30. Configuración de permisos de comunicación

En el código se configura la autorización basada en políticas por medio del método `AddDefaultPolicy`, en el cual se configuran los permisos para autorizar cualquier método HTTP y cualquier *header* proveniente del `front-end` mediante los métodos `AllowAnyMethod` y `AllowAnyHeader` respectivamente.

2.5.2. *STUB* PARA EL FRONT-END

Creación del proyecto en Angular

Para la implementación de un proyecto con Angular se utilizan los comandos descritos en la Tabla.1.2.

Se crea un nuevo proyecto llamado `front-end` a través de la ejecución del comando `ng new front-end` en una consola.

Entidades

Para gestionar de forma adecuada a las entidades, en el proyecto para Angular, se crea una nueva carpeta y se le asigna el nombre `modelos`. Dentro de la capeta, se crea la clase `FotoEtiqueta` con los mismos atributos que la clase `FotoEtiquetaDTO` definida en el *back-end* (ver Figura 2.11). En la Figura 2.31 se presenta la clase `FotoEtiqueta`.

```
export class FotoEtiqueta {
  public fotoId?: number;
  public etiquetaNombre?: string;
  public ubicacion?: string;
}
```

Figura 2.31. Clase `FotoEtiqueta`

Componente

Mediante el comando `ng g c listado_fotos` se crea el componente llamado `listado-fotos`. Este componente será el encargado de la visualización de fotografías y el funcionamiento de la lógica de consulta de fotografías.

Por medio de HTML y CSS se puede implementar la interfaz de usuario esquematizada en la Figura 2.6. La interfaz implementada se presenta en la Figura 2.32.

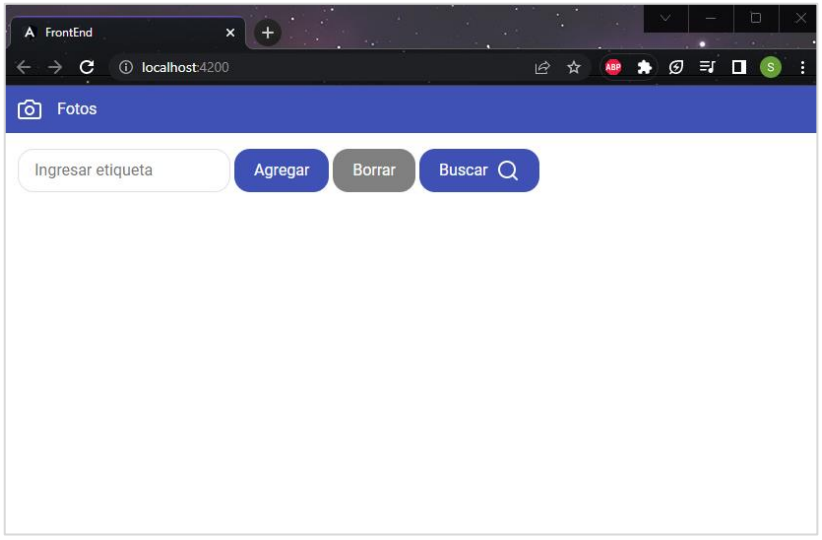

Figura 2.32. Vista del componente `listado-fotos`

Cliente HTTP

Para que la aplicación construida con Angular (*front-end*) pueda interactuar con la web API (*back-end*) mediante los *endpoints*, se puede hacer uso de cliente HTTP. Para que el cliente pueda interactuar con la web API mediante peticiones HTTP, se requiere importar un módulo en la clase `AppModule` del proyecto, esto se puede hacer en el archivo `app.module.ts` (ver Figura 2.33). Las líneas enmarcadas en rojo son las que deben ser agregadas para importar el módulo HTTP.

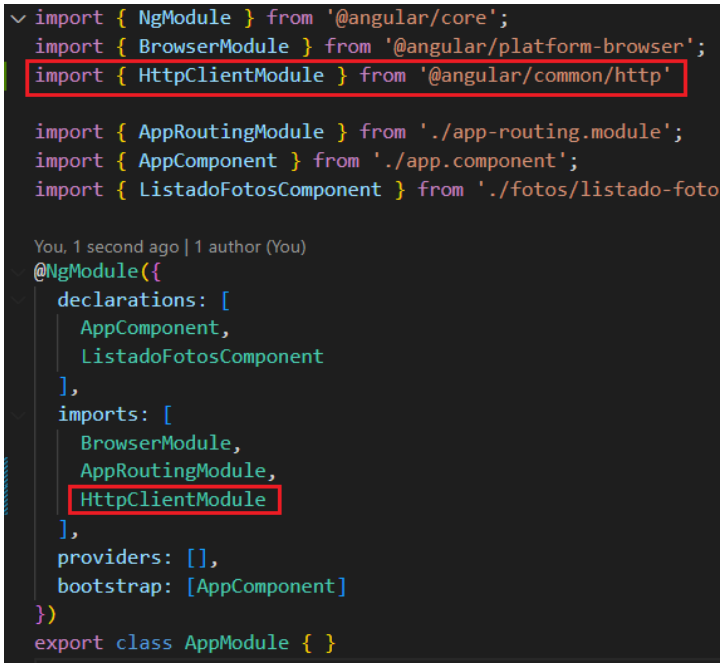

Figura 2.33. Importación de módulo HttpClient

Mediante el comando `ng g s fotos` se crea un servicio llamado `fotos`, cuyo resultado es disponer de una clase denominada `FotoService`.

En la clase `FotosService` se debe instanciar el cliente HTTP, para lo cual se puede usar el constructor de la clase; adicionalmente se puede generar una variable (`apiURL`) con la URL del *back-end*, tal y como se muestra en la Figura 2.34.

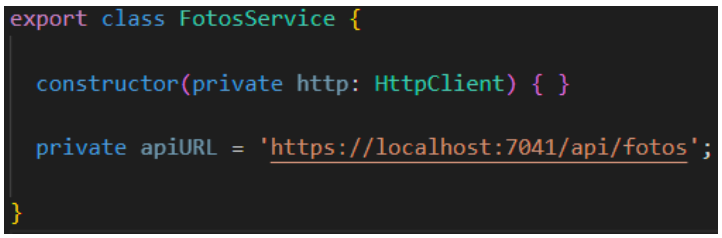

Figura 2.34. Estructura básica de servicio

Lógica de consulta de todas las fotografías

El método `obtenerTodo` del servicio `FotosService` permite recuperar todas las fotografías almacenadas en el *back-end*. Este método realizará una petición HTTP GET a la URL establecida para obtener todas las fotografías, luego de lo cual las almacena en un arreglo de objetos de tipo `FotoEtiqueta`, tal y como se muestra en la Figura 2.35.

```
public obtenerTodo(): Observable<FotoEtiqueta[]> {
  return this.http.get<FotoEtiqueta[]>(this.apiURL);
}
```

Figura 2.35. Método para obtener todas las fotografías

En el componente `ListadoFotosComponent` (ver Figura 2.36) se debe instanciar el servicio `FotosService` en el constructor de la clase; por otro lado, a nivel del componente se crea un arreglo de objetos de tipo `FotoEtiqueta` denominado `fotos` para almacenar la información de las fotografías. Finalmente, en el método `OnInit` se llama al método `obtenerTodo` del servicio `FotosService` y se almacena su respuesta en la variable `fotos`.

```
export class ListadoFotosComponent implements OnInit {
  fotos: FotoEtiqueta[];

  constructor(private fotosService: FotosService) { }

  ngOnInit(): void {
    this.fotosService.obtenerTodo().subscribe(res => {
      this.fotos = res;
    });
  }
```

Figura 2.36. Llamada al método `obtenerTodo` en el componente `listado-fotos`

Por otro lado, en el archivo `listado-fotos.component.html`, se presentarán las fotografías almacenadas en la variable `fotos`, para ello se itera el contenido de la variable `fotos` por medio de la propiedad `*ngFor` y se carga una imagen en un elemento HTML de tipo `img` especificando la ruta de la fotografía por medio del atributo `ubicacion`, como se muestra en la Figura 2.37.

```
<div class="grid">
    <div *ngFor="let foto of fotos">
        <img [src]="foto.ubicacion" class="foto">
    </div>
</div>
```

Figura 2.37. Presentación de las fotografías

Al ingresar a la aplicación de *front-end* mediante un explorador web, en el cual se coloque como URL `localhost:4200`, si todo está correcto, se presentará por defecto el listado de todas las fotografías almacenadas en el *back-end*, como se aprecia en la Figura 2.38.

Figura 2.38. Página inicial del *stub* de consulta

Lógica de consulta de fotografías por etiquetas

El método `buscarPorEtiquetas` del servicio `FotosService` permite recuperar todas las fotografías almacenadas en el *back-end* con base en un conjunto de etiquetas definidas. Este método realizará una petición HTTP GET a la URL establecida para obtener todas las fotografías filtradas por etiquetas, luego de lo cual las almacena en un arreglo de objetos de tipo `FotoEtiqueta`, tal y como se muestra en la Figura 2.39.

```
public buscarPorEtiquetas(etiquetas: string[]): Observable<FotoEtiqueta[]> {
  return this.http.get<FotoEtiqueta[]>(`${this.apiURL}/${etiquetas}`);
}
```

Figura 2.39. Método para obtener fotografías por etiquetas

En el componente `ListadoFotosComponent` (ver Figura 2.40) a nivel del componente se crea un arreglo de objetos de tipo `String` denominado `etiquetas` para almacenar las etiquetas definidas por el usuario, también se crea la variable `nombreEtiqueta` para almacenar el valor de etiqueta que será ingresado a través de un elemento *input*.

```typescript
export class ListadoFotosComponent implements OnInit {
  fotos: FotoEtiqueta[];
  etiquetas: string[] = [];
  nombreEtiqueta: string;

  constructor(private fotosService: FotosService) { }

  ngOnInit(): void {
    this.fotosService.obtenerTodo().subscribe(res => {
      this.fotos = res;
    });
  }

  agregarEtiqueta(etiqueta: any): void {
    if (etiqueta) {
      this.etiquetas.push(etiqueta);
      this.nombreEtiqueta = '';
    }
  }

  borrarEtiquetas(): void {
    this.etiquetas = [];
  }

  buscar(): void {
    this.fotosService.buscarPorEtiquetas(this.etiquetas).subscribe(res => {
      this.fotos = res;
      this.nombreEtiqueta = '';
    });
  }
}
```

Figura 2.40. Métodos para gestionar etiquetas

Adicionalmente, se implementan los métodos `agregarEtiqueta` para añadir una etiqueta al arreglo `etiquetas`, `borrarEtiquetas` para vaciar el arreglo `etiquetas` y `buscar` el cual llama al método `buscarPorEtiquetas` del servicio `FotosService`, proporciona como argumento el arreglo `etiquetas` y almacena su respuesta en la variable `fotos`.

En el archivo `listado-fotos.component.html`, asociar los métodos `agregarEtiqueta`, `borrarEtiquetas` y `buscar` al evento click de los botones Agregar, Borrar y Buscar respectivamente. El código se presenta en la Figura 2.41.

```html
<div class="d-flex gap-1">
    <input
        id="buscar-etiqueta"
        [(ngModel)]="nombreEtiqueta"
        type="text"
        class="input-buscar input-border"
        placeholder="Ingresar etiqueta"
    >
    <button class="btn btn-primario" (click)="agregarEtiqueta(nombreEtiqueta)">Agregar</button>
    <button class="btn btn-secundario" (click)="borrarEtiquetas()">Borrar</button>
    <button class="flex-content-center gap-05 btn btn-primario" (click)="buscar()">
        Buscar
        <img src="../../../assets/imagenes/search.svg">
    </button>
</div>
```

Figura 2.41. Código HTML para interacción de etiquetas

Adicionalmente, en el archivo `listado-fotos.component.html`, se itera el arreglo `etiquetas` mediante la propiedad `*ngFor` para que las etiquetas se muestren en la interfaz como botones. El código se presenta en la Figura 2.42.

```
<div class="contenedor-etiquetas d-flex gap-1">
    <div *ngFor="let etiqueta of etiquetas">
        <button class="btn-etiqueta"> {{etiqueta}} </button>
    </div>
</div>
```

Figura 2.42. Código HTML para mostrar etiquetas

Al ingresar a la aplicación de *front-end* mediante un explorador web, en el cual se coloque como URL `localhost:4200`. Si se introduce las etiquetas `Etiqueta1` y `Etiqueta2`, si todo está correcto, se presentará la fotografía llamada `foto_2.png`, la cual corresponde a todas las etiquetas definidas, tal como se aprecia en la Figura 2.43.

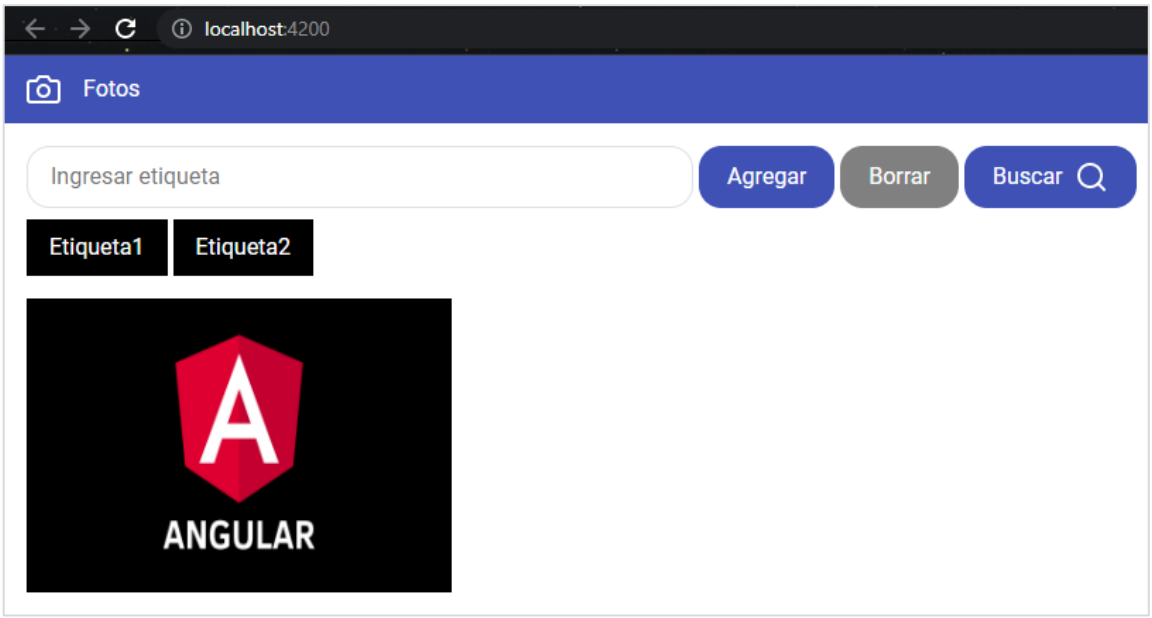

Figura 2.43. Resultado de consulta por etiquetas

2.5.3. PRUEBAS

En esta sección se describen las pruebas de funcionalidad del prototipo del subsistema de consulta, las cuales fueron realizadas por usuarios que probaron y evaluaron el funcionamiento del subsistema de consulta.

2.5.4. ENCUESTA

Se realizó una encuesta y se la aplicó a cinco usuarios, a quienes previamente se les pidió que empleen el subsistema y a través de la encuesta evalúen su funcionamiento. Con base en los resultados se realizaron correcciones para disponer de un trabajo funcional. Las preguntas de la encuesta se presentan en la Figura 2.45.

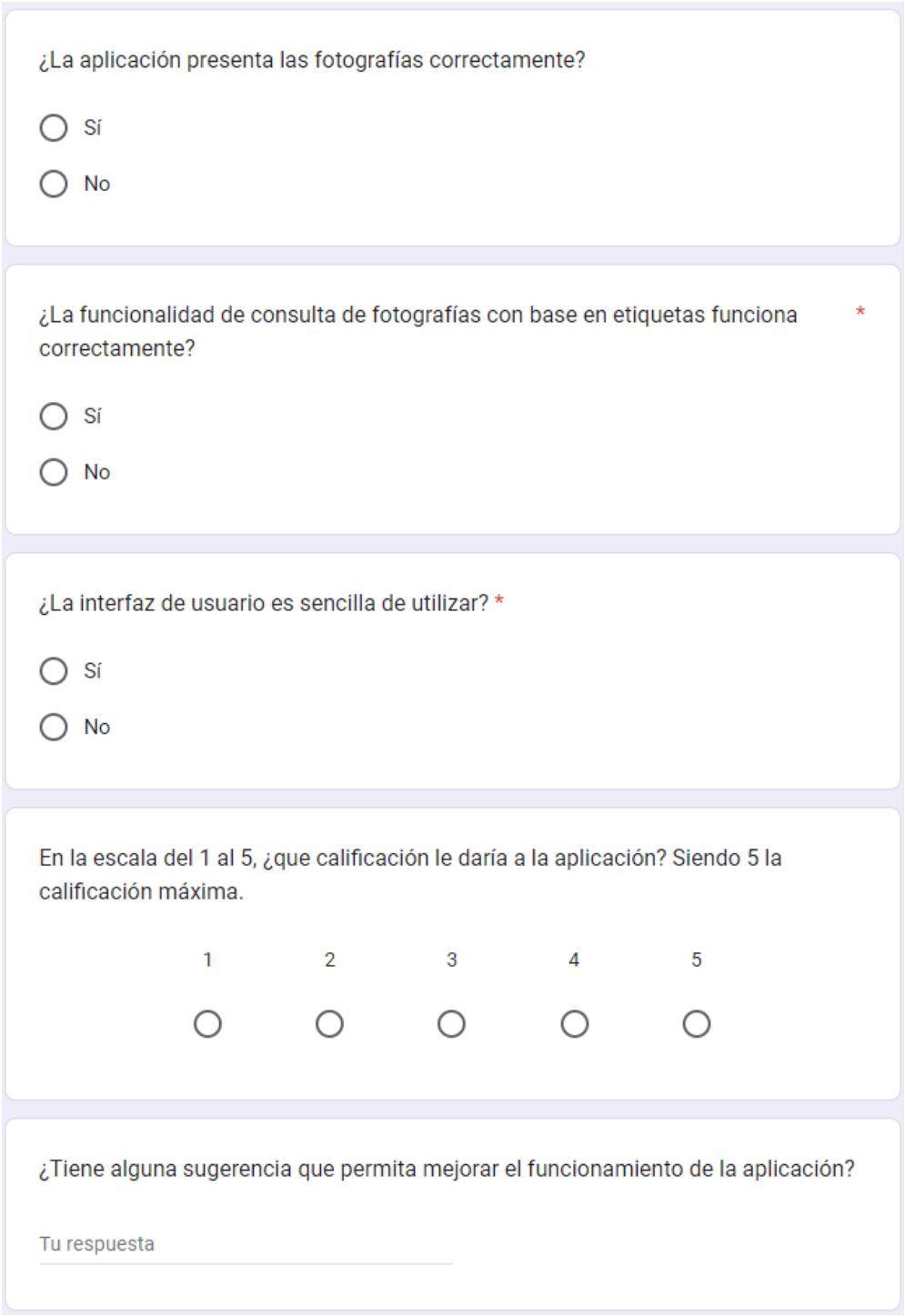

Figura 2.44. Encuesta de evaluación

3. RESULTADOS, CONCLUSIONES Y RECOMENDACIONES

Esta sección contiene los resultados, conclusiones y recomendaciones resultantes tras finalizar el prototipo del subsistema de consulta.

3.1. RESULTADOS

Como resultado de este trabajo se dispone de un subsistema de consulta que permite recuperar fotografías con base en etiquetas. Las pruebas de usuario para el subsistema de consulta y los *stub* han sido exitosas. Los resultados de la encuesta realizada a los cinco usuarios se presentan en esta sección.

En la Figura 3.2 se presenta el resultado de la primera pregunta de la encuesta, en la cual se aprecia que los cinco usuarios han respondido que las fotografías se presentan de forma correcta en la aplicación.

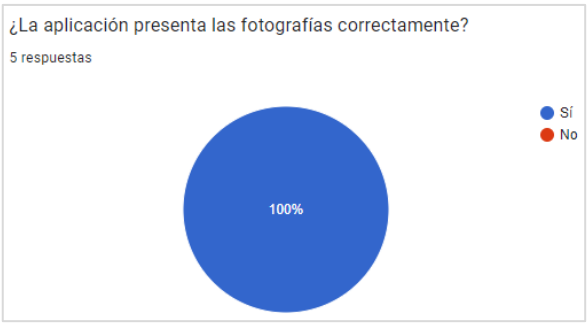

Figura 3.1. Resultado de la pregunta 1 de la encuesta de evaluación

En la Figura 3.3 se presenta el resultado de la segunda pregunta de la encuesta, en la cual se aprecia que los cinco usuarios han respondido que la funcionalidad de consulta de fotografías por etiquetas funciona correctamente.

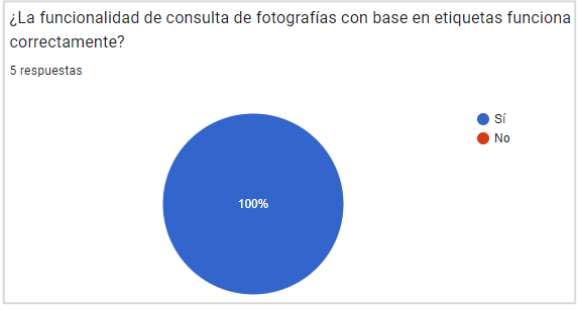

Figura 3.2. Resultado de la pregunta 2 de la encuesta de evaluación

En la Figura 3.4 se presenta el resultado de la tercera pregunta de la encuesta, en la cual se aprecia que los cinco usuarios han respondido que la interfaz de usuario es sencilla de utilizar.

Figura 3.3. Resultado de la pregunta 3 de la encuesta de evaluación

En la Figura 3.5 se presenta el resultado de la cuarta pregunta de la encuesta, en la cual se pide al usuario que califique el desempeño de la aplicación en una escala del 1 al 5. Con base en el resultado se tiene un promedio de calificación de 4,6.

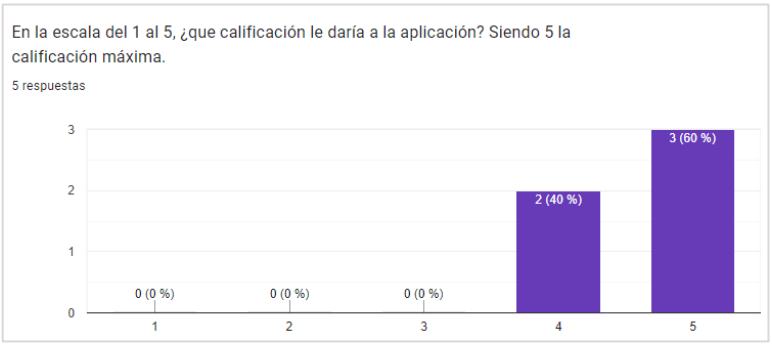

Figura 3.4. Resultado de la pregunta 4 de la encuesta de evaluación

En la Figura 3.6 se presenta el resultado de la quinta pregunta de la encuesta, en la cual se pide al usuario que proporcione alguna sugerencia de mejora de la aplicación, en el caso que la tenga. Al ser una pregunta opcional dos usuarios han presentado sus comentarios sin ninguna sugerencia relevante.

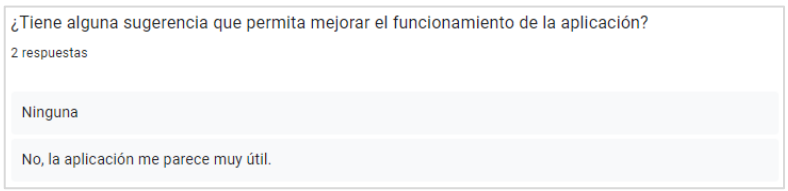

Figura 3.5. Resultado de la pregunta 5 de la encuesta de evaluación

Con base en los resultados obtenidos de las encuestas de evaluación realizadas por los cinco usuarios, se puede llegar a la conclusión de que el desarrollo del prototipo del subsistema de consulta ha sido exitoso, ya que cumple con su propósito y presenta un desempeño sobresaliente. Para este caso no ha sido necesario realizar corrección de errores ya que los usuarios no reportaron ninguna observación.

3.2. CONCLUSIONES

Las conclusiones obtenidas durante el desarrollo del subsistema de consulta se presentan a continuación:

- Como resultado de este trabajo se dispone de un *back-end* que tiene un servicio web implementado mediante ASP.NET web API, el cual permite consultar fotografías con base en etiquetas definidas por un usuario. Esto significa que el subsistema tiene la capacidad de buscar y mostrar fotografías especificas que han sido etiquetadas con palabras clave por el usuario. Este enfoque permite al usuario acceder rápidamente a las fotografías que busca en función de los temas específicos que son de su interés. En resumen, la implementación del subsistema de consulta ha resultado en un servicio web eficiente para la búsqueda de fotografías con base en etiquetas definidas por un usuario.
- El uso de herramientas como ASP.NET web API, Angular y Entity Framework permite simplificar y acelerar el proceso de desarrollo del subsistema de consulta en un proyecto de software. Estas herramientas proporcionan bases para la creación de aplicaciones web y bases de datos, lo que permite centrarse en la lógica del negocio y la funcionalidad del subsistema en lugar de preocuparse por la complejidad de la infraestructura subyacente. En conjunto, estas herramientas permiten disponer de un subsistema de consulta con mayor eficiencia y calidad, lo que puede traducirse en una mejor experiencia de usuario y un producto final más satisfactorio.
- Por medio de *stubs* se dispone de la emulación de los subsistemas de almacenamiento, clasificación y adquisición de fotografías; los cuales fueron implementados por medio de diferentes tecnologías e integrados al subsistema de consulta de forma correcta y eficiente, dando como resultado un prototipo de subsistema de consulta funcional que cumple con los objetivos planteados en el presente Trabajo de Integración Curricular.

- Con base en los resultados de las encuestas realizadas a los cinco usuarios y las pruebas realizadas a lo largo del desarrollo del prototipo del subsistema de consulta, se ha determinado que el prototipo funciona correctamente y presenta un desempeño sobresaliente. Sin embargo, es importante considerar que esta conclusión se basa en un número limitado de usuarios y pruebas, por lo cual es posible que el prototipo presente problemas o limitaciones que no hayan sido identificados en el estudio. Por lo tanto, es recomendable continuar evaluando y mejorando el prototipo en función de pruebas realizadas en un entorno más amplio.

3.3. RECOMENDACIONES

Durante el proceso de implementación del prototipo de subsistema de consulta de plantean las siguientes recomendaciones:

- Utilizar herramientas o *frameworks* que permitan simplificar el proceso de desarrollo del proyecto de software. Por ejemplo, se puede utilizar Entity Framework para generar y gestionar una base de datos a través de código y comandos sin necesidad de interactuar directamente con la misma. Por otro lado, se puede utilizar la herramienta Automapper para automatizar el procesamiento de información entre clases.
- Antes de implementar una base de datos relacional realizar un diagrama entidad-relación que permita describir y modelar de forma más concreta y especifica las entidades, atributos y relaciones que permitirán disponer del subsistema. Caso contrario se puede generar e implementar un modelo ineficiente con entidades o atributos irrelevantes y relaciones innecesarias.
- En el caso de emplear Entity Framework para gestionar una base de datos, se debe considerar que su configuración varía según el tipo de relación que exista entre entidades. En el caso del desarrollo del *stub* del subsistema de almacenamiento, se trabajó con una relación muchos a muchos, lo cual implicó una serie de consideraciones que había que tomar en cuenta para la generación adecuada del modelo en la base de datos.
- Al integrar una herramienta o *framework* adicional a la web API asegurase de realizar las respectivas configuraciones en el archivo de configuración `Program.cs`, con el fin de que la herramienta pueda integrarse al proyecto de

forma efectiva. Además, para el caso de la conexión del *front-end* con la web API (*back-end*) también se debe considerar configurar las políticas de dominio en el archivo de configuración mencionado, lo cual permite que la web API acepte peticiones provenientes del *front-end*.

- Considerar utilizar clases para realizar DTO para la transferencia de información entre distintas capas; ya que permite simplificar la complejidad del código, reducir la cantidad de datos transmitidos, mejorar el rendimiento y evita la exposición de las clases que interactúan directamente con la base de datos.

4. REFERENCIAS BIBLIOGRÁFICAS

[1]. NUCBA. "¿Qué es la arquitectura cliente-servidor?". NUCBA. Mayo, 27, 2021. Acceso: Diciembre, 3, 2022. [En línea]. Disponible en: https://nucba.medium.com/qu%C3%A9-es-la-arquitectura-cliente-servidor-eb9f402506cc

[2]. Álvarez. N. "¿Qué es MVC?". DesarrolloWeb. Julio, 28, 2020. Acceso: Diciembre, 10, 2022. [En línea]. Disponible en: https://desarrolloweb.com/articulos/que-es-mvc.html

[3]. Costanzo. M. "¿Qué es el patrón MVC?". Platzi. 2020. Acceso: Diciembre, 10, 2022. [En línea]. Disponible en: https://platzi.com/tutoriales/1248-pro-arquitectura/5466-que-es-el-patron-mvc/?utm_source=google&utm_medium=cpc&utm_campaign=19643931773&utm_adgroup=&utm_content=&gclid=CjwKCAiAuaKfBhBtEiwAht6H73wxro1Yb9aHnN-Yof_Rl3MoQxN4km9CxNTHlTWD8xfcUaaBpX496RoCsEwQAvD_BwE&gclsrc=aw.ds

[4]. Bello. E. "*Framework*: Qué es, para qué sirve y por qué deberías usarlo". IEBS. Diciembre, 27, 2021. Acceso: Diciembre, 20, 2022. [En línea]. Disponible en: https://www.iebschool.com/blog/framework-que-es-agile-scrum/

[5]. Microsoft. "Información general de ASP.NET". Microsoft. Octubre, 5, 2022. Acceso: Diciembre, 20, 2022. [En línea]. Disponible en: https://learn.microsoft.com/es-es/aspnet/overview

[6]. Carmona. J. "Qué es ASP.NET y cuáles son sus puntos fuertes". OpenWebinars. Marzo, 18, 2022. Acceso: Diciembre, 20, 2022. [En línea]. Disponible en: https://openwebinars.net/blog/que-es-aspnet-y-cuales-son-sus-puntos-fuertes/

[7]. Azabache. G. "Qué es REST, RESTful, API RESTful y JSON". Brave Developer. Septiembre, 1, 2021. Acceso: Diciembre, 23, 2022. [En línea]. Disponible en: https://bravedeveloper.com/2021/09/01/que-es-rest-restful-api-restful-y-json/

[8]. Chojrin. M. "Qué es una API REST (API RESTful)". Platzi. Acceso: Diciembre, 23, 2022. [En línea]. Disponible en: https://platzi.com/clases/1638-api-rest/21611-que-significa-rest-y-que-es-una-api-restful/

[9]. Yambadwar. S. "What is web API and why we use it?". Geeks for geeks. Mayo, 31, 2020. Acceso: Diciembre, 23, 2022. [En línea]. Disponible en: https://www.geeksforgeeks.org/what-is-web-api-and-why-we-use-it/

[10]. Gilibets. L. "Qué es la metodología Kanban y como utilizarla". IEBS. Enero, 12, 2022. Acceso: Diciembre, 29, 2022. [En línea]. Disponible en: https://www.iebschool.com/blog/metodologia-kanban-agile-scrum/#:~:text=Kanban%20es%20una%20metodolog%C3%ADa%20de,tableros%20kanban%20y%20tarjetas%20kanban.

[11]. Fonseca. L. "Cómo crear un diagrama de clases". VENNGAGE. Junio, 27, 2022. Acceso: Diciembre, 29, 2022. [En línea]. Disponible en: https://es.venngage.com/blog/diagrama-de-clases/

[12]. Lucidchart. "Qué es un diagrama entidad-relación". Lucidchart. Acceso: Diciembre, 29, 2022. [En línea]. Disponible en: https://www.lucidchart.com/pages/es/que-es-un-diagrama-entidad-relacion

[13]. Mishra. A. "*Stub* vs *Mock*". Geeks for geeks. Noviembre, 17, 2022. Acceso: Enero, 3, 2023. [En línea]. Disponible en: https://www.geeksforgeeks.org/stub-vs-mock/

[14]. Blancarte. O. "*Data Transfer Object* (DTO)". Oscar Blancarte. Noviembre, 30, 2018. Acceso: Enero, 3, 2023. [En línea]. Disponible en: https://www.oscarblancarteblog.com/2018/11/30/data-transfer-object-dto-patron-diseno/

[15]. Taylor. A. "¿Cómo crear puntos finales y por qué los necesita?". AppMaster. Marzo, 31, 2022. Acceso: Enero, 3, 2023. [En línea]. Disponible en: https://appmaster.io/es/blog/como-crear-puntos-finales-y-por-que-los-necesita

[16]. Khan. H. "AutoMapper *in* .NET Core". C# Corner. Agosto, 19, 2020. Acceso: Enero, 15, 2023. [En línea]. Disponible en: https://www.c-sharpcorner.com/article/automapper-in-net-core/

[17]. Musso. E. "Entity Framework *using C#?*". C# Corner. Octubre, 13, 2020. Acceso: Enero, 15, 2023. [En línea]. Disponible en: https://www.c-sharpcorner.com/article/entity-framework-introduction-using-c-sharp-part-one/

[18]. Microsoft. Instalación de Entity Framework Core. Microsoft. Septiembre, 28, 2022. Acceso: Enero, 15, 2023. [En línea]. Disponible en: https://learn.microsoft.com/es-es/ef/core/get-started/overview/install

[19]. Microsoft. Referencia de herramientas de Entity Framework *Core:* consola de Administrador de paquetes en Visual Studio. Microsoft. Enero, 20, 2022. Acceso: Enero, 15, 2023. [En línea]. Disponible en: https://learn.microsoft.com/es-es/ef/core/cli/powershell

[20]. Angular. "*What is Angular?*". Angular. Acceso: Enero, 29, 2023. [En línea]. Disponible en: https://angular.io/guide/what-is-angular

[21]. Gongalves. M. "¿Qué es Angular y para qué sirve?". Hiberus blog. Octubre, 10, 2021. Acceso: Enero, 29, 2023. [En línea]. Disponible en: https://www.hiberus.com/crecemos-contigo/que-es-angular-y-para-que-sirve/

[22]. Angular. "*Introduction to Angular concepts*". Angular. Acceso: Enero, 29, 2023. [En línea]. Disponible en: https://angular.io/guide/architecture

[23]. Álvarez. M. "Servicio en Angular". Servicios en Angular. Mayo, 14, 2020. Acceso: Enero, 29, 2023. [En línea]. Disponible en: https://desarrolloweb.com/articulos/servicios-angular.html

[24]. Gavilán. F. "Desarrollando Aplicaciones en Angular 12 y ASP.NET Core 5". Udemy. Marzo, 2022. Acceso: Noviembre, 1, 2022. [En línea]. Disponible en: https://www.udemy.com/course/desarrollando-aplicaciones-en-angular-y-aspnet-core/?utm_source=adwords&utm_medium=udemyads&utm_campaign=WebDevelopment_v.PROF_la.ES_cc.LATAM&utm_term=_._ag_122306547969_._ad_507479116473_._kw__._de_c_._dm__._pl__._ti_dsa-1190286622959_._li_9069516_._pd__._&matchtype=&gclid=EAIaIQobChMIlqTmyJeg_QIVx4FaBR3kpA5tEAAYASAAEgJEbvD_BwE

[25]. Básalo. A. "Comunicaciones http en Angular". Academia Binaria. Abril, 18, 2020. Acceso: Noviembre, 25, 2022. [En línea]. Disponible en: https://academia-binaria.com/comunicaciones-http-en-Angular/

[26]. Microsoft. Entity Framework Core*:* Relaciones. Microsoft. Acceso: Noviembre, 25, 2022. [En línea]. Disponible en: https://learn.microsoft.com/es-es/ef/core/modeling/relationships?tabs=fluent-api%2Cfluent-api-simple-key%2Csimple-key

[27]. Microsoft. Entity Framework Core*:* Relaciones, propiedades de navegación y claves externas. Microsoft. Acceso: Noviembre, 25, 2022. [En línea]. Disponible en: https://learn.microsoft.com/es-es/ef/ef6/fundamentals/relationships

[28]. Microsoft. *Routing in* ASP.NET Core. Microsoft. Acceso: Noviembre, 30, 2022. [En línea]. Disponible en: https://learn.microsoft.com/en-us/aspnet/core/fundamentals/routing?view=aspnetcore-7.0

5. ANEXOS

ANEXO I. Código.

ANEXO I. CODIGO

El código, debido a su tamaño, se incluye en el CD adjunto.