ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA

DESARROLLO DE UN SISTEMA DISTRIBUIDO PARA CLASIFICACIÓN DE FICHAS LEGO BASADO EN IMÁGENES

SUBSISTEMA DE ALMACENAMIENTO DE INFORMACIÓN

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

MIGUEL ALEXANDER GUTIÉRREZ LEÓN

miguel.gutierrez@epn.edu.ec

DIRECTOR: ING. RAÚL DAVID MEJÍA NAVARRETE, M.Sc.

david.mejia@epn.edu.ec

DMQ, febrero 2022

CERTIFICACIONES

Yo, Miguel Alexander Gutiérrez León 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.

MIGUEL ALEXANDER GUTIÉRREZ LEÓN

Certifico que el presente trabajo de integración curricular fue desarrollado por Miguel Alexander Gutiérrez León, bajo mi supervisión.

ING. RAUL DAVID MEJÍA NAVARRETE, M.Sc
DIRECTOR

DECLARACIÓN DE AUTORÍA

A través de la presente declaración, afirmamos que el trabajo de integración curricular aquí descrito, así como el producto resultante 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.

MIGUEL ALEXANDER GUTIÉRREZ LEÓN

ING. RAÚL DAVID MEJÍA NAVARRETE, M.Sc.

DEDICATORIA

A mis padres Juanita León y Ramiro Gutiérrez, así como a mis hermanos Carlos y Sofia, por todas sus palabras de apoyo.

Finalmente, a la Escuela Politécnica Nacional.

AGRADECIMIENTO

A mis papis y mis hermanos por apoyarme y motivarme a seguir adelante, con mis estudios en la Poli y en la vida.

Gracias por su paciencia, en las diversas actividades que tenido que hacer, y por los malos ratos que les hice pasar, no hay palabras para expresarles todo lo que siento.

A todos los grandes amigos que he hecho en la universidad, por su apoyo y compañía, durante los estudios y fiestas.

A todos mis profesores de la Escuela Politécnica Nacional, por sus enseñanzas.

A mi tutor, David Mejía, M.Sc., por su tiempo, apoyo y por toda la paciencia al dirigir este trabajo.

ÍNDICE DE CONTENIDO

RESUMEN

Para el presente Trabajo de Integración Curricular se realizó el diseño e implementación de un subsistema de almacenamiento para un sistema de clasificación de fichas Lego basado en imágenes.

El subsistema permite almacenar imágenes de fichas Lego que sean envías desde un subsistema que se encarga de clasificarlas permitiendo almacenar atributos tanto de la imagen como de la ficha Lego.

El presente documento se encuentra estructurado en 3 capítulos.

En el primer capítulo se presentan conceptos sobre las herramientas usadas, así como las diferentes tecnologías de software que fueron usadas, en este caso se menciona el funcionamiento de Entity Framework, así como el enfoque que fue usado *Database First*, al igual que de la metodología Kanban que fue aplicada en el desarrollo del subsistema.

En el segundo capítulo, se presenta el proceso de desarrollo del subsistema, así como el proceso seguido para generar las respectivas historias de usuario. Seguidamente se presenta el respectivo diagrama de entidad – relación para la base de datos, para posteriormente describir brevemente el proceso seguido para la implementación para las diversas funcionalidades del prototipo.

En el tercer capítulo se presentan las pruebas de funcionamiento realizadas, para cada uno de los elementos del subsistema, seguidamente se presentan las conclusiones y recomendaciones del presente Trabajo de Integración Curricular.

PALABRAS CLAVE: Entity Framework, Database First, Kanban.

ABSTRACT

For the present Curriculum Integration Work, the design and implementation of a storage subsystem for a Lego chip classification system based on images was carried out.

The subsystem allows you to store images of Lego chips that are sent from a subsystem that is responsible for classifying them, allowing you to store attributes of both the image and the Lego chip.

This document is structured in 3 chapters.

In the first chapter, concepts about the tools used are presented, as well as the different software technologies that were used, in this case the operation of the Entity Framework is mentioned, as well as the approach that was used Database First, as well as the methodology Kanban that was applied in the development of the subsystem.

In the second chapter, the development process of the subsystem is presented, as well as the process followed to generate the respective user stories. Next, the respective entity-relationship diagram for the database is presented, to subsequently briefly describe the process followed for the implementation of the various functionalities of the prototype.

In the third chapter, the performance tests carried out are presented for each of the elements of the subsystem, followed by the conclusions and recommendations of this Curriculum Integration Work.

KEYWORDS: Entity Framework, Database First, Kanban

1 INTRODUCCIÓN

Como parte del presente Trabajo de Integración Curricular se plantea el desarrollo de un subsistema de almacenamiento, que guarde diferentes atributos de imágenes que contengan una ficha Lego. Este subsistema es parte de un sistema de clasificación de fichas Lego mediante imágenes.

Para este subsistema se hará uso de Entity Framework, que es una herramienta que permitirá trabajar con la base de datos de forma indirecta, haciendo uso de un esquema orientado a objetos, así también se empleará una base de datos la misma que será implementada en SQL Server.

El subsistema es capaz de almacenar la información de las diferentes imágenes Lego, así como atributos propios de las fichas Lego que serán usados por un subsistema de consulta, el cual obtendrá la información que se requiera mostrar a través de consultas que contengan diferentes parámetros. Es por esto que, entre los atributos necesarios que se almacenaran están las características como el color de las fichas, la forma que tienen, el factor de la ficha, entre otras.

Para poder comprobar que el subsistema implementado cumple con los diferentes parámetros requeridos, se ha incluido un *stub*, que ayudará a simular el funcionamiento de un subsistema diferente y que muestra que efectivamente se está realizando el almacenamiento correspondiente de la información requerida, con esto se podrá integrar este subsistema con los diferentes subsistemas que requieran hacer uso de la base de datos, y al usar un ORM como Entity Framework, se podrán evitar conflictos con la conexión a la base de datos.

La información que se encuentra almacenada, podrá ser presentada ya sea que otro subsistema lo requiera o para verificar que efectivamente se está almacenando la información, para ello se puede hacer uso de diferentes herramientas para la generación de los *stub*, entre ellos se puede usar ASP.NET MVC, o una aplicación de Windows Form. Para este Trabajo de Integración Curricular se hará uso una aplicación ASP.NET MVC para verificar el almacenamiento de la información.

1.1 Objetivo general

Desarrollar un subsistema de almacenamiento de información mediante el uso de Entity Framework, para un sistema de clasificación de fichas Lego basado en imágenes.

1.2 Objetivos específicos

Los objetivos específicos de este Trabajo de Integración Curricular son:

- Analizar las herramientas necesarias para el desarrollo del subsistema requerido.
- Determinar los componentes que conforman el subsistema de almacenamiento, de acuerdo a los requerimientos establecidos.
- Implementar el subsistema de almacenamiento conforme a los componentes identificados para el mismo.

1.3 Alcance

En el presente Trabajo de Integración Curricular se plantea realizar el diseño e implementación de un subsistema de almacenamiento para fichas Lego, con lo cual se pueda almacenar la información aprendida desde una imagen que contiene una ficha Lego, para posteriormente poder mostrar esta información.

Inicialmente se realizará un estudio de la herramienta Entity Framework, así como la generación de los requerimientos de la información que gestionará el subsistema, esto incluirá el diseño del diagrama entidad-relación.

Posteriormente se realizará la codificación, en la cual se procederá a la creación de las diferentes clases que gestionarán la información de las fichas Lego recuperadas desde una base de datos, incluyendo los atributos que se determinaron en el diseño. Además, se generará un *stub* para posteriormente demostrar la funcionalidad de la base de datos.

Una vez implementada la base de datos, así como el *stub* del subsistema de consulta, se realizarán las diferentes pruebas que consistirán en realizar búsquedas empleando diferentes parámetros como color, forma, etc.

1.4 Marco teórico

En esta sección, se presenta la teoría y conceptos relacionados a la temática del mismo.

1.4.1 Entity Framework

Entity Framework[1] es un *Object-Relational Mapper* (ORM), de código abierto dirigido a aplicaciones desarrolladas con .NET Framework de Microsoft. Entity Framework usa un marco orientado a objetos, lo cual permite simplificar el código de acceso a datos, haciendo uso de clases [5].

Entity Framework ayuda al manejo de forma abstracta de los datos manejándolos como objetos, y al ser un componente de .NET Framework las aplicaciones desarrolladas con Entity Framework son compatibles son equipos que emplean .NET Framework desde la versión 3.5SP1 [5].

En la Figura 1.1 se presenta la arquitectura usualmente empleada para el desarrollo de las aplicaciones con .NET Framework. En esta arquitectura, Entity Framework se ubica en la capa[2] de datos y sirve de punto de conexión entre la base de datos y la aplicación misma.

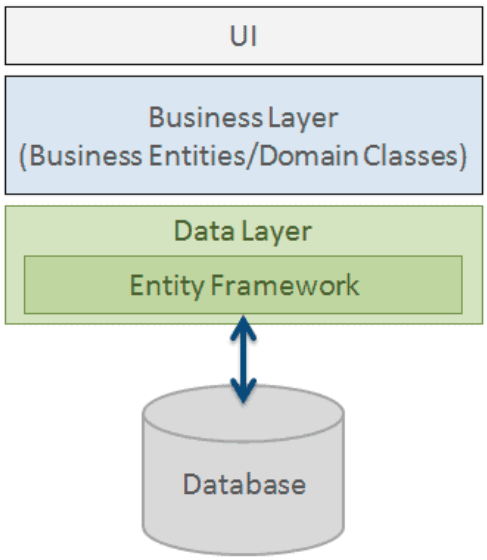


Figura 1.1. Arquitectura de una aplicación con Entity Framework

Entre algunas características de Entity Fremawork se tiene:

- Permite realizar consultas usando LINQ[3], para ejecutar diferentes operaciones con la base de datos.

[1] *Framework*: estructura, plantilla o esquema previo para el desarrollo de proyectos.
[2] Capa: En el área de programación una capa hace referencia a una estructura separada en la lógica de una aplicación y con una función en específico.
[3] LINQ: *Language Integrated-Query* conjunto herramientas para realizar consultas a bases de datos.

- Permite realizar operaciones como *INSERT, UPDATE* y *DELETE* en la base de datos, de acuerdo a los cambios que se realizan desde el código.
- Convierte los resultados de las consultas en objetos con los cuales se podrá trabajar en la aplicación.

Para trabajar con Entity Framework, se puede emplear 2 enfoques:

- *Database-First,*
- *Code-Fisrt*[4]

1.4.1.1. *Database First*

Es uno de los enfoques con los que se puede trabajar en Entity Framework. Como su nombre indica, este enfoque usa una base de datos ya existente, a través de la cual se obtendrán los objetos que se usarán para su gestión.

Las clases que se crean al usar *Database-First* permitirán manipular los datos como objetos; la creación de estas clases se realiza de forma automática mediante Entity Framework.

Este enfoque se usa cuando se desea trabajar en el diseño entidad-relación de la base de datos para posteriormente implementar o asociar la base de datos con la aplicación a través de Entity Framework [7],[8].

En *Database-First* el modelo se almacena en un archivo de extensión EDMX[5], y con base en este archivo se crean las clases con las que la aplicación interactúa con la base de datos.

Entre las ventajas que se tiene están:

- Sencillo de implementar.
- Es útil cuando se trabaja con una base de datos grande y con muchas tablas.
- La pérdida de datos por modificaciones es menos común al usar el enfoque *Database-First*.

[4] *Code-First:* enfoque usado por Entity Framework en cual se codifica las clases que interactuaran con la base de datos, en lugar de hacer uso de una base ya existente.
[5] EDMX: es un tipo de extensión asociado con archivos empleados por ADO.NET

1.4.1.2 Clase de Contexto

Al usar Entity Framework se usa la clase de contexto la cual gestiona la información de la base de datos y permite realizar operaciones CRUD[6] para crear, leer, eliminar o actualizar la información. Dentro de esta clase de encuentran propiedades como `set` y `get`, que permiten gestionar los datos que se almacenaran en las tablas de la base de datos [1].

En la Figura 1.2 se observa un ejemplo de una clase que hereda de la clase `DbContext`, en donde se observa que entre la línea 6 a 10 se define el constructor de la clase que a su vez llama al constructor de la clase base para establecer la cadena de conexión con la base de datos.

Mientras que en las líneas 14 y 15 se definen las propiedades `set` y `get`, las cuales se encargan los encargados de manipular los datos que se envían o se consultan de la base de datos.

```
6       public class ContextoBaseDatos : DbContext
7       {
8           public ContextoBaseDatos()
9               : base("DefaultConnection")
10          {
11
12          }
13
14          public DbSet<Tag> Tags { get; set; }
15          public DbSet<Post> Posts { get; set; }
16      }
```

Figura 1.2. Clase de Contexto (*DbContext*)

1.4.2 GIT

Es un sistema de control de versiones de código abierto que permite llevar un registro de los cambios que se van realizando en un proyecto en el transcurso del tiempo, es decir ayuda a mejor la eficiencia en el desarrollo de un proyecto.

Entre algunas características de Git se puede mencionar:

- Permite realizar una gestión distribuida

[6] CRUD: hace referencia a un acrónimo en inglés, de acuerdo a las 4 operaciones fundamentales para aplicaciones: *Create, Read, Update, Delete*.

- Permite realizar gestión más eficiente de proyectos grandes y que requieren la participación entre varias personas
- Ofrece mayor rapidez al implementar cambios.

1.4.3 GITHUB

Es una plataforma que permite administrar aplicaciones que utilizan el sistema Git, así como subir código de diferentes aplicaciones y desarrollados por diferentes programadores, además de poder descargar y realizar colaboraciones en proyectos, es decir también realiza el manejo de control de versiones de manera distribuida.

Además, permite implementar cambios o regresar a versiones anteriores de una aplicación en la que se está trabajando.

Entre algunas ventajas de utilizar GitHub se tienen:

- El alojamiento de proyectos es de forma gratuita.
- Los repositorios son de carácter público, aunque también se puede optar por repositorios privados.
- Contiene una herramienta llamada GitHub Copilot [7]que ayuda a trabajar de manera más eficiente y permite realizar corrección de errores.
- Facilita el compartir los proyectos con otras personas y de igual manera el realizar colaboraciones con diferentes desarrolladores.

GitHub administra y autoriza los cambios que realizan los colaboradores en caso de que exista más de un desarrollador, y en caso de cometer errores ser capaz de volver a una versión estable de proyecto es decir a una versión anterior [4].

1.4.4 SQL Server

Es un sistema gestor de base de datos relacional proporcionado por Microsoft, da soporte para aplicaciones y puede funcionar ya sea en el mismo computador o en uno diferente que esté conectado a través de una red.

[7] GitHub Copilot: es un asistente que ayuda a generar código proporcionado por la herramienta GitHub, para realizar proyectos y reducir errores.

Algunas de las características de SQL Server son:

- Alta disponibilidad.
- Rápida conmutación.
- Robusto.
- Flexible

Las características antes descritas son proporcionadas gracias a las funciones de memoria que están integradas en el motor de base de datos.

1.4.5 ASP.NET MVC

ASP.NET MVC emplea el patrón Modelo-Vista-Controlador (MVC), mediante el cual se separa la lógica de la aplicación, de la forma gestionar y de visualizar los datos [2].

Permite desarrollar aplicaciones web, haciendo uso del lenguaje de programación C#[8], de igual manera permite incorporar diversas tecnologías web como Javascript[9], CSS[10] o HTML[11].

En MVC, se separan los datos que se usan en una aplicación, la interfaz de usuario y la lógica en tres componentes diferentes.

- El modelo es aquel que contiene los datos que maneja el sistema.
- La vista, maneja la información que se envía al usuario, así como los mecanismos que interactúan con el mismo.
- El controlador es el que actúa como intermediario entre el modelo y la vista, y gestiona el flujo de la información entre estas dos partes.

En la Figura 1.3 se presenta un ejemplo del patrón MVC en el cual se observan los diferentes pasos que se realiza, a partir de que un usuario realiza una petición (1), el controlador la procesa y solicita al modelo información (2), el modelo interactúa con la base de datos (3), la base de datos responde con la información solicitada y la retorna al

[8] C#: Lenguaje de programación perteneciente a Microsoft orientado a objetos, está dirigido para aplicaciones de .NET
[9] Javascript: Leguaje de programación que permite implementar cierta funcionalidad más compleja que puede ejecutarse en el lado del cliente o en el lado del servidor.
[10] CSS (*Cascade Style Sheet*): Lenguaje usado para dar diferentes estilos al diseño de páginas web.
[11] HTML (Hyper Text Markup Language): Lenguaje de marcado para el desarrollo de páginas web.

controlador (4), el controlador procesa la información y la envía a la vista (5), una vez renderizada la vista se genera una interfaz de usuario, la misma que se entrega al usuario (6).

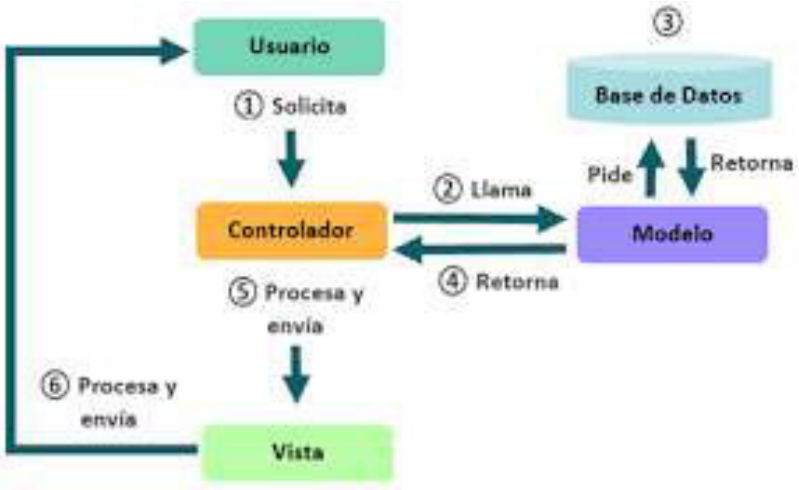


Figura 1.3. Funcionamiento de MVC

Algunas ventajas que se tiene al trabajar con aplicaciones basadas en MVC son:

- Facilita el desarrollo de pruebas denominadas TDD[12]
- Facilita el trabajo de desarrolladores y diseñadores web, ya que brinda un mayor control sobre el comportamiento de la aplicación.
- Al dividir la lógica de la aplicación disminuye la complejidad de la aplicación.

1.4.5 Metodología Kanban

Kanban [3] es una palabra japonesa formada por KAN que significa visual y BAN que quiere decir tarjeta, lo que hace referencia a tarjetas visuales. La metodología Kanban forma parte de las metodologías agiles y su objetivo es la de gestionar las tareas a realizar.

Para el desarrollo de software, la metodología Kanban usa un tablero con múltiples columnas, en las cuales se representan las diferentes etapas del desarrollo de un proyecto.

Para el tablero se hace uso de tarjetas organizadoras de izquierda a derecha.

[12] TDD: (*Test-Driven Develpment*): Práctica para realizar pruebas de funcionalidad que consiste en realizar primero la prueba y después la implementación del código funcional.

El tablero contiene columnas que definen el estado de las tareas. El tablero corresponde al *backlog* del proyecto.

En la Figura 1.2 se puede observar el *backlog* en el cual se han definido columnas que indican el estado de cada una de las tareas, y se dividen en: pendientes o por hacer, en desarrollo o en proceso y hechas o finalizadas. Las actividades van cambiando de columna conforme se vayan cumpliendo con las mismas.

Figura 1.2. Tablero Kanban

2 METODOLOGÍA

Para el presente Trabajo de Integración Curricular se empleó la metodología Kanban, con la cual se definieron las tareas que se debían ser desarrolladas; estas tareas fueron colocadas en un tablero con un conjunto de columnas donde se definen las etapas para el desarrollo (por realizar, en proceso y finalizado).

Como parte del diseño, se realizaron entrevistas con el Director a cargo de este Trabajo de Integración Curricular, para poder establecer una visión del trabajo a implementar y definir las historias de usuario y los requerimientos de la aplicación a ser desarrollada.

Una vez analizada la información obtenida a través de la entrevista se identificaron las tareas necesarias que fueron incluidas en el *product backlog*.

Posteriormente se procedió con el diseño del prototipo, empezando por establecer la arquitectura a utilizar, generando el respectivo diagrama de entidad – relación de la base de datos, así como el diagrama de clases correspondiente.

Como parte de las herramientas requeridas están: el administrador de base de datos Microsoft SQL Management Studio 18, el IDE (*Integrated Develoment Enviroment*) Visual Studio 2019, Entity Framework, ASP. NET MVC, GitHub y el lenguaje de programación C#.

Para la base de datos se usó el modelo *Database – First* por lo que se procedió a codificar un *scritp* para la base de datos, el mismo que sirvió como molde para la creación de las clases que corresponde al modelo. A continuación, se desarrollaron los respectivos controladores y sus métodos para gestionar las peticiones enviadas desde el cliente, de igual forma, se implementaron las correspondientes vistas para la visualización de la información correspondiente a las imágenes de fichas Lego.

Finalmente, se llevaron a cabo pruebas para comprobar el correcto funcionamiento de cada una de las funcionalidades implementadas, para lo cual se realizaron operaciones de creación, edición y eliminación de imágenes con fichas Lego, además, de pruebas de búsqueda por diferentes parámetros como color, forma, factor entre otros.

2.1 Historias de Usuario

Las historias de usuario fueron obtenidas mediante entrevista al Director a cargo del Trabajo de Integración Curricular, de las cuales se obtuvieron los requerimientos funcionales y no funcionales que debe cumplir el prototipo.

Requerimientos funcionales

- Permitir el almacenamiento de imágenes Lego en formato JPG.
- Almacenar características de las imágenes en la base de datos.
- Permitir la consulta de las imágenes almacenadas, así como sus características.

Requerimientos no funcionales

- Para gestionar los datos se hará uso de Entity Framework versión 6.4.4
- Para almacenar los datos se usará SQL Server 2019
- Para la presentación de los datos se usará ASP.NET MVC 5.
- Para la gestión del código se usará GitHub

En la Tabla 2.1 se presentan las historias de usuario generadas, las cuales están compuestas del siguiente formato, `HU_numero_historia_usuario`.

Tabla 2.1. Historia de Usuarios

ID	Nombre	Descripción
HU_01	Almacenar Imágenes	La aplicación permitirá guardar imágenes.
HU_02	Mostrar imágenes	La aplicación permitirá mostrar las imágenes guardadas.

2.1.1 Product backlog

Las actividades que constan en el *product backlog* fueron desarrolladas con base en el resultado de la entrevista realizada al Director del Trabajo de Integración Curricular. En la Tabla 1.1 se presenta el *producto backlog*.

Tabla 2.2. *Product Backlog*

No.	Requerimientos
1	Generar la funcionalidad que permita almacenar imágenes de fichas Lego y sus características
2	Generar la funcionalidad que permita mostrar la información almacenada en la base de datos.

2.2 Diseño

El subsistema se implementó con base en una arquitectura cliente – servidor y siguiendo con el patrón MVC; el servidor web es el encargado de recibir las peticiones y entregar los recursos mediante HTTP; el servidor de aplicaciones es ASP.NET y se encarga de procesar los pedidos que serán solicitados por el servidor web, a partir de solicitudes remitidas por el navegador web debido a las interacciones que el usuario realiza en una página web.ASP.NET MVC se encarga de enrutar el pedido al controlador correspondiente, el cual ejecuta el método adecuado para solicitar la información requerida a la base de datos o para que esta actualice los datos correspondientes. En la base de datos se emplea SQL Server; para almacenar la información. Finalmente, el modelo se encarga de recuperar o actualizar la información en la base datos relacional. El modelo emplea Entity Framework para la gestión de los datos.

2.2.1 Diseño de la base de datos

Para determinar las diferentes características que deben ser almacenadas en la base de datos, se optó por dividir la información de acuerdo a las características de la imagen, de la ficha Lego, y por el tipo de ficha Lego. En la Figura 2.1 se presenta el diagrama relacional de la base de datos realizada.

La tabla `tblLego` contiene la información de la ficha Lego que será almacenada, contará con atributos como color, identificador y las respectivas llaves foráneas de las tablas con las que se encuentra relacionada; esta tabla se relaciona con la tabla `tblTipoLego` mediante una relación de tipo uno a muchos ya que un elemento de la tabla `tblTipoLego`

puede pertenecer a varios elementos de la tabla `tblLego`, de igual forma la tabla `tblLego` tiene una relación de tipo uno a muchos con la tabla `tblImagen` lo que implica que al igual que con la tabla `tblTipoLego` un elemento de la tabla `tblImagen` puede pertenecer a varios elementos de la tabla `tblLego`. En la tabla `tblTipoLego` se almacena la información relacionada al tipo de la ficha Lego, mientras que en la tabla `tblImagen` se almacena la imagen y características asociadas a dicha ficha Lego.

La tabla `tblTipoLego` contiene atributos como identificador, descripción y factor, los cuales dan las características complementarias a la ficha Lego, y la misma tiene una relación de tipo uno a muchos con la tabla `tblLego` ya que un elemento de la tabla `tblTipoLego` puede pertenecer a muchos elementos de la tabla `tblLego`.

En la tabla `tblImagen` se almacena la información de la imagen de la ficha Lego y contara con atributos como alto, ancho, identificador y la imagen como tal. La tabla `tblImagen` se relaciona con la tabla `tblLego` con una relación del tipo uno a mucho ya que una misma imagen puede pertenecer a varias fichas Lego.

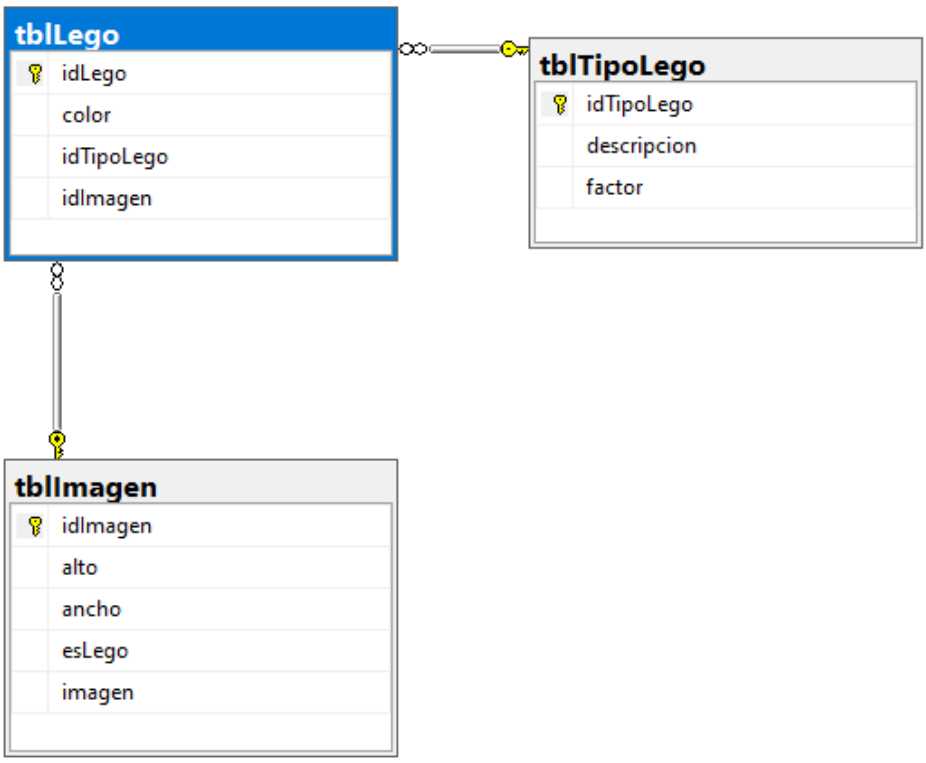


Figura 2.1. Diagrama Relacional de la base de datos.

2.2.2 Diseño de Clases

Con las tablas establecidas en la base de datos, se generó el modelo mediante Entity Framework junto con los atributos requeridos para la gestión de la información.

En la Figura 2.2 se presenta el diagrama de clases del modelo generado haciendo uso de Entity Framework.

En el diagrama se observan las propiedades generadas, las cuales corresponden con cada atributo de la tabla respectiva de la base de datos, así como propiedades de navegación que permiten asociar los datos de una tabla con otra.

Este diagrama es equivalente al diagrama relacional que se muestra en la Figura 2.1.

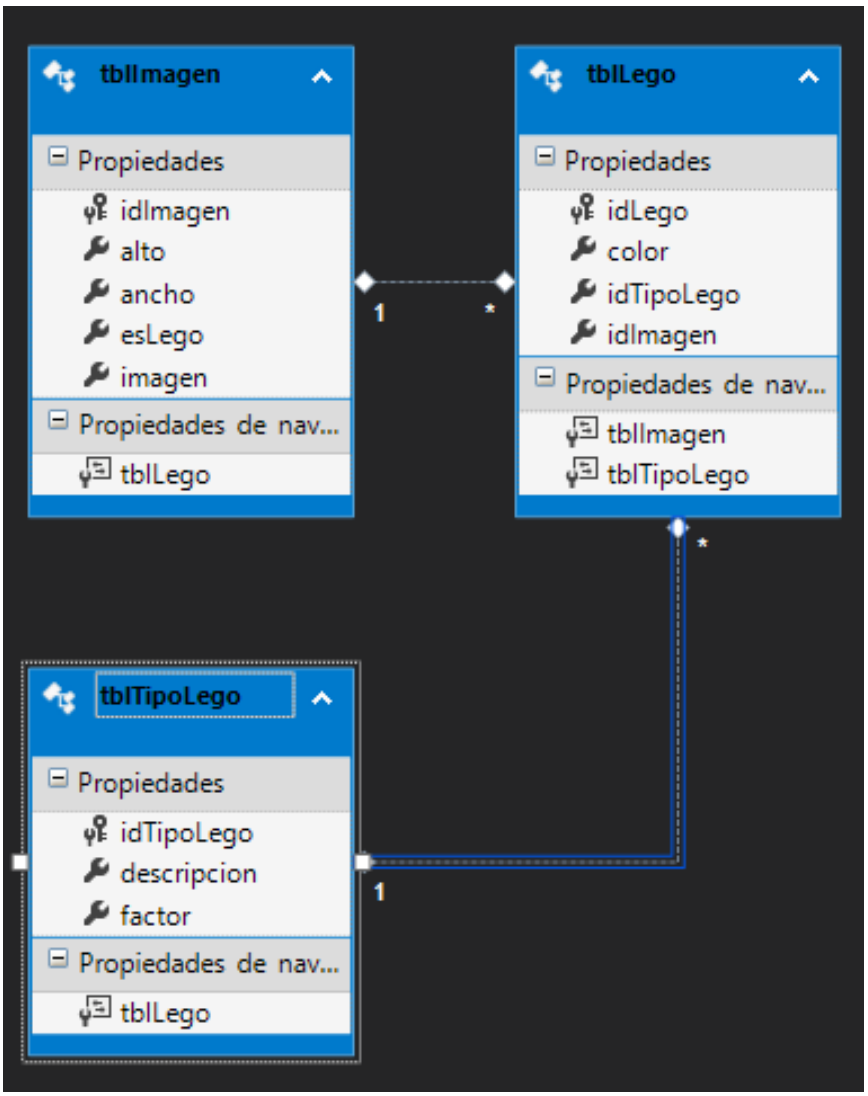


Figura 2.2. Diagrama de clases de modelos creado por Entity Framework

En la Figura 2.3 se muestra un código ejemplo de la una de las clases generadas mediante Entity Framework, en la cual se observa que la clase contiene un constructor (líneas 18 a 21), propiedades (líneas 23 a 27) que permiten el acceso a los atributos del modelo encapsulando los mismos, y propiedades de navegación (línea 30), para gestionar la relación con las diferentes clases generadas por Entity Framework.

```
15  public partial class tblImagen
16  {
17      [System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2214:DoNotCallOverridableMethodsInConstructors")]
18      public tblImagen()
19      {
20          this.tblLego = new HashSet<tblLego>();
21      }
22
23      public int idImagen { get; set; }
24      public string alto { get; set; }
25      public string ancho { get; set; }
26      public Nullable<int> esLego { get; set; }
27      public byte[] imagen { get; set; }
28
29      [System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2227:CollectionPropertiesShouldBeReadOnly")]
30      public virtual ICollection<tblLego> tblLego { get; set; }
31  }
32  }
33
```

Figura 2.3. Clase creada mediante Entity Framework

2.3 Implementación

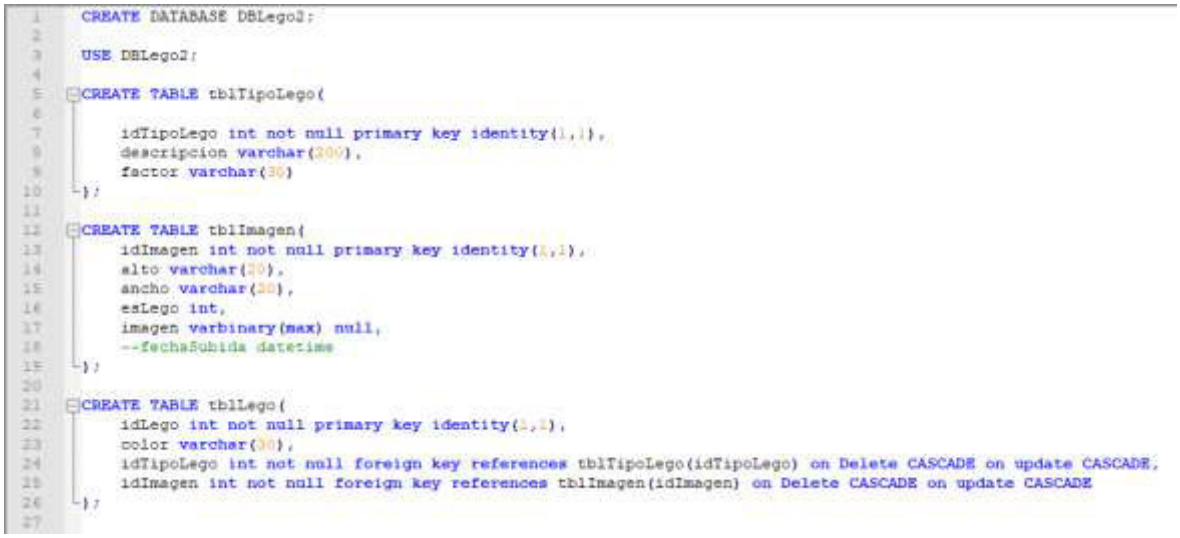
Para el presente trabajo de integración curricular se procedió a desarrollar un script para crear la base de datos con el administrador de bases de datos Microsoft SQL Management Studio 18.

Además, se instalaron los paquetes necesarios para la codificación de las vistas, modelos y controladores que manejarán la información.

2.3.1 Codificación de la base de datos.

Para la correspondiente codificación de la base de datos se procedió a crear un *script* de acuerdo al diagrama entidad – relación realizado.

En la Figura 2.4 se presenta el *script* escrito en SQL mediante el cual se crean las diferentes tablas implementadas: `tblLego`, `tblTipoLego`, `tblImagen`, mediante la sentencia `CREATE TABLE` (líneas 5,12 y 21) la cual contiene los parámetros requeridos para cada tabla.



```
CREATE DATABASE DBLego2;

USE DBLego2;

CREATE TABLE tblTipoLego(

    idTipoLego int not null primary key identity(1,1),
    descripcion varchar(300),
    factor varchar(30)
);

CREATE TABLE tblImagen(
    idImagen int not null primary key identity(1,1),
    alto varchar(30),
    ancho varchar(30),
    esLego int,
    imagen varbinary(max) null,
    --fechaSubida datetime
);

CREATE TABLE tblLego(
    idLego int not null primary key identity(1,1),
    color varchar(30),
    idTipoLego int not null foreign key references tblTipoLego(idTipoLego) on Delete CASCADE on update CASCADE,
    idImagen int not null foreign key references tblImagen(idImagen) on Delete CASCADE on update CASCADE
);
```

Figura 2.4. *Script* para la creación de la base de datos

En la Figura 2.5 se presenta el código para la inserción de datos, en particular se usa la sentencia `INSERT INTO` en las líneas 28, 31 y 34 para indicar la tabla y las columnas en la cuales se insertará la información correspondiente a las características. En este caso en particular se ingresan datos en `tblTipoLego` y mediante la sentencia `values` se indica la información que será almacenada en la tabla.

```
28  insert into tblTipoLego(descripcion,factor)
29  values ('Cuadrado','1x1');
30
31  insert into tblTipoLego(descripcion,factor)
32  values ('Rectangular','2x1');
33
34  insert into tblTipoLego(descripcion,factor)
35  values ('En L','3x1');
```

Figura 2.5: *Script* para la inserción de datos en la tabla `tblTipoLego`

2.3.2 Vistas

Para la generación de cada una de las vistas se trabajó con plantillas generadas con Razor. Se crearon 3 vistas denominadas `Imagen`, `Lego` y `TipoLego`. La vista `Imagen` muestra un listado de las imágenes que han sido ingresadas con sus respectivos atributos y la vista `Lego` muestra un listado de imágenes que han sido clasificadas de igual forma con sus respectivos atributos. En la Figura 2.6 se muestra la vista `Imagen` la cual contiene un

listado de las imágenes que han sido ingresadas, junto con las respectivas opciones para crear o ingresar una nueva imagen, editar, ver los detalles o borrar una de las imágenes ingresadas.

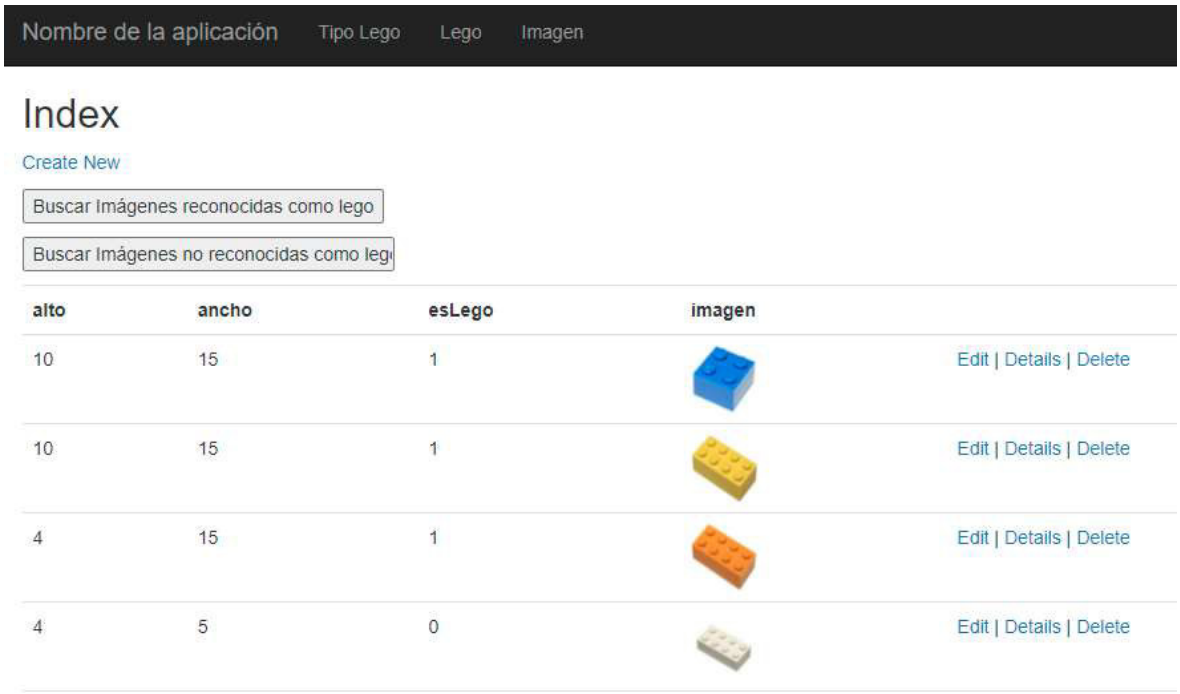


Figura 2.6. Vista `Imagen` para visualizar imágenes ingresadas

Como ya se mencionó para la generación de las vistas se hizo uso de plantillas Razor, las cuales permiten usar una combinación de etiquetas escritas en el lenguaje HTML con código escrito en el lenguaje C#.

En la Figura 2.7 se muestra una sección de código de la vista generada para gestionar la información de la tabla `tblIamgen`, donde en la línea 7 se usa la etiqueta `<h2>` con la cual se define un título, en la línea 10 se utiliza el método `ActionLink`, el cual es uno de los HTML *Helpers* que permite generar el enlace a un método de acción especifico de un controlador, en este caso el método de acción `create` del controlador `ImagenController`.

Entre las líneas 14 a 18 se define un formulario el mismo que está asociado al método de acción `BuscarImagenes` del controlador `ImagenController`, y se indica que, cuando el usuario presione en el botón de tipo `submit` se enviará una solicitud POST al controlador, y en el cuerpo se incluirá el parámetro denominado `esLego`, el mismo que permite establecer si la búsqueda corresponde a imágenes que han sido reconocidas como

fichas Lego, o no. Algo simular se realiza entre las líneas 21 a la 25, pero para buscar imágenes que no contienen una ficha Lego.

```
6
7   <h2>Index</h2>
8
9   <p>
10      @Html.ActionLink("Create New", "Create")
11  </p>
12
13  <p>
14      <form action="@Url.Action("BuscarImagenes","Imagen")">
15
16          <input type="hidden" name="esLego" value="1" />
17          <input type="submit" value="Buscar Imágenes reconocidas como lego"/>
18      </form>
19  </p>
20  <p>
21      <form action="@Url.Action("BuscarImagenes","Imagen")">
22
23          <input type="hidden" name="esLego" value="0" />
24          <input type="submit" value="Buscar Imágenes no reconocidas como lego" />
25      </form>
26  </p>
27
```

Figura 2.7. Código de la Vista `Imagen`

En la Figura 2.6 se presenta la vista con un listado de las imágenes ingresadas, así como sus atributos usando un formato de tabla. Para generar esta parte de la vista se emplea el código de la Figura 2.8.

En la línea 29 se utiliza la etiqueta `<table class= "table">` con la cual se establece que se generará una tabla, la misma que será definida entre las líneas 30 a la 44, en estas líneas, se configura los nombres de las diferentes columnas, que en este caso son 4, sus propiedades alto, ancho, un indicador que establece si la imagen tiene o no una ficha Lego y finalmente la imagen en sí. Para definir el encabezado de cada una de las columnas se usa la etiqueta `<th>`, mientras que la etiqueta `<tr>` se usa para indicar que estas celdas serán de contenido y corresponderán a cada columna, de acuerdo al orden definido.

De las líneas 46 a 67 se emplea una sentencia de código `foreach`, mediante la cual se iterará entre los diferentes elementos disponibles en la base de datos para presentar los detalles de cada una de las diferentes imágenes que han sido cargadas en la base de datos, así como sus atributos. Adicionalmente, se incluye código para que se generen enlaces que permitan realizar las acciones correspondientes a Edición (`Edit`), Detalles (`Details`) o Eliminación (`Delete`).

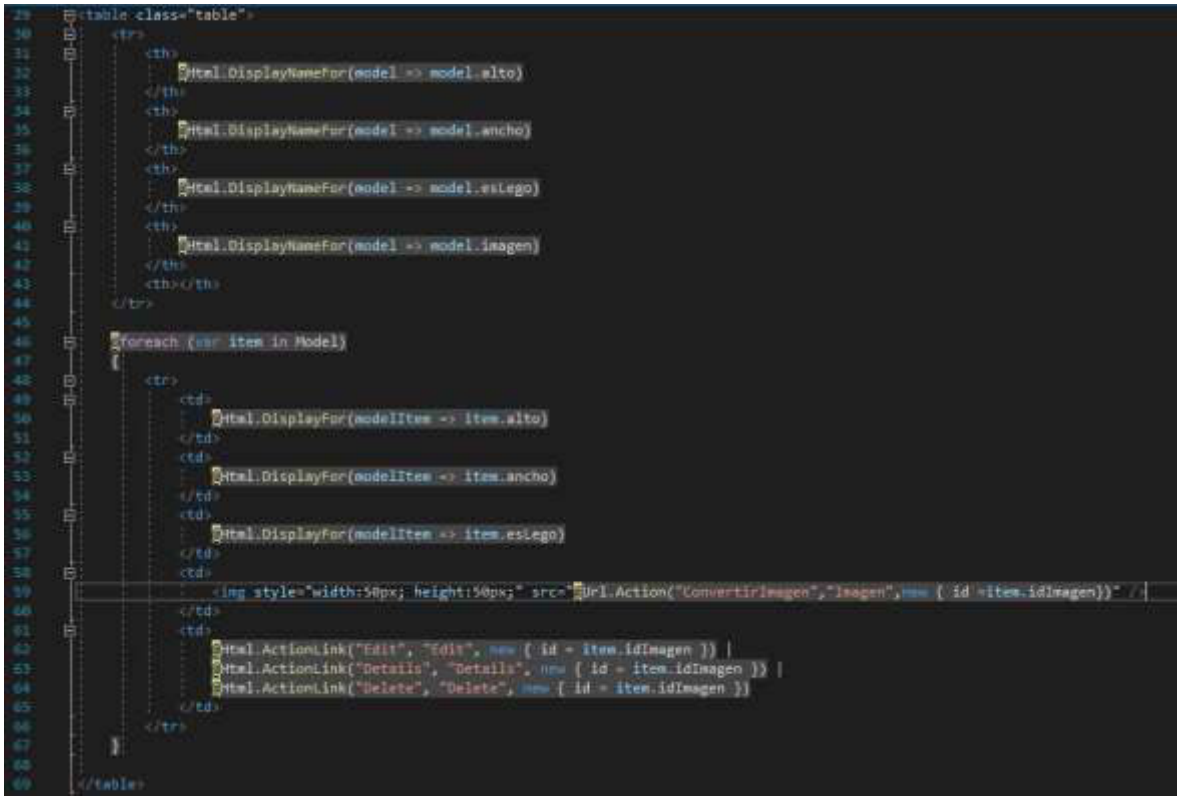


Figura 2.8. Código ejemplo para la creación de la tabla para visualizar las imágenes de `tblImagen`

Las vistas para la visualización de los atributos de la ficha Lego y para los tipos de ficha Lego tienen un funcionamiento similar a la indicada para la vista de `tblImagen`, con la diferencia de que se presentan los atributos propios de cada tabla. En la Figura 2.9 se presenta la información de la tabla `tblTipoLego` mientras que en la Figura 2.10 se presenta la información de la tabla `tblLego`

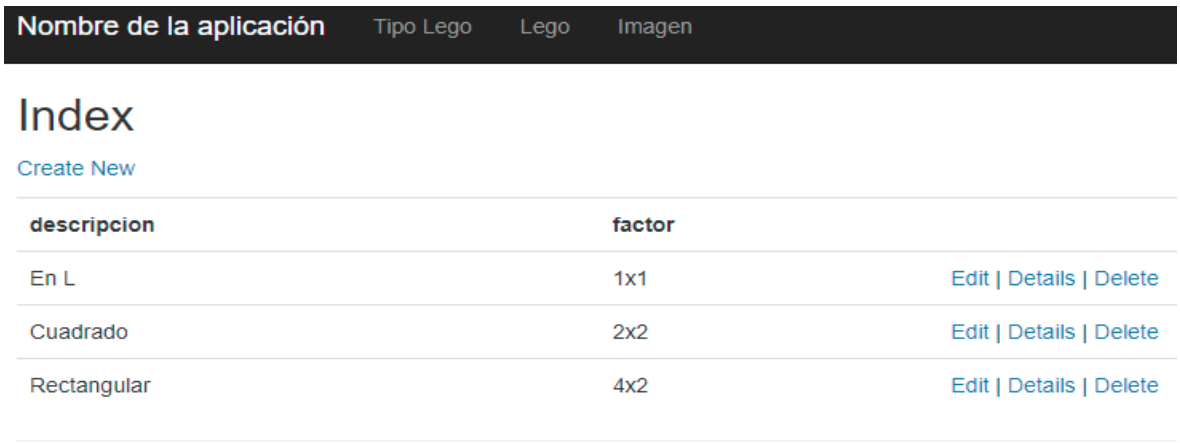


Figura 2.9. Vista para la presentación de información de la tabla `tblTipoLego`

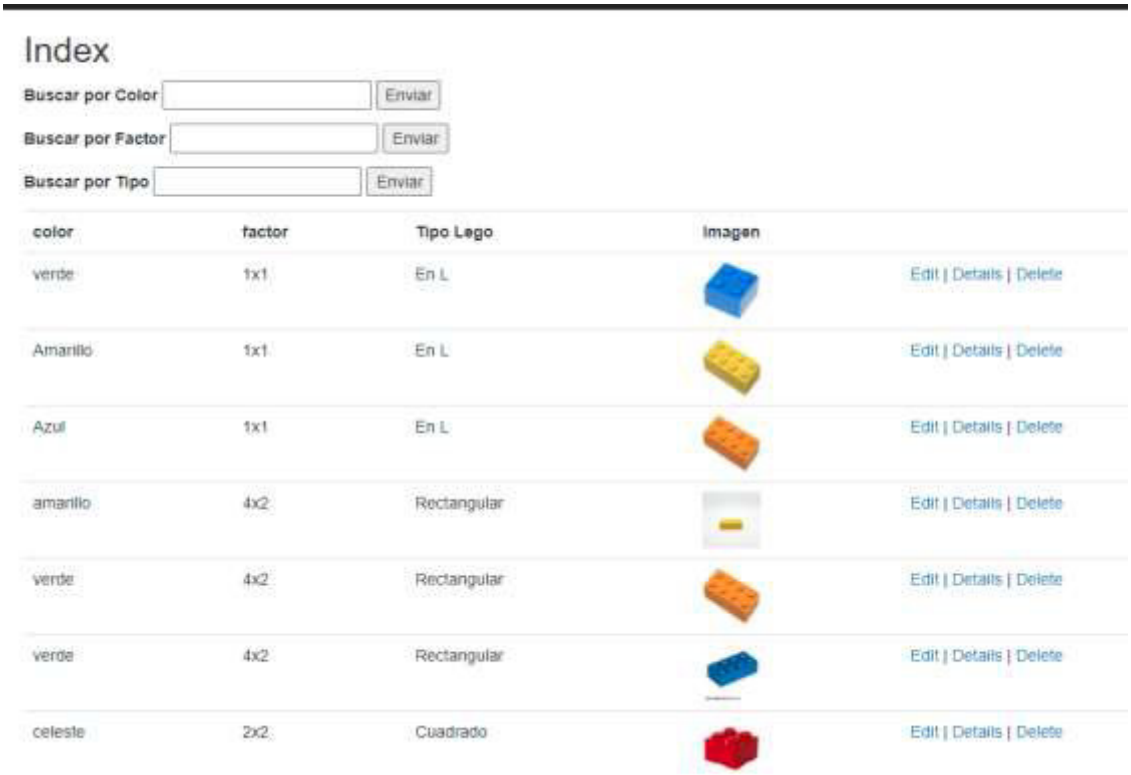


Figura 2.10. Vista para presentación de información de la tabla `tblLego`

2.3.3 Controladores

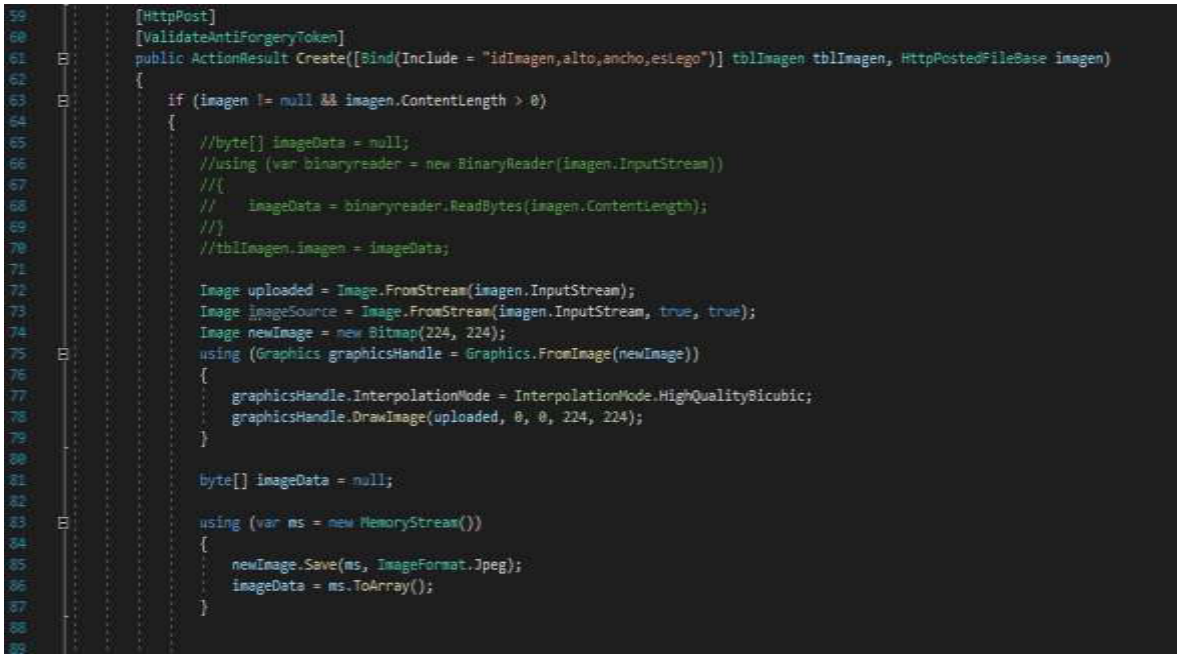
El prototipo consta de 4 controladores `HomeController`, `ImagenController`, `LegoController` y `TipoLegoController`, de los cuales el controlador `HomeController` fue creado por Visual Studio por defecto al generar la solución para gestionar el código de este Trabajo de Integración Curricular, mientras que los demás fueron creados para el manejo de las acciones relacionadas con la información de las respectivas imágenes.

El controlador `ImagenControlle` contiene los métodos correspondientes para el manejo de las imágenes, ya sea para el ingreso, edición o para eliminar la imagen de la base de datos.

Para el ingreso de una nueva imagen que contiene una ficha Lego, cuando el usuario interactúe con la vista, se generara una solicitud desde el explorador web, la misma que provocará que en el servidor se llame a un método específico del controlador `ImagenController` para realizar la acción determinada y asociada a la solicitud.

En la Figura 2.11 se puede observar el código relacionado al ingreso y almacenamiento de una nueva imagen que contiene de una ficha Lego, en el controlador `ImagenController`.

Se establece el atributo `HttpPost` para indicar que el método `Create` corresponda a una solicitud de tipo POST (línea 59), la cual acepta como parámetros los atributos de la imagen como: alto, ancho y el contenido de la imagen.

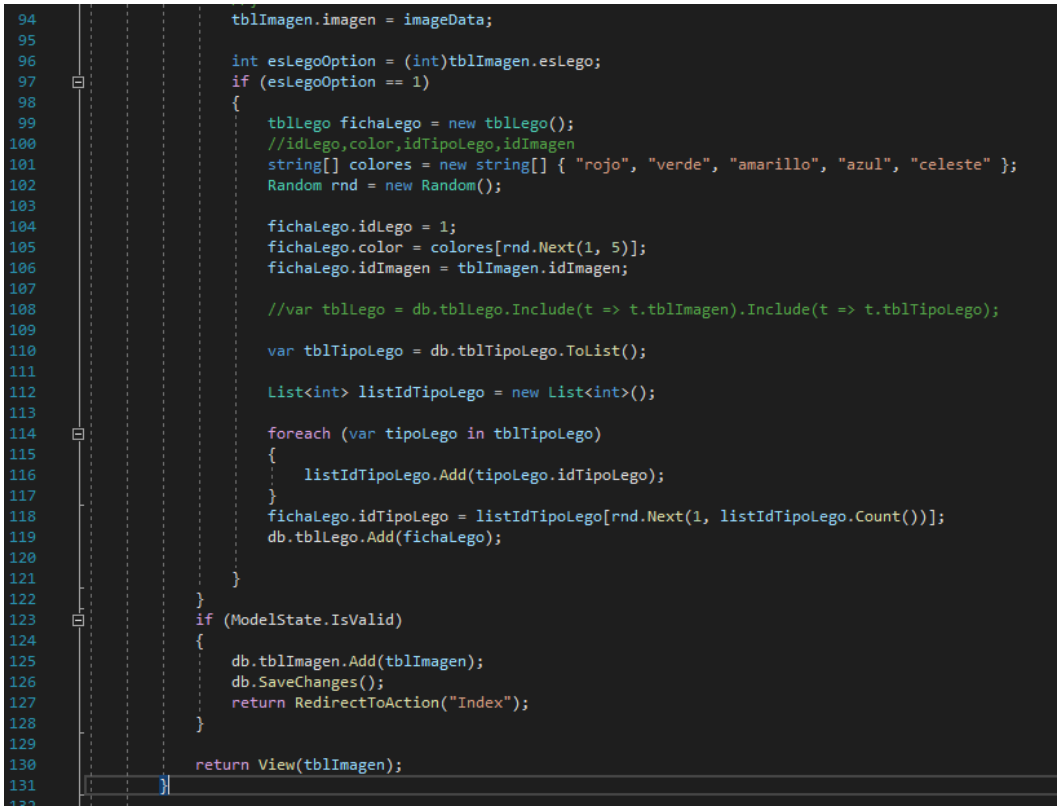


```
59      [HttpPost]
60      [ValidateAntiForgeryToken]
61      public ActionResult Create([Bind(Include = "idImagen,alto,ancho,esLego")] tblImagen tblImagen, HttpPostedFileBase imagen)
62      {
63          if (imagen != null && imagen.ContentLength > 0)
64          {
65              //byte[] imageData = null;
66              //using (var binaryreader = new BinaryReader(imagen.InputStream))
67              //{
68              //    imageData = binaryreader.ReadBytes(imagen.ContentLength);
69              //}
70              //tblImagen.imagen = imageData;
71
72              Image uploaded = Image.FromStream(imagen.InputStream);
73              Image imageSource = Image.FromStream(imagen.InputStream, true, true);
74              Image newImage = new Bitmap(224, 224);
75              using (Graphics graphicsHandle = Graphics.FromImage(newImage))
76              {
77                  graphicsHandle.InterpolationMode = InterpolationMode.HighQualityBicubic;
78                  graphicsHandle.DrawImage(uploaded, 0, 0, 224, 224);
79              }
80
81              byte[] imageData = null;
82
83              using (var ms = new MemoryStream())
84              {
85                  newImage.Save(ms, ImageFormat.Jpeg);
86                  imageData = ms.ToArray();
87              }
88
89
```

Figura 2.11. Código `ImagenController` para el ingreso de una imagen

Se realiza un tratamiento a la imagen para realizar un cambio en las dimensiones de la misma (líneas 72 a 79), para posteriormente almacenar su contenido en un arreglo de bytes (líneas 81 a 87).

En la Figura 2.12 se presenta un fragmento de código que genera de forma aleatoria datos para simular el subsistema de clasificación. Mediante este código, se observa que en las líneas 97 a 106 se han colocado 5 tipos de colores que serán asignados a las imágenes que cumplen con la condición de contener una ficha Lego (`esLego = 1`), de igual forma para la asignación del atributo `TipoLego`, se lee la tabla `tblTipoLego` y se asigna de forma aleatoria el tipo de ficha Lego a la imagen cargada (líneas 110 a 119), mientras que de las líneas 123 a la 130 se realiza el almacenamiento de la imagen cargada en la tabla `tblImagen` de la base de datos, y se devuelve la vista correspondiente.



```
94                  tblImagen.imagen = imageData;
95
96                  int esLegoOption = (int)tblImagen.esLego;
97                  if (esLegoOption == 1)
98                  {
99                      tblLego fichaLego = new tblLego();
100                     //idLego,color,idTipoLego,idImagen
101                     string[] colores = new string[] { "rojo", "verde", "amarillo", "azul", "celeste" };
102                     Random rnd = new Random();
103
104                     fichaLego.idLego = 1;
105                     fichaLego.color = colores[rnd.Next(1, 5)];
106                     fichaLego.idImagen = tblImagen.idImagen;
107
108                     //var tblLego = db.tblLego.Include(t => t.tblImagen).Include(t => t.tblTipoLego);
109
110                     var tblTipoLego = db.tblTipoLego.ToList();
111
112                     List<int> listIdTipoLego = new List<int>();
113
114                     foreach (var tipoLego in tblTipoLego)
115                     {
116                         listIdTipoLego.Add(tipoLego.idTipoLego);
117                     }
118                     fichaLego.idTipoLego = listIdTipoLego[rnd.Next(1, listIdTipoLego.Count())];
119                     db.tblLego.Add(fichaLego);
120
121                 }
122             }
123             if (ModelState.IsValid)
124             {
125                 db.tblImagen.Add(tblImagen);
126                 db.SaveChanges();
127                 return RedirectToAction("Index");
128             }
129
130             return View(tblImagen);
131         }
```

Figura 2.12 Generación de datos aleatorios, y almacenamiento de la imagen

En la Figura 2.13 se presenta el método `Edit` que pertenece al controlador `ImagenController`, este método se encarga de editar los parámetros, de ser necesario de la imagen ingresada y se llama cuando el usuario presiona en el enlace que se crea en la vista `Imagen`. De las líneas 136 a la 139 se devuelve un mensaje de error en caso de que el `id` sea nulo, si no es nulo en la línea 140 se crea un objeto del tipo `tblImagen` como resultado de la llamada al método `Find` que acepta por parámetro el `id` de la imagen seleccionada, en caso de no existir datos asociados a ese `id` en la tabla de se devuelve un mensaje de error, el cual se generara mediante el método `HttpNotFound`, caso contrario se devuelve el objeto mediante la vista respectiva que además permitirá presentar todos los atributos de dicho objeto.

El método `CovertirImagen` (líneas 148 a 152) permite realizar la consulta de una imagen con base en un parámetro que corresponda a la identificación del elemento en la base de datos mediante Entity Framework. La información se obtiene usando expresiones

Lambda[13] mediante el método `Where`, así como el método FisrtOrDefault que permite que solo se devuelva un elemento, sea este el primero que coincida con el criterio de búsqueda o el valor por defecto.

Una vez determinado el elemento de la base de datos que corresponde al `id` indicado se retorna la imagen en el formato establecido para presentarlo en la vista (línea 151). Este método es usado tanto para la vista `Imagen` como para la vista `Lego`.

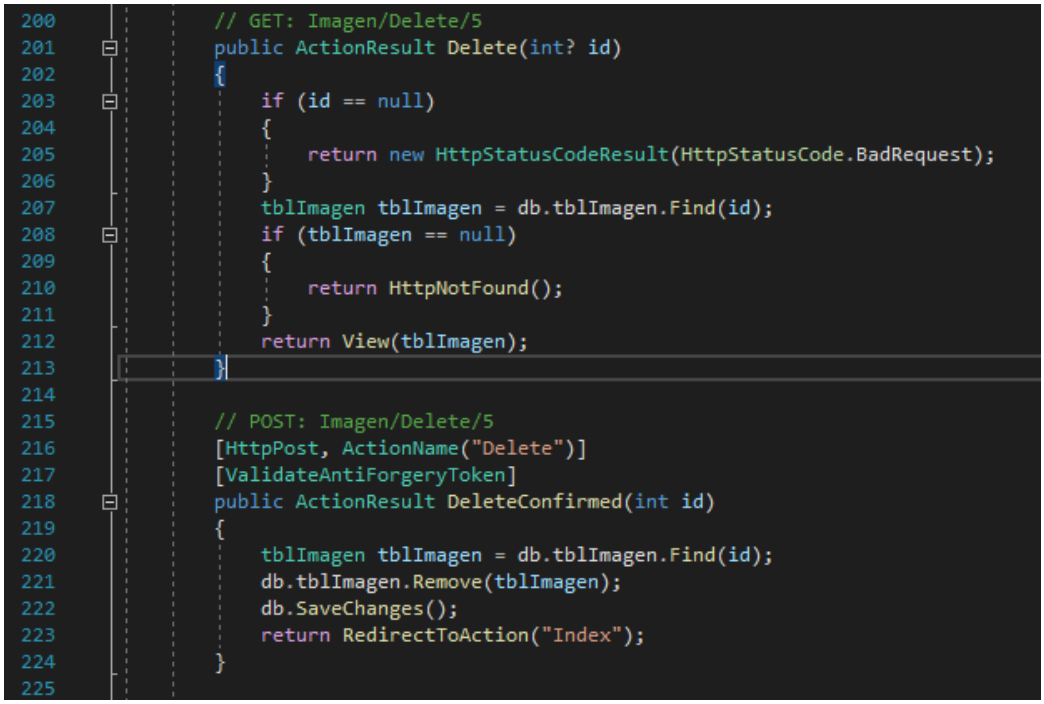
```
133         // GET: Imagen/Edit/5
134         public ActionResult Edit(int? id)
135         {
136             if (id == null)
137             {
138                 return new HttpStatusCodeResult(HttpStatusCode.BadRequest);
139             }
140             tblImagen tblImagen = db.tblImagen.Find(id);
141             if (tblImagen == null)
142             {
143                 return HttpNotFound();
144             }
145             return View(tblImagen);
146         }
147
148         public ActionResult ConvertirImagen(int id)
149         {
150             var imagen = db.tblImagen.Where(x => x.idImagen == id).FirstOrDefault();
151             return File(imagen.imagen, "image/jpeg");
152         }
```

Figura 2.13. Método `Edit y ConvetirImagen`

El método `Delete` será ejecutado cuando el usuario presione en el enlace correspondiente de la vista `Imagen`. Para esto el explorador web, una vez el usuario presione dicho enlace realiza una petición HTTP de tipo GET y la envía al servidor. El servidor entrega la solicitud al controlador y este a su vez ejecuta el método `Delete`. El método `Delete` (líneas 201 a 213) acepta como argumento el identificador del elemento que se desea buscar y devuelve una vista (línea 212) en la que se presentará la imagen que se desea borrar de la base, así como algunas propiedades de la imagen como color y descripción. Por otro lado, si el usuario indica que desea borrar dicha imagen, se generara una nueva solicitud desde el explorador web, pero esta vez usando el método POST de HTTP, el cual provocara que en el servidor se ejecute el método `DeleteConfirmed` presentado entre las líneas 216 a la 224, en el cual se realizan los respectivos cambios en la base de datos, primero determinándola imagen que debe ser borrada (línea 220), eliminando la imagen mediante

[13] Expresión lambda: Función anónimas o no enlazadas a un identificador que ayudan a realizar consultas LINQ

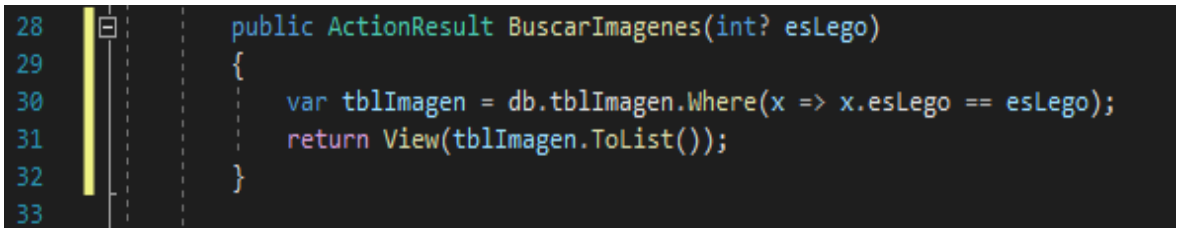
el método `Remove` (línea 221) y guardando los cambios en la base de datos mediante el método `Savechanges` (línea 222), para realizar una redirección a la vista `Index` en la línea 223 .



```
200         // GET: Imagen/Delete/5
201         public ActionResult Delete(int? id)
202         {
203             if (id == null)
204             {
205                 return new HttpStatusCodeResult(HttpStatusCode.BadRequest);
206             }
207             tblImagen tblImagen = db.tblImagen.Find(id);
208             if (tblImagen == null)
209             {
210                 return HttpNotFound();
211             }
212             return View(tblImagen);
213         }
214
215         // POST: Imagen/Delete/5
216         [HttpPost, ActionName("Delete")]
217         [ValidateAntiForgeryToken]
218         public ActionResult DeleteConfirmed(int id)
219         {
220             tblImagen tblImagen = db.tblImagen.Find(id);
221             db.tblImagen.Remove(tblImagen);
222             db.SaveChanges();
223             return RedirectToAction("Index");
224         }
225
```

Figura 2.14. Método *Delete* del `ImagenController`

De igual manera en el controlador `ImagenController` se tiene el método `BuscarImagen` (ver Figura 2.15) encargado de realizar la búsqueda mediante una expresión Lambda (línea 30), y retorna una vista que contiene una lista de objetos del tipo `tblImagen` (línea 31); este método se utiliza en la vista `Imagen` para el listar las imágenes que han sido reconocidas como fichas Lego y aquellas que no han sido reconocidas como fichas Lego.



```
28      public ActionResult BuscarImagenes(int? esLego)
29      {
30          var tblImagen = db.tblImagen.Where(x => x.esLego == esLego);
31          return View(tblImagen.ToList());
32      }
33
```

Figura 2.15. Función de búsqueda en el controlar `ImagenController.cs`

2.3.4 Modelo

En la Figura 2.16 se muestra el modelo creado a partir de Entity Framework con el enfoque *Database – First*, el cual consta de 3 clases `tblImagen`, `tblLego`, `tblTipoLego`, las cuales son las responsables de interactuar con la base de datos para la diferentes consultas y actualizaciones que se requieran implementar. En la carpeta `Models` se almacenan los archivos que ayudan con el manejo de la base de datos.

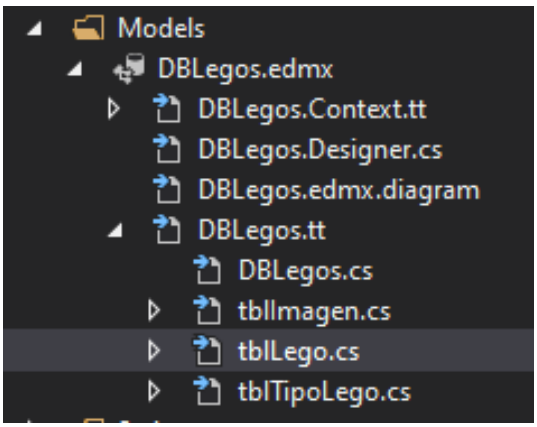


Figura 2.16. Modelo y clases que interactúan con la base de datos.

3 RESULTADOS, CONCLUSIONES Y RECOMENDACIONES

En este capítulo se presentan los resultados de las pruebas realizadas, así como un conjunto de conclusiones y recomendaciones producto de este Trabajo de Integración Curricular.

3.1 Resultados

Se realizaron pruebas de los controladores implementados, para verificar el correcto funcionamiento del prototipo realizado. Para ello se ingresaron diversas imágenes que sirvieron para comprobar el funcionamiento de la base de datos, así como para detectar los errores que se tuvieron durante la implementación.

Para las respectivas pruebas se realizaron operaciones para crear, editar y borrar imágenes, esto con el fin de poder observar que efectivamente las imágenes se guardaron en la base de datos, además de poder probar la interfaz visual para la visualización de la información.

3.1.1. Pruebas de Funcionamiento

En la interfaz de usuario ventana generada a partir de la vista `Imagen` se presenta un listado de imágenes que se encuentran guardadas en la base de datos, así como botones que permiten registrar otra imagen o realizar búsquedas como se observa en la Figura 3.1.

Figura 3.1. Interfaz de Usuario generada a partir de la vista `Imagen`

Al presionar en el botón *Crearte new* que se muestra en la Figura 3.1 la aplicación presenta la interfaz de usuario generada a partir de la vista XX que permite ingresar una nueva

imagen, dicha interfaz se observa en la Figura 3.2. Esta interfaz permite indicar la imagen que debe ser cargada, así como ingresar los campos como `alto` y `ancho`, y un atributo denominado `esLego` que ayuda a la simulación del subsistema de clasificación. Se aclara que esta interfaz es simple y tampoco realiza validaciones puesto que no es parte de este componente, están solo un *stub* que emula la tarea que el subsistema de adquisición de imágenes debe realizar. Una vez definidas las propiedades de la imagen e indicada la imagen a cargar se debe presionar en el botón *Create* para que se envié la información al servidor web

Figura 3.2. Ventana para el ingreso de una nueva imagen

Para verificar que la imagen efectivamente ha sido almacenada en la base de datos, se verifica la lista de imágenes en la interfaz generada a partir de la vista `Imagen`, en la cual el nuevo elemento ingresado se ubicará en la posición final de la tabla presentada, como se observa en la Figura 3.3

Figura 3.3. Imagen ingresada con éxito

Almacenamiento de imágenes clasificadas como fichas Lego

Como se mencionó anteriormente, al momento de ingresar una nueva imagen se debe especificar el parámetro que ayuda a la simulación de la clasificación, con el cual se puede indicar que la imagen tiene o no tiene una ficha Lego.

Se aclara que esta interfaz también es simple, puesto que es la terea del subsistema de presentación de información la generación de la interfaz respectiva, y es responsabilidad del subsistema de clasificación el determinar el tipo de ficha Lego. El parámetro `esLego` si tiene el valor 1, indica que la imagen contiene una ficha Lego y por tanto debe ser guardada en la tabla `tblLego` de la base de datos, al igual que en la tabla `tblImagen`.

En la Figura 3.4 se presenta la interfaz que muestra las imágenes que fueron clasificadas como Lego y por tal motivo se añadieron a la tabla `tblLego`, esta interfaz se genera usando la vista `Lego`, y muestra las propiedades de la ficha Lego como color, factor y su tipo.

Por otro lado, en la Figura 3.5, se observa las imágenes con el atributo `esLego` igual a 1 que las determina como fichas clasificadas correctamente y que corresponde al listado de fichas Lego existente en la base de datos.

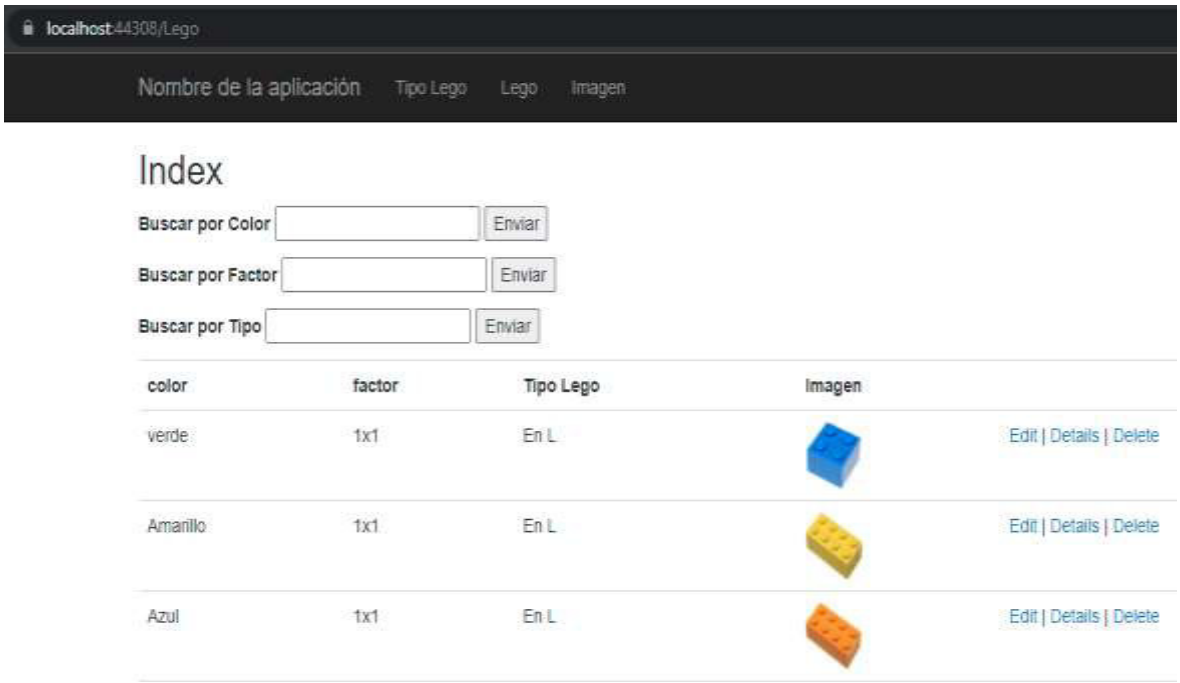


Figura 3.4. Interfaz generada a partir de la vista `Lego` con Imágenes de fichas Lego correctamente clasificadas

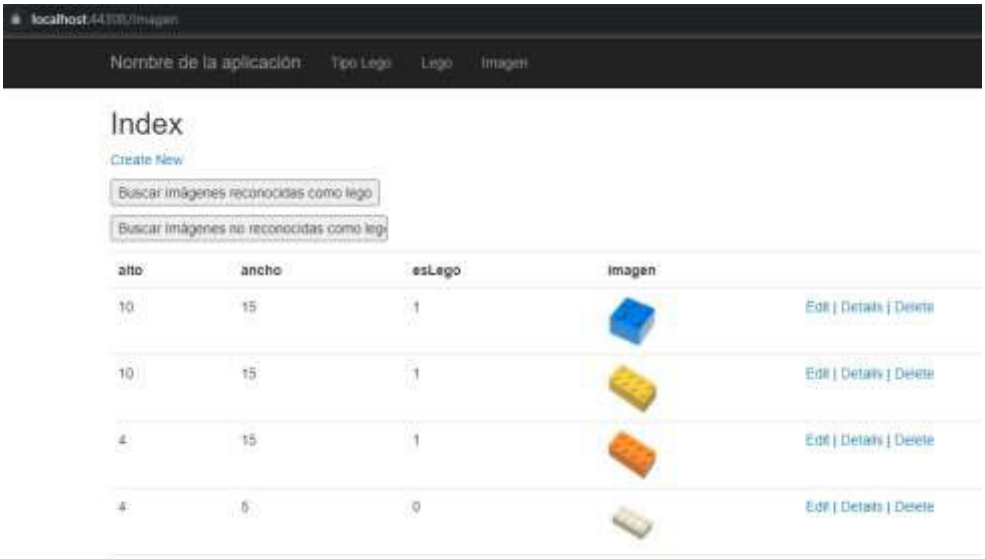


Figura 3.5. Listado de Imágenes de fichas Lego con el atributo `esLego` igual a 1

Búsqueda por parámetros

Una de las funciones implementadas en la aplicación es la búsqueda de las imágenes que contienen una ficha Lego que han sido clasificadas mediante diferentes parámetros, para el ello se tiene en la vista `Imagen` dos opciones, una de las cuales permiten observar el número de imágenes que han sido clasificadas como fichas Lego, la cual se muestra en la Figura 3.6, mientras que en la Figura 3.7 se muestra las imágenes que no han sido clasificadas como fichas Lego.

Figura 3.6. Imágenes que ha sido clasificadas como fichas Lego

Figura 3.7. Imágenes que no han sido clasificadas como fichas Lego

En la interfaz generada por la vista `Lego` que se muestra en la Figura 3.8, se presentan 3 campos para realizar búsquedas, las cuales son: color, factor y tipo de Lego.

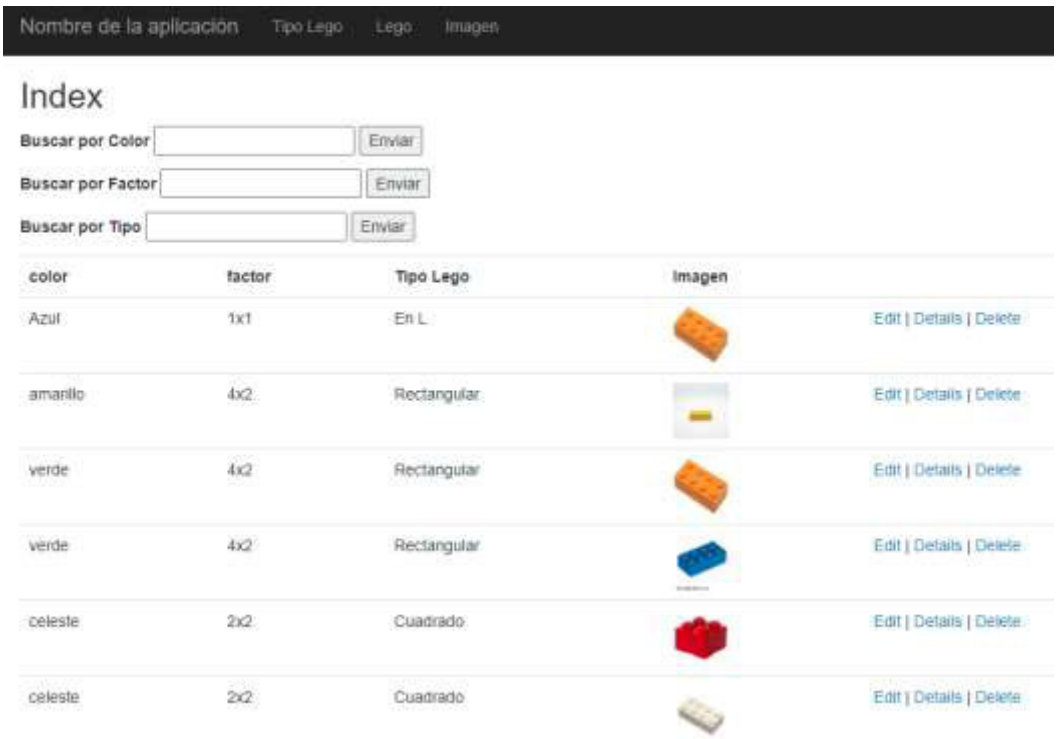


Figura 3.8. Búsqueda por color, facto y Tipo

Al realizar la búsqueda por cualquiera de estos parámetros, la vista resultante presentara la cantidad de elementos que coinciden con el parámetro buscado, así como un listado de los elementos encontrados durante la búsqueda.

En la Figura 3.9 se muestra el resultado de la búsqueda por color, se puede ver que se encontró 1 elemento, así como un listado de dicho elemento.

Figura 3.9. Resultado de la búsqueda por color

El funcionamiento de búsqueda por factor y por tipo es de manera similar, al presentado en la Figura 3.9.

Figura 3.10. Resultado de la búsqueda por factor

Figura 3.11. Resultado de la búsqueda por tipo

3.2 Conclusiones

- Al finalizar con presente Trabajo de Integración Curricular se dispone de un subsistema capaz de almacenar imágenes de fichas Lego para un sistema de clasificación de fichas Lego basado en imágenes.
- Para el desarrollo del subsistema de almacenamiento se empleó Entity Framework y SQL Server. También para poder demostrar el uso del funcionamiento del subsistema se implementó una aplicación web basada en la arquitectura cliente – servidor y se usó el patrón MVC Modelo – Vista – Controlador.
- Para el manejo de la información en la base de datos se empleó Entity Framework con el enfoque Database – First, con lo cual primero se creó la base de datos, y con esto se puedo implementar segmentos de cogido que interactúen con la base de datos, sin requerir el uso de sentencias escritas en el lenguaje SQL.
- Para el desarrollo de la interfaz web de la aplicación web, se usaron plantillas Razor, las cuales permiten mezclar código escrito en lenguaje de programación C# y en el lenguaje de marcado HTML.
- Para la simulación de la clasificación, así como de ingreso de imágenes de fichas Lego se usó, una variable específica, así como métodos aleatorios que permitieron probar el funcionamiento del subsistema de almacenamiento.
- Para el almacenamiento de las imágenes se empleó un arreglo de bytes los cuales se guardan en la base de datos, y los mismo son leídos para su visualización en las respectivas vistas, con el uso de funciones.

- Entre las pruebas de funcionamiento del subsistema de almacenamiento se implementó métodos de búsqueda por parámetros que simula otra funcionalidad del subsistema de consulta, que facilita la búsqueda y visualización de la información almacenada en la base de datos.
- Para el ingreso de las imágenes estas son almacenadas en formato JPG, las cuales son validadas en el subsistema de adquisición de imágenes.
- Las vistas de la aplicación web poseen un formato simple ya que únicamente se usó plantillas Razor para configurar las respectivas páginas web que se presentan al usuario.

3.3 Recomendaciones

- Sería recomendable el almacenamiento de la información mediante registros en lugar de un arreglo de bytes, para que así la base de datos no sea demasiado pesada.
- Se podría considerar el uso de Entity Framework Core, ya que es compatible con plataformas Linux, además de ser compatibles con aplicaciones .NET Framework.
- Se podría usar el enfoque *Code-First* para el manejo de datos con Entity Framework, debido a que no se tendría que modificar directamente la base datos en caso de ser necesario.
- Se podría tomar en consideración mejorar el diseño de la base de datos para cuando el sistema tenga una mayor complejidad.
- Es preferible implementar el *stub* en una aplicación web usando el patrón MVC para que el usuario pueda interactuar desde el navegador web con mayor comodidad.
- El diseñar la base de datos con los atributos correspondientes a las necesidades ayuda a que el desarrollo del subsistema sea constante y no se deban hacer cambios significativos en el mismo.

4 REFERENCIAS BIBLIOGRÁFICAS

[1] Microsoft, «Microsoft,» 14 Octubre 2020. [En línea]. Disponible: https://docs.microsoft.com/en-us/ef/ef6/fundamentals/working-with-dbcontext. [Último acceso: 08 01 2022].

[2] «Microsoft,» 10 Abril 2021. [En línea]. Disponible: https://docs.microsoft.com/es-es/aspnet/core/tutorials/first-mvc-app/start-mvc?WT.mc_id=dotnet-35129-website&view=aspnetcore-6.0&tabs=visual-studio. [Último acceso: 09 01 2022].

[3] Redaccion APD, «apd,» 08 Junio 2021. [En línea]. Disponible: https://www.apd.es/metodologia-kanban/. [Último acceso: 05 01 2022].

[4] G. B., «HOSTINGER TUTORIALES,» martes 08 2021. [En línea]. Disponible: https://www.hostinger.es/tutoriales/que-es-github. [Último acceso: 09 01 2022].

[5] EmilianoMusso, «C#Corner,» 13 Octubre 2020. [En línea]. Disponible: https://www.c-sharpcorner.com/article/entity-framework-introduction-using-c-sharp-part-one/. [Último acceso: 07 enero 2022].

[6] Microsoft, «Microsoft,» 15 Septiembre 2021. [En línea]. Disponible: https://docs.microsoft.com/en-us/dotnet/framework/data/adonet/ef/overview. [Último acceso: 08 01 2022].

[7] Microsoft, «microsoft,» 14 Octubre 2020. [En línea]. Disponible: https://docs.microsoft.com/en-us/ef/ef6/modeling/designer/workflows/database-first. [Último acceso: 09 01 2022].

[8] J. M. Alarcon, «campusMVP,» 12 Marzo 2018. [En línea]. Disponible: https://www.campusmvp.es/recursos/post/entity-framework-code-first-database-first-y-model-first-en-que-consiste-cada-uno.aspx. [Último acceso: 06 01 2022].

[9] Entendiendo las expresiones Lambda", *Geeky Theory*, 2022. [En línea]. Disponible: https://geekytheory.com/entendiendo-las-expresiones-lambda/. [Último acceso: 21-Feb- 2022].

5 ANEXOS

ANEXO I. Código fuente del prototipo implementado

ANEXO I

El anexo se adjunta de manera digital y contiene los archivos de código creados en el desarrollo del subsistema.