ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE INGENIERÍA ELECTRÍCA Y ELECTRÓNICA

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

SUBSISTEMA DE ADQUISICIÓN DE IMÁGENES

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

DIEGO SEBASTIÁN GARCÍA NAVARRETE

diego.garcia@epn.edu.ec

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

david.mejia@epn.edu.ec

DMQ, febrero 2022

CERTIFICACIONES

Yo, Diego Sebastián García Navarrete 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.

DIEGO SEBASTIÁN GARCÍA NAVARRETE

Certifico que el presente trabajo de integración curricular fue desarrollado por Diego Sebastián García Navarrete, bajo mi supervisión.

ING. RAÚL 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.

DIEGO SEBASTIÁN GARCÍA NAVARRETE

ING. RAÚL DAVID MEJÍA NAVARRETE, M.Sc.

DEDICATORIA

A mis padres Mireya Navarrete y Luis García, así como a mis hermanos Santiago y Matías, por guiarme en este camino y nunca permitir que me rinda.

AGRADECIMIENTO

A mis papas y hermanos, por su amor, esfuerzo y paciencia en esta etapa de mi vida.

A mis amigos, Michael, Josué y Katy, por soportarme tal como soy y estar pendientes de mí.

A mi tutor, Ing. David Mejía, por la guía brindada para el desarrollo del mismo.

A todos los profesores de la Escuela Politécnica Nacional, por compartir sus conocimientos.

ÍNDICE DE CONTENIDO

RESUMEN

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

El subsistema permite adquirir imágenes de fichas Lego para procesarlas y obtener información que será enviada a un subsistema de clasificación.

El presente documento está estructurado en 3 capítulos:

En el primero capítulo se presenta conceptos teóricos sobre las herramientas y tecnologías de software que fueron empleadas, en este caso se menciona el patrón arquitectónico Modelo-Vista-Controlador, así como el funcionamiento del *framework* Bootstrap, adicionalmente se menciona el espacio de nombres System.Drawing que fue empleado en el procesamiento de la imagen, al igual que el enfoque de la metodología de desarrollo ágil Kanban que fue empleado en el desarrollo del subsistema.

En el segundo capítulo, se presenta el proceso de desarrollo del subsistema, así como el proceso para generar las historias de usuario. Adicionalmente se presenta el diagrama de clases del subsistema, para posteriormente describir el proceso en la implementación de las funcionalidades del subsistema.

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

PALABRAS CLAVE: ASP.NET, MVC, Kanban, Bootstrap, procesamiento de imagen.

ABSTRACT

In this Curriculum Integration Work, the design and implementation of an image acquisition subsystem for a Lego tile classification system based on images was carried out.

The subsystem allows acquiring images of Lego chips to process them and obtain information that will be sent to a classification subsystem.

This document is structured in 3 chapters:

In the first chapter, theoretical concepts about the software tools and technologies that were used are presented. In this case, the Model-View-Controller architectural pattern is mentioned, as well as the operation of the Bootstrap framework. Additionally, the System.Drawing library is mentioned that was used in the processing of the image, as well as the Kanban agile development methodology approach that was used in the development of the subsystem.

In the second chapter, the subsystem development process is presented, as well as the process to generate user stories. Additionally, the class diagram of the subsystem is presented, to later describe the process in the implementation of the functionalities of the subsystem.

In the third chapter, the tests of operation of each of the elements of the subsystem are presented, later the conclusions and recommendations of the present Curriculum Integration Work are presented.

KEYWORDS: ASP.NET, MVC, Kanban, Bootstrap, image processing.

.

1 INTRODUCCIÓN

En el presente Trabajo de Integración Curricular se plantea el desarrollo de un subsistema de adquisición de imágenes, que procese la imagen y obtenga información de la misma. Este subsistema forma parte de un sistema de clasificación de fichas Lego basado en imágenes. Para este subsistema se empleará ASP.NET MVC que es un patrón arquitectónico en el desarrollo de aplicaciones web, así también se hará uso del *framework* Bootstrap que permitirá dar a las interfaces de usuario una mejor apariencia visual.

El subsistema permite adquirir una imagen para realizar su procesamiento y obtener información que será usada por el subsistema de clasificación, es por esto que la información que será enviada consta de tres matrices correspondientes a los componentes de color rojo, verde y azul.

Para verificar que el subsistema implementado cumple con los diferentes parámetros requeridos, se ha incluido un *stub* para simular el funcionamiento de un subsistema diferente y que muestre que efectivamente se está realizando el procesamiento de la imagen, con esto se permite integrar este subsistema al subsistema de clasificación.

Para este Trabajo de Integración Curricular se hará uso de una aplicación ASP.NET MVC para verificar el procesamiento de la imagen.

1.1 Objetivo general

Desarrollar un subsistema de adquisición de imágenes mediante el uso de ASP.NET MVC, que permita procesar una imagen Lego para su posterior clasificación.

1.2 Objetivos específicos

1. Analizar las herramientas requeridas para este proyecto.
2. Determinar los componentes que conforman el subsistema de procesamiento de imágenes, acorde a los requerimientos establecidos.
3. Implementar el subsistema de procesamiento de imágenes considerando los componentes identificados para el mismo.

1.3 Alcance

En el presente Trabajo de Integración Curricular se plantea desarrollar un prototipo de subsistema de adquisición de imágenes que permita procesarlas, generar matrices para sus componentes de color rojo, verde y azul e implementar un *stub* para su uso en el subsistema de clasificación de fichas Lego basado en imágenes.

El prototipo contará con tres vistas: 1) la vista principal, permitirá acceder a la vista de carga de imagen; 2) la vista de carga de imagen, que permitirá la carga de la imagen desde el explorador de archivos de Windows, este explorador de archivos permitirá al usuario cargar imágenes únicamente de formato JPG; 3) la vista de lista de imágenes, que mostrará una lista de imágenes procesadas.

Las tres vistas serán responsivas, permitiendo que se adapten ante cualquier tamaño de ventana del navegador web.

La vista de carga de imagen contara con tres opciones las cuales son:

1) Procesar la imagen, que previo a realizar cualquier acción, verificará que se haya cargado un archivo y que sea del formato correcto para luego procesarlo, esta opción realizará una redimensión de pixeles en ancho y alto a un valor de 224 x 224 pixeles y a partir de esta imagen se generan matrices RGB (*Red-Green-Blue*), esta información se la implementará en un *stub*, el cual será usado en la clasificación. Una vez procesada la imagen se mostrará la segunda vista que muestra una lista de imágenes previamente procesadas.
2) Volver a cargar una imagen en caso de que haya realizado una selección errónea.
3) Presentar una lista con la información de las imágenes previamente procesadas.

1.4 Marco teórico

1.4.1 ASP.NET MVC

ASP.NET MVC es un *framework*[1] creado por Microsoft, se lo considera un marco de presentación, ligero, gratuito y de código abierto con una gran capacidad de prueba,

[1] ***Framework:*** Entorno o marco de trabajo que ofrece una estructura base para elaborar un proyecto.

permitiendo la implementación de aplicaciones web usando los lenguajes de programación C#[2] o VB.NET[3]. Las aplicaciones web desarrolladas bajo este *framework* pueden usar el patrón arquitectónico Modelo-Vista-Controlador (MVC[4]).

El patrón MVC da la posibilidad de manejar de forma independiente las distintas responsabilidades de una aplicación web en componentes lógicos que se encargan de gestionar una tarea en concreto permitiendo que las aplicaciones sean flexibles, escalables y dinámicas, además de permitir un mejor control de código escrito en el lenguaje HTML[5] [1].

El funcionamiento de MVC ocurre luego de que el servidor reciba una petición enviada a través del protocolo HTTP[6]. Esta petición se provoca debido a que el usuario manipula la interfaz de la aplicación desde su navegador web.

El servidor web recibe esta petición y la remite al controlador, para determinar a qué controlador debe entregarse una petición en particular se emplea un procedimiento conocido como enrutamiento.

El controlador se encarga de acceder al modelo y consultar la información solicitada, realizar las acciones respectivas y elegir la vista que será remitida al usuario con la información solicitada [1], la misma que será enviada al usuario mediante una respuesta enviada a través del protocolo HTTP.

Modelo: Se refiere al código que permite acceder a los datos que la aplicación web usa. Los objetos del modelo devuelven información que se recupera desde una base de datos y guardan la información en la misma.

El modelo es un conjunto de clases que describe los datos con los que se está trabajando, así como las reglas para manipularlos [2].

El modelo tiene total independencia de la vista y el controlador. Usualmente, el modelo puede ser construido usando Entity Framework[7].

---

[2] **C#:** Lenguaje de programación modero y multiparadigma basado en objetos.
[3] **BV.NET:** Lenguaje de programación orientado a objetos con beneficios del *framework* .NET.
[4] **MVC:** Arquitectura de software que separa los datos de aplicación, interfaz de usuario y lógica de control en tres componentes distintos.
[5] **HTML:** *HyperText Markup Lenguaje*, lenguaje de marcado que permite estructurar y desplegar contenido web mediante etiquetas.
[6] **HTTP:** *Hipertext Transfer Protocol*, protocolo cliente-servidor que permite la transferencia de datos y recursos.
[7] **Entity Framework:** Marco de trabajo de mapeo objeto relacional que ofrece un mecanismo automatizado para acceder y guardar información en una base de datos.

Vista: Se refiere al código que permite presenta la información al usuario mediante una interfaz gráfica con la que el usuario pueda interactuar.

Una vista es una plantilla escrita usando HTML, CSS[8], JavaScript, y que tiene código escrito en un lenguaje como C#, mediante una sintaxis de marcado de Razor[9]. Las vistas permiten separar el código escrito en un lenguaje de programación de la interfaz de usuario con el fin de establecer una separación entre ambos en una aplicación MVC [3].

Las vistas suelen ser agrupadas considerando las características de la aplicación, de esta forma se tiene mejor organizada la aplicación, lo que facilita la búsqueda de las vistas cuando se tiene una característica en común entre ellas, además de permitir la modificación de estas, sin tener que realizar cambios en otras partes de la aplicación.

Controlador: El controlador es una clase constituida por un grupo de métodos relacionados y que actúa como un intermediario entre la vista y el modelo en la gestión de flujo de datos entre ellos. Estos métodos se conocen como métodos de acción y se usan para procesar solicitudes HTTP del usuario que son enviadas desde el navegador web. El controlador toma la decisión de que método de acción ejecutar y manipular la información del modelo requerido y en última instancia puede devolver una vista para presentar al usuario la información solicitada.

En la Figura 1.1 se presenta un ejemplo de controlador, el cual se denomina `HomeController` y dispone de un método de acción denominado `Index`; este método de acción devuelve un `ActionResult` que permite entregar una vista; la vista que se devuelva será determinada por ASP.NET de entre todas las vistas disponibles. ASP.NET retorna la vista considerando convención en lugar de configuración, es decir si los nombres de los modelos, vistas y controladores siguen un patrón, ASP.NET determina la vista que está asociada a un método de acción especifico de un controlador y usará el método asociado a este controlador.

```
8   {
        0 referencias
9       public class HomeController : Controller
10      {
            0 referencias
11          public ActionResult Index()
12          {
13              return View();
14          }
15      }
16  }
```

Figura 1.1. Ejemplo de controlador

[8] **CSS:** *Cascading Style Sheets*, es un lenguaje de estilos usados para otorgar características visuales y estéticas a contenido web.
[9] **Razor:** Motor de vistas que optimiza la presentación de páginas simplificando la sintaxis.

En resumen, la responsabilidad del controlador es: manejar la lógica de la aplicación web, procesar las solicitudes del usuario, trabajar con el modelo y seleccionar la vista que se empleará para generar la interfaz del usuario que se enviará al usuario [3].

Para que el pedido pueda ser remitido al controlador especifico se emplea el enrutamiento. En el pedido HTTP enviado por el usuario se incluye la URL[10] del recurso solicitado. La URL tiene la siguiente estructura: `http://domain:port/{controller}/{action}/{parameter}`, como se aprecia se define el controlador que se empleará, así como el método de acción que deberá ejecutar dicho controlador y los parámetros requeridos por ese método de acción.

En el ejemplo de la Figura 1.2 se presenta una URL, en la cual se puede observar el nombre del controlador, el nombre del método de acción y los parámetros de entrada, de ser el caso, que han sido especificados.

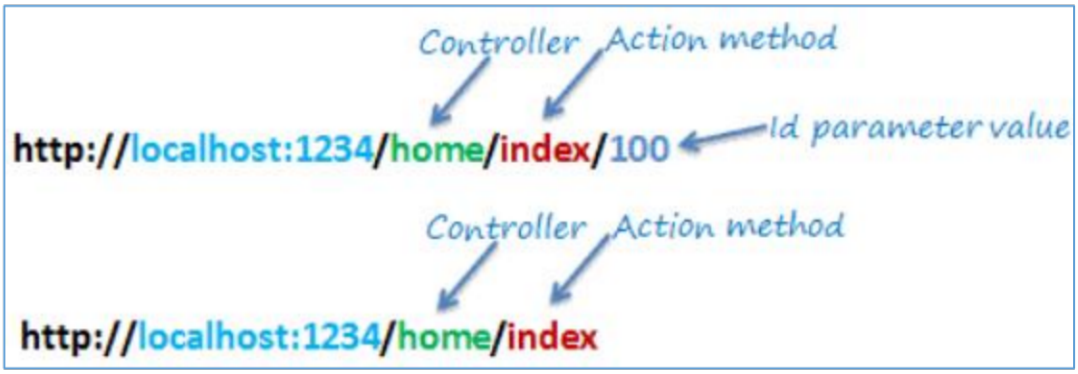

Figura 1.2. Ejemplo de URL de petición [4]

Un método de acción del controlador devuelve algo que se denomina resultado de acción [5].

Para generar los resultados de acción se usa como base a la clase `ActionResult`, la cual representa los resultados de toda posible acción.

Las distintas clases que se emplean para gestionar los resultados de acción se conocen como tipos de acción.

En la Tabla 1.1 se muestra una lista de los diferentes tipos de acción y su comportamiento en ASP.NET MVC.

[10] **URL:** *Uniform Resource Locator,* dirección única otorgada a un recurso dentro de la red.

Tabla 1.1. Tipos de acción en ASP.NET MVC

Tipo de acción	Método	Descripción
`ViewResult`	`View ()`	Devuelve una vista.
`PartialViewResult`	`PartialView ()`	Devuelve una vista parcial.
`JsonResult`	`Json ()`	Devuelve datos en formato JSON[11].
`RedirectResult`	`Redirect ()`	Redirecciona hacia una nueva URL indicada.
`FIleResult`	`File ()`	Devuelve contenido de un archivo en binario.
`EmptyResult`	`EmptyResult ()`	Devuelve un valor vacío.

1.4.2 System.Drawing, Bitmap

La clase `Bitmap` se encuentra en el espacio de nombres `System.Drawing`, dicho espacio de nombres proporciona funcionalidades como trabajar con imágenes definidas por pixeles, convertir imágenes de un tipo de dato a otro, entre otras.

La clase `Bitmap` encapsula un mapa de bits generada a través de GDI+[12], que consiste en los datos de una imagen gráfica y sus atributos representados como pixeles. Un mapa de bits es un objeto utilizado para trabajar con imágenes definidas por datos de pixeles [6].

Para almacenar mapas de bits dentro de un archivo, se tiene muchos formatos estándar que GDI+ admite, entre los cuales se tiene: BMP[13], JPG[14], entre otros.

La clase `Bitmap` dispone de diversos constructores que permiten entre otras actividades, escalar una imagen a valores de ancho y alto especificados, establecer el formato de pixel, etc.

En la Figura 1.3. se presenta un ejemplo de uso de objetos creados usando la clase `Bitmap`, en la línea 88 se crea una instancia `Image` y se le puebla con datos que fueron extraídos desde la entrada estándar, en la línea 89 se crea un objeto `Bitmap` con la resolución de 250 pixeles de ancho y 250 pixeles de alto, usando el objeto `Image`.

---

[11] **JSON:** *JavaScript Object Notation,* basado en JavaScript y siendo un formato ligero en el intercambio de datos presentando fácil escritura y lectura.

[12] **GDI+:** *Graphics Device Interface,* interfaz que permite desarrollar aplicaciones que contienen gráficos y texto formateado independientemente del dispositivo.

[13] **BMP:** Formato de archivo para almacenar imágenes en mapa de bits independientemente del dispositivo de visualización.

[14] **JPG:** *Joint Photographic Experts Group,* formato de compresión de imágenes con alta calidad.

```
88          Image inImage = Image.FromStream(Ingreso.InputStream);
89          Bitmap bmp = new Bitmap(inImage, 250, 250);
```

Figura 1.3. Ejemplo de uso de la clase `Bitmap`

Los métodos de la clase `Bitmap` dan la posibilidad de manipular las propiedades del objeto u obtener información del mismo. En la Figura 1.4 se presenta un ejemplo del método `GetPixel`, en las líneas 95 y 96 se aprecia el uso de un objeto `Color` que se usará para almacenará la información de un pixel, extraído en una posición especifica empleando coordenadas en el eje x y en el eje y (200, 100).

```
93          Image inImage = Image.FromStream(Ingreso.InputStream);
94          Bitmap bmp = new Bitmap(inImage, 224, 224);
95          Color col1 = new Color();
96          col1 = bmp.GetPixel(200, 100);
```

Figura 1.4. Ejemplo de uso de método `GetPixel`

1.4.3 Bootstrap

Bootstrap es un *framework* de código abierto basado en HTML, CSS y JavaScript usado para el desarrollo del front-end para generar páginas y sitios web interactivos y responsivos, permite la fácil adaptación a distintos tamaños de dispositivos, mediante el uso de elementos de maquetación, por ejemplo: ventanas, botones, iconos, entre otros, dando a la interfaz de usuario una mejor apariencia visual.

Los elementos de maquetación que proporciona Bootstrap permiten al programador enfocarse más en el desarrollo del sitio o aplicación web y no emplear mucho tiempo en escribir código CSS para la personalización de la interfaz de usuario.

Bootstrap permite que el programador no tenga que escribir código usando HTML o CSS en los archivos de su aplicación web, sino que haga uso de Bootstrap desde un sitio externo haciendo referencia al archivo de Bootstrap disponible en un enlace especifico. En la Figura 1.5 se presenta la referencia al archivo CSS que se encuentra en un sitio externo y que será usado como hoja de estilo para la aplicación web.

```
<link rel="stylesheet" href="https://use.fontawesome.com/releases/v5.7.2/css/all.css">
```

Figura 1.5. Ejemplo de referencia a archivo Bootstrap externo

JavaScript: Es un lenguaje interpretado y ligero orientado a objetos con funciones de primera clase, conocido como lenguaje de *script* para aplicaciones o sitios web [7]. JavaScript puede usarse en el lado del cliente o en lado del servidor.

1.4.4 Metodología de desarrollo ágil Kanban

La metodología ágil Kanban consiste en la gestión de un proyecto de manera general. Su implementación implica la mejora continua del proceso mediante flujos de trabajo visual [8].

Kanban permite realizar una visualización de las tareas en el desarrollo de un proyecto y su correspondiente seguimiento, el seguimiento de las tereas se realiza dividiéndolas en tres grupos que son: por hacer, en proceso y hecho. Toda tarea pasa por cada uno de estos grupos y se asigna una cantidad de tiempo para mantener un equilibrio en el flujo del desarrollo del proyecto.

Product BackLog: Es un listado ordenado y priorizado de los requisitos necesarios para la implementación de un proyecto [9].

Tablero Kanban: El tablero Kanban es una herramienta útil para la asociación y la visualización del flujo de trabajo, está dividido en filas y columnas. Cada columna muestra una fase del proceso y las filas representa los diferentes tipos de actividades requeridos para completar con éxito el proyecto [10].

En el tablero Kanban se presentan los tres grupos: por hacer, en proceso y hecho, con las tareas específicas que se encuentran en cada estado.

2 METODOLOGÍA

En el presente Trabajo de Integración Curricular se empleó la metodología Kanban, con la cual se definieron las tareas que debían ser desarrolladas; dichas tareas fueron colocadas en un tablero, el cual cuenta con las columnas: tareas pendientes, tareas en proceso y tareas finalizadas.

Como parte del diseño, se realizó una entrevista al director a cargo del presente Trabajo de Integración Curricular, de esta forma se definieron las historias de usuario y los requerimientos de prototipo a ser desarrollado.

Una vez examinada la información obtenida a través de la entrevista y las historias de usuario, se identificaron las tareas que forman parte del *product backlog*.

Posteriormente se realizó el diseño del prototipo, empezando por definir la arquitectura a emplearse, generando el diagrama de clases correspondiente, así como los *sketches* de las interfaces de usuario.

Las herramientas utilizadas son: el IDE Visual Studio 2018, ASP.NET MVC 5, Microsoft SQL Management Studio 19 y Bootstrap 4.

Para el procesamiento de la imagen se usó la clase `Bitmap` del espacio de nombres `System.Drawing`, misma que permitió el redimensionamiento de la imagen y generar información de la misma.

Posteriormente se desarrollaron las vistas y controlador necesarios para responder las peticiones enviadas desde el navegar web.

Finalmente se realizaron pruebas de cada funcionalidad implementada para verificar el correcto funcionamiento del prototipo, dichas pruebas fueron: pruebas de ingreso de una imagen, listado de imágenes, responsividad del prototipo, entre otras.

2.1 Historias de usuario

En base a la entrevista realizada al director del Trabajo de Integración Curricular, se obtuvieron los requerimientos funcionales y requerimientos no funcionales del prototipo.

Requerimientos funcionales

- Permitir al usuario ingresar imágenes de formato JPG.

- La imagen ingresada debe ser redimensionada a 224 pixeles de ancho y 224 pixeles de alto.
- La imagen redimensionada debe ser procesada para obtener tres matrices, una para el componente rojo, una para el componente verde y una para el componente azul.

Requerimientos no funcionales

- El prototipo se realizará usando ASP.NET MVC 5.
- El prototipo web debe tener comportamiento responsivo para lo cual se aplicará el *framework* Bootstrap 4.
- El prototipo correrá sobre un servidor IIS[15].

En la Tabla 2.1 se presentan las historias de usuario, las cuales siguen el siguiente formato: `HU-numero`, `Nombre`, `Descripción`.

Tabla 2.1. Historias de usuario

Identificador	Nombre	Descripción
HU-01	Adquisición de la imagen	La aplicación permitirá al usuario ingresar una imagen.
HU-02	Procesar imagen	La aplicación permitirá procesar una imagen.
HU-03	Listar imágenes	La aplicación permitirá listar las imágenes ingresadas.
HU-04	Comportamiento responsivo	La aplicación tendrá comportamiento responsivo.

2.1.1 Product backlog

Las actividades que forman parte del *product backlog* fueron obtenidas con base al resultado de la entrevista realizada al director del Trabajo de Integración Curricular. La Tabla 2.2 presenta el *product backlog*.

[15] **IIS:** *Internet Information Server,* servidor para sistemas operativos Windows, da la capacidad de convertir un ordenador en un servidor web.

Tabla 2.2. *Product backlog*

Tareas
Generar la interfaz gráfica que permita adquirir la imagen
Generar la interfaz gráfica que permita presentar las imágenes ingresadas
Generar la base de datos que permita almacenar la información de la imagen ingresada

2.2 Diseño

2.2.1 Arquitectura

EL subsistema se implementó con base en una arquitectura Cliente-Servidor siguiendo el patrón Modelo-Vista-Controlador, el servidor web *IIS Express* se encarga de recibir las peticiones y entregar los recursos solicitados mediante el protocolo *HTTP*; ASP.NET se encarga de procesar las peticiones que son solicitadas por el servidor web, a partir de solicitudes enviadas desde el navegador web por consecuencia de interacciones del usuario con la página web. ASP.NET es el encargado de enrutar la solicitud al correspondiente controlador y determinar el método de acción para procesar una imagen o mostrar una vista asociada. Como parte del *stub*, se empleó una base de datos en SQL Server para almacenar la información de la imagen procesada. Finalmente, el modelo se encarga de almacenar y recuperar la información de la base de datos empleando Entity Framework para la gestión de los datos.

2.2.2 Diseño de la base de datos

Para determinar la información que debe ser generada del procesamiento de la imagen, se obtuvo la información de las características de la imagen que serán empleadas por el subsistema de clasificación. En la Figura 2.1 se presenta la tabla que contiene los atributos de la imagen procesada que será almacenada como identificador, ancho, alto y una secuencia de bits.

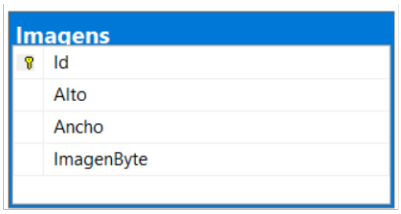

Figura 2.1. Tabla `Imagens`

2.2.3 Diseño de clases

En la Figura 2.2 se presenta el diagrama de clases generado para este proyecto, las clases que se emplearán son `Imagen`, `ImagenCapaNegocio`, `ImagenViewModel`, `ListaImagenViewModel`, `ImagenController` e `ImagenDAL`. Las cuatro primeras clases se derivan de la clase `Object`; mientras que la clase `ImagenController` se deriva de la clase abstracta `Controller` lo que permite que el programador no tenga que preocuparse de ciertos detalles requeridos en el `Controller`, si no únicamente de las funcionalidades que debería definir en la clase del controlador; por último, se tiene la clase `ImagenDAL` que se deriva de la clase `DbContext` lo que permite contar con los métodos necesarios para el acceso a la base de datos.

La clase `Imagen` representan el modelo que permite manejar las propiedades de la imagen y la clase `ImagenCapaNegocio` permite gestión de la base de datos. Las clases `ImagenViewModel` y `ListaImagenViewModel` sirven para separar la lógica del modelo y de las vistas además de manejar el formato de presentación de las imágenes en las vistas. La clase `ImagenController` representa el controlador, entre las acciones más importantes de esta clase se tiene la de procesar la imagen. La clase `ImagenDAL` es empleada para manipular el acceso a la base de datos por medio de métodos proporcionados por Entity Framework.

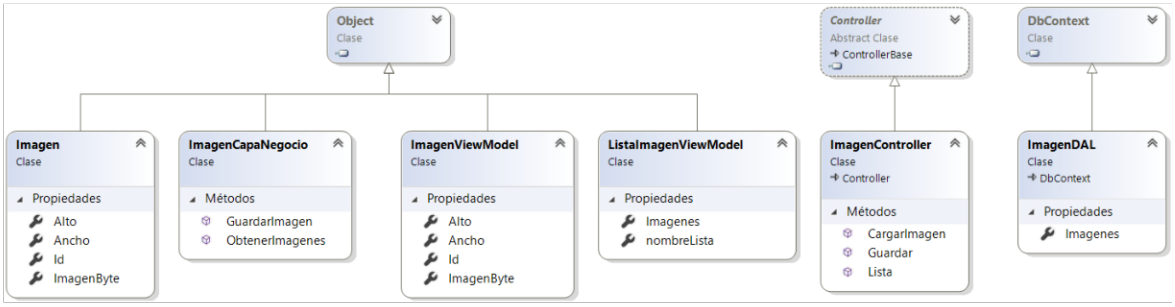

Figura 2.2. Diagrama de clases

2.2.4 Sketches de interfaz de usuario

Los *sketches* se desarrollarán acorde a los requerimientos obtenidos. Se crearon tres *sketches* denominados `_Layout`, `CargarImagen` y `Lista`.

En la Figura 2.3 se presenta el *sketch* `_Layout` correspondiente a la interfaz de usuario principal compartida en la aplicación web, la cual cuenta con un menú lateral desplegable

con las opciones `CARGAR IMAGEN` para la adquisición de la imagen, e `IDENTIFICAR LEGO` y `BUSQUEDA` que representan elementos del *stub* para la funcionalidad que deberá realizar el siguiente componente.

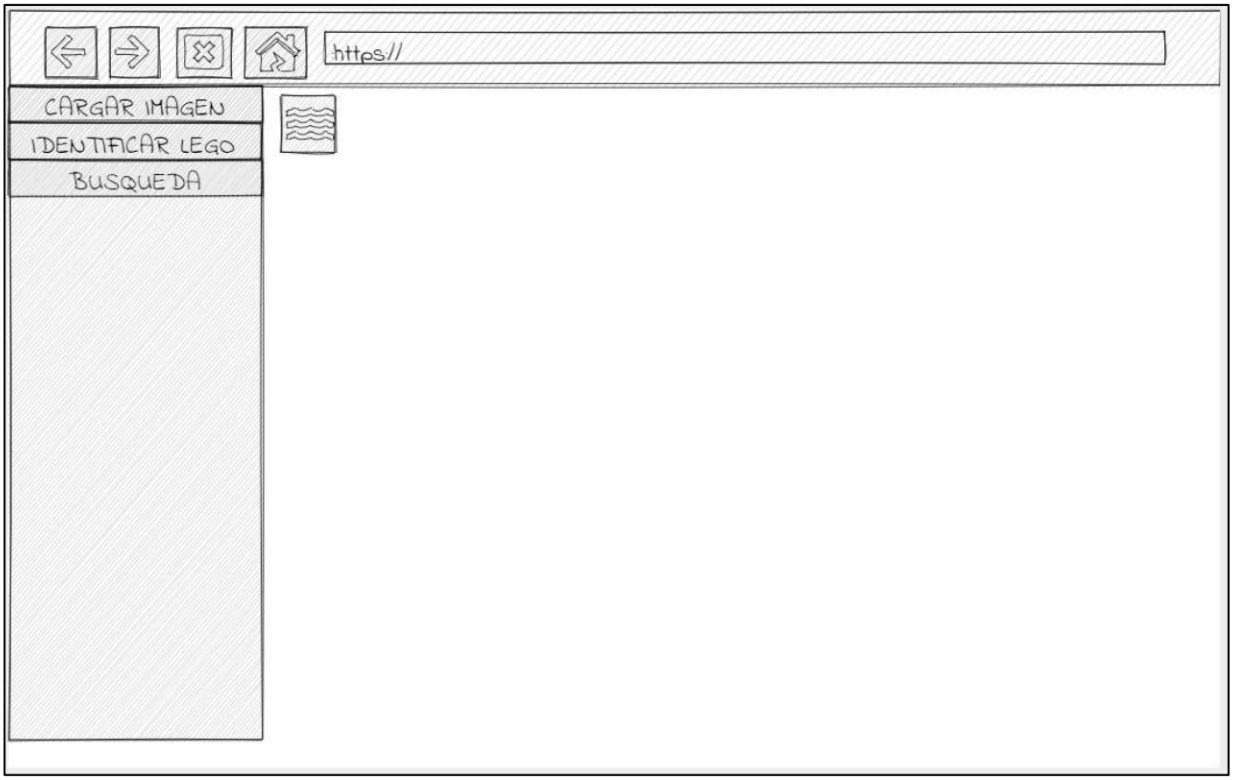

Figura 2.3. *Sketch* `_Layout`

En la Figura 2.4 se presenta el *sketch* `CargarImagen` que permite adquirir la imagen, cuenta con las siguientes opciones:

- `ELEGIR ARCHIVO`: permite al usuario ingresar la imagen.
- `PROCESAR`: permite procesar la imagen ingresada.
- `REINICIAR`: permite volver a cargar una imagen en caso de una selección errónea.
- `LISTA DE IMÁGENES`: permite presentar la lista de imágenes procesadas.

En la Figura 2.5 se presenta el *sketch* `Lista` que permite mostrar una lista de imágenes procesadas, adicionalmente cuenta con un botón denominado `CARGAR IMAGEN` que permite volver a cargar una imagen en caso de que el usuario lo desee.

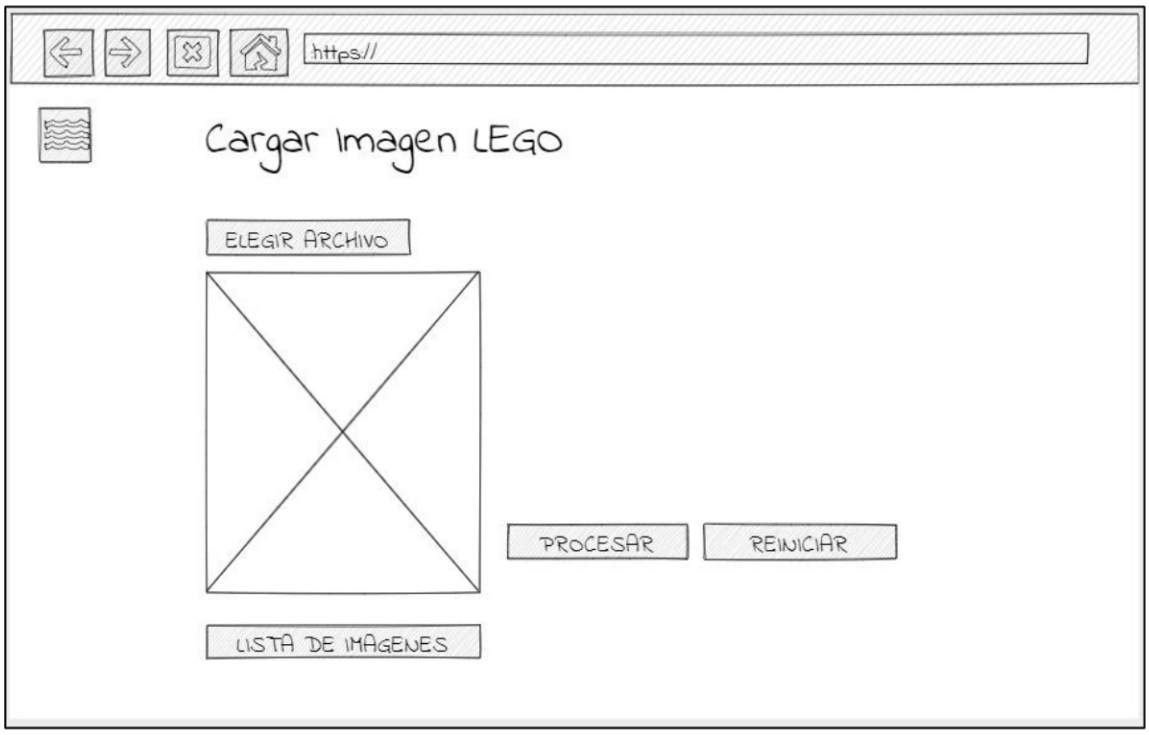

Figura 2.4. *Sketch* de carga de imagen

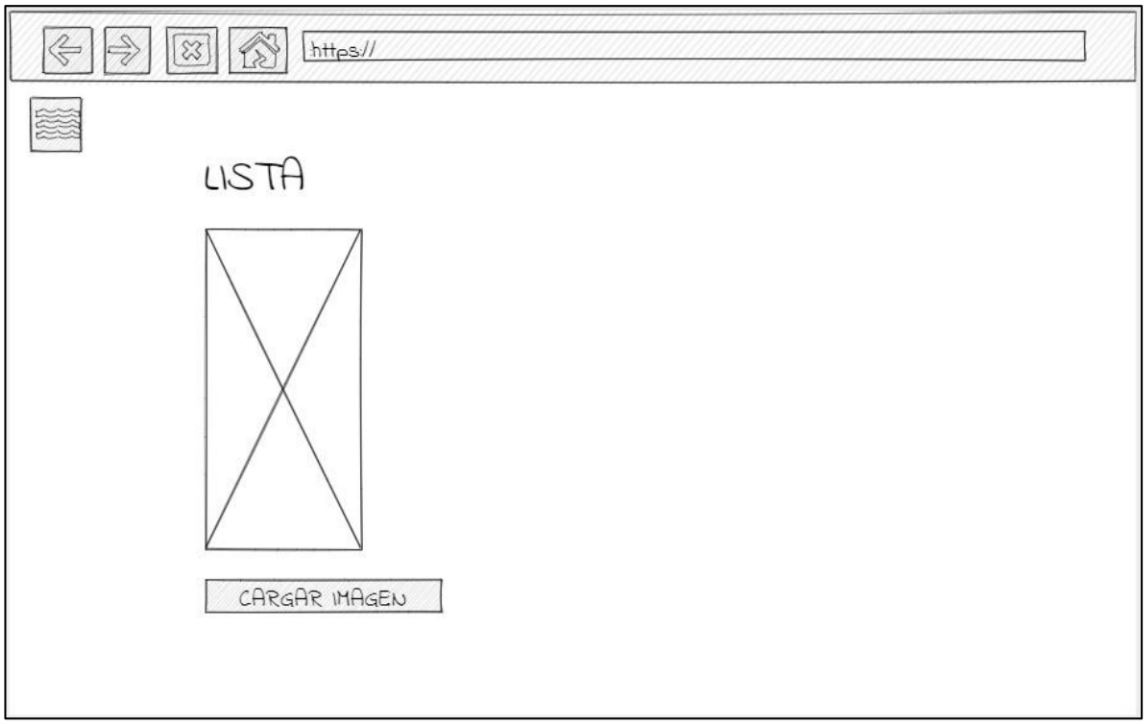

Figura 2.5. *Sketch* de lista de imágenes.

2.3 Implementación

Para el presente Trabajo de Integración Curricular, se emplearon los *frameworks* presentes en la Tabla 2.3.

Tabla 2.3. *Frameworks* empleados

Paquete	Versión
EntityFramework	6.4.4
bootstrap	4.0.0

2.3.1 Modelos

En la Figura 2.6 se muestra el modelo `Imagen` creado, el cual cuenta con atributos que permiten manejar las propiedades de la imagen, tales como identificador, ancho, alto y una secuencia de bits (líneas 15 a 18). Adicionalmente, en la línea 8 se emplea `DataAnnotations` que permite identificar al atributo `Id` como la llave primaria en la base de datos marcándolo con la palabra reservada `Key`.

```
12  public class Imagen
13  {
14      [Key]
        0 referencias
15      public int Id { get; set; }
        2 referencias
16      public int Alto { get; set; }
        2 referencias
17      public int Ancho { get; set; }
        2 referencias
18      public byte[] ImagenByte { get; set; }
19  }
20  }
```

Figura 2.6. Clase `Imagen`

2.3.2 Acceso a la base de datos

Para el acceso y manipulación de la base de datos se trabajó con EntityFramework, se crearon dos clases denominadas `ImagenCapaNegocio` e `ImagenDAL`. La clase `ImagenDAL` hereda de la clase `DbContext`, lo que permite realizar operaciones como crear u obtener información de la base de datos. En la Figura 2.7 en la línea 16 se observa que la clase `ImagenDAL` hereda de la clase `DbContext`, mientras que en la línea 18 se

definen las propiedades `get` y `set`, las cuales se encargan de manipular las imágenes que se envían o se consultan de la base de datos.

```
10  using System.Data.Entity;
11  namespace ProcesamientoImagenLego.AccesoDatos
12  {
13      //Esta clase debe ser derivada de DbContext para ganar
14      //todos los metodos necesarios para que sql trabaje por detras
15      //sin que se tenga que escribir lineas de codigo en sql
            4 referencias
16      public class ImagenDAL : DbContext
17      {
                2 referencias
18          public DbSet<Imagen> Imagenes { get; set; }
19      }
20  }
```

Figura 2.7. Clase `ImagenDAL`

La clase `ImagenCapaNegocio` contiene los métodos correspondientes para recuperar imágenes y guardar una imagen en la base de datos. En la Figura 2.8 se puede observar el método `ObtenerImagenes` (líneas 14 a 18) encargado de recupera un conjunto de imágenes de la base de datos, y el método `GuardarImagen` (líneas 20 a 30) encargado de guarda la imagen en la base de datos.

```
12      public class ImagenCapaNegocio
13      {
            1 referencia
14          public List<Imagen> ObtenerImagenes()
15          {
16              ImagenDAL imagenDAL = new ImagenDAL();
17              return imagenDAL.Imagenes.ToList();
18          }
19
            1 referencia
20          public Imagen GuardarImagen(Imagen im)
21          {
22
23              ImagenDAL imagenDAL = new ImagenDAL();
24              //Se agrega en el DbC la imagen
25              imagenDAL.Imagenes.Add(im);
26              //Se guarda los cambios en la base
27              imagenDAL.SaveChanges();
28              //Retorna la imagen para trabajar con ella
29              return im;
30          }
31      }
32  }
```

Figura 2.8. Clase `ImagenCapaNegocio`

2.3.3 Vistas

Para la generación de las vistas se trabajó con el *framework* Bootstrap y sintaxis Razor que permite combinar código escrito en lenguaje *HTML* con código escrito en lenguaje C#. Se crearon tres vistas denominadas `_Layout`, `CargarImagen` y `Lista`.

La vista `_Layout` muestra la interfaz de usuario inicial en donde se puede acceder a los diferentes subsistemas, la vista `CargarImagen` permite ingresar una imagen y la vista `Lista` muestra un listado de imágenes con sus respectivos atributos que han sido procesadas. En la Figura 2.9 se muestra la vista `_Layout`, la cual cuenta con un menú para acceder a la vista `CargarImagen`.

Figura 2.9. Vista `_Layout`

En la Figura 2.10 se observan los archivos correspondientes para el uso de Bootstrap en las vistas (líneas 9 a 13), lo cual permite que las vistas presentadas en el navegador web tengan comportamiento responsivo, mejor apariencia visual, entre otras.

```
8   <!-- Bootstrap -->
9   <link rel="stylesheet" href="~/Content/bootstrap.min.css">
10  <script src="~/Scripts/jquery-3.4.1.slim.min.js"></script>
11  <script src="~/Scripts/popper.min.js"></script>
12  <script src="~/Scripts/bootstrap.min.js"></script>
```

Figura 2.10. Bootstrap en la vista `_Layout`

En la Figura 2.11 se muestra la vista `CargarImagen`, la cual cuenta con las opciones de ingresar un archivo, procesar la imagen, reiniciar para volver a ingresar un archivo y lista de imágenes.

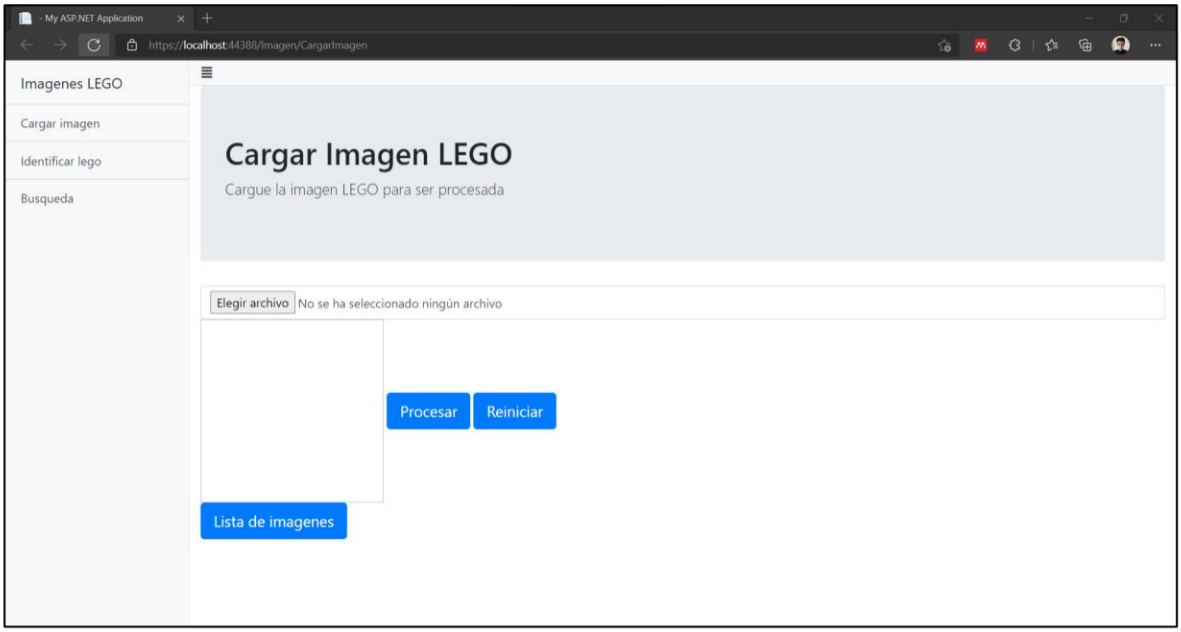

Figura 2.11. Vista `CargarImagen` para ingresar una imagen

En la Figura 2.12 se muestra una sección de código de la vista `CargarImagen` para la adquisición y procesamiento de la imagen, y listar imágenes, para ello se definieron dos formularios.

Entre las líneas 50 y 61 se define el primer formulario para la adquisición de la imagen, en la línea 10 se utiliza el método `BeginForm`, el cual es un HTML *Helpers* que permite crear el formulario asociado al método de acción `Guardar` del controlador `ImagenController`, y se indica que, cuando el usuario presione en el botón de tipo `submit` se enviará una solicitud `Post` al controlador incluyendo el parámetro denominado `Ingreso`, el mismo que corresponde a la imagen ingresada.

Adicionalmente se permite el ingreso de archivos de formato JPG o JPEG como se observa en la línea 56.

Desde las líneas 63 a 65 se define el segundo formulario encargado de mostrar la vista con el listado de imágenes ingresadas, en este caso, el formulario está asociado al método de acción `Lista` del controlador `ImagenController`.

```
50      <div>
51          @using (Html.BeginForm("Guardar", "Imagen", FormMethod.Post, new { enctype = "multipart/form-data" }))
52          {
53              @Html.AntiForgeryToken()
54              <div>
55                  @*Especificador de tipo de archivo unico para ser seleccionado solo .jpg*@
56                  <input type="file" class="form-control" id="inLego" accept="image/jpg, image/jpeg" onchange="validarFormato()" name="Ingreso" required />
57                  <img id="imagenLego" height="240" width="240" />
58                  <input type="submit" name="BtnGuardar" value="Procesar" onclick="existeImagen()" class="btn btn-primary btn-lg" />
59                  <input type="button" name="BtnReiniciar" value="Reiniciar" onclick="Reiniciar()" class="btn btn-primary btn-lg" />
60              </div>
61          }
62
63          <form action="/Imagen/Lista" method="post">
64              <input type="submit" name="BtnLista" value="Lista de imagenes" class="btn btn-primary btn-lg" />
65          </form>
66
67      </div>
68  </body>
```

Figura 2.12. Código de la vista `CargarImagen`

En la Figura 2.13 se muestra la vista `Lista`, la cual contiene un listado de imágenes con sus respectivas características que han sido procesadas.

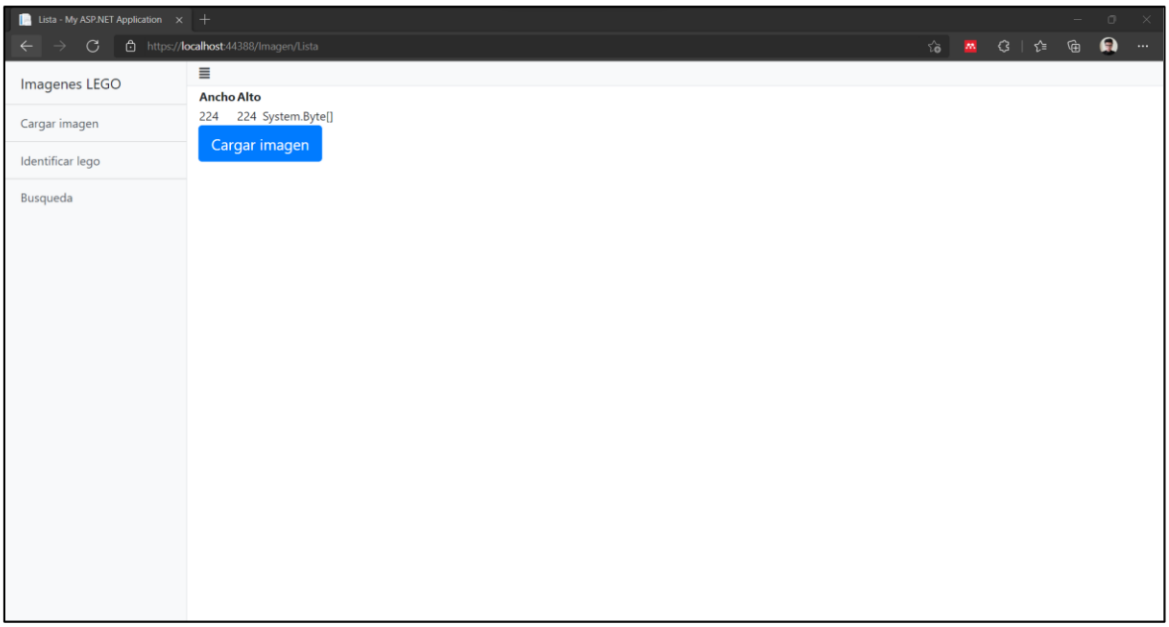

Figura 2.13. Vista `Lista` para visualizar imágenes procesadas

En la Figura 2.14 se muestra el código empleado en la vista `Lista` para generar una tabla que contenga las imágenes procesadas.

De las líneas 16 a 29 se emplea la etiqueta `<table>` para generar una tabla, en las líneas 18 y 19 se establece los nombres de las columnas empleando la etiqueta `<th>`, que para este caso son 2, sus propiedades ancho y alto, mientras que desde las líneas 24 a 26 se usa la etiqueta `<tr>` para indicar que estas celdas son de contenido y corresponde a cada columna.

Desde las líneas 21 a 28 se emplea una sentencia de código `foreach`, la cual permite iterar entre las imágenes procesadas disponibles en la base de datos y presentar las características de cada una de ellas. Adicionalmente, en la línea 34 se crea un enlace utilizando el método `ActionLink`, dicho enlace está asociado al método de acción `CargarImagen` del controlador `ImagenController`, el cual se encarga de mostrar la vista `CargarImagen` en caso de que el usuario desee ingresar una nueva imagen.

```
15  <div>
16      <table>
17          <tr>
18              <th>Ancho</th>
19              <th>Alto</th>
20          </tr>
21          @foreach (ImagenViewModel item in Model.Imagenes)
22          {
23              <tr>
24                  <td>@item.Ancho</td>
25                  <td>@item.Alto</td>
26                  <td>@item.ImagenByte</td>
27              </tr>
28          }
29      </table>
30  </div>
31
32  <div>
33
34      @Html.ActionLink("Cargar imagen", "CargarImagen", null, new { @class = "btn btn-primary btn-lg" })
35
36  </div>
```

Figura 2.14. Código de la vista `Lista`

2.3.4 Controlador

EL prototipo cuenta con dos controladores denominados `HomeController` e `ImagenController`, donde `HomeController` fue generado por defecto al crear la solución en Visual Studio, mientras que el controlador `ImagenController` se encarga de procesar una imagen, almacenar la información de la imagen en la base de datos y listar las imágenes procesadas.

Cuando el usuario interactúe con la vista, el navegador web generará una solicitud que provocará que el servidor llame a un método de acción específico del controlador. Los métodos de acción presentes en el controlador `ImagenController` son `CargarImagen`, `Lista` y `Guardar`. El método de acción `CargarImagen` se encarga de mostrar la vista para el ingreso de imágenes. El método de acción `Lista` se encarga de obtener la información de las imágenes presentes en la base de datos. El método de acción `Guardar` se encarga de redimensionar la imagen, obtener las matrices de los componentes de color rojo, verde y azul, y finalmente almacenar la información en la base de datos.

En la Figura 2.15 se observa el código empleado para redimensionar la imagen, en la línea 65 emplea el atributo `HttpPost` para establecer que el método de acción `Guardar` corresponde a una solicitud tipo POST en la que se incluye la imagen. En la línea 75 se redimensiona la imagen a valores de 224 pixeles de ancho y 224 pixeles de alto mediante el uso de un constructor proporcionado por la clase `Bitmap`. Desde la línea 78 a 88 se crean los atributos empleados en el procesamiento de la imagen correspondientes a ancho y alto de la imagen, y tres matrices correspondientes para los componentes de color rojo, verde y azul. Desde las líneas 92 a 106 se emplea la sentencia de código `for` para recorrer los pixeles de la imagen redimensionada, en cada pixel se ejecuta la función `GetPixel` para obtener el valor de su componente de color rojo, verde y azul, y poblar con dichos valores las respectivas matrices.

```
65      [HttpPost]
66      [ValidateAntiForgeryToken]
        0 referencias
67      public ActionResult Guardar([Bind(Include = "Ingreso")] Imagen imagen, HttpPostedFileBase Ingreso)
68      {
69          if (Ingreso != null && Ingreso.ContentLength > 0)
70          {
71              //Se recibe un archivo desde la entrada estandar
72              Image inImage = Image.FromStream(Ingreso.InputStream);
73              //Se establece un objeto Bitmap con la imagen recibida y
74              //se asigna ancho y alto en pixeles
75              Bitmap bmp1 = new Bitmap(inImage, 224, 224);
76              Color col1 = new Color();
77
78              int wi1 = bmp1.Width;
79              int he1 = bmp1.Height;
80
81              imagen.Ancho = bmp1.Width;
82              imagen.Alto = bmp1.Height;
83
84              //Se crea tres matrices que representan al componente de color rojo
85              //componente de color verde y componente de color azul
86              int[,] redsrt = new int[wi1, he1];
87              int[,] greensrt = new int[wi1, he1];
88              int[,] bluesrt = new int[wi1, he1];
89
90              //Se recorre cada pixel del objeto Bitmap obteniendo su color
91              //y almacenandolo en su respectiva matriz
92              int red = 0, green = 0, blue = 0;
93              for (int i = 0; i < wi1; i++)
94              {
95                  for (int j = 0; j < he1; j++)
96                  {
97                      col1 = bmp1.GetPixel(i,j);
98                      red = col1.R;
99                      green = col1.G;
100                     blue = col1.B;
101
102                     redsrt[i, j] = red;
103                     greensrt[i, j] = green;
104                     bluesrt[i, j] = blue;
105                 }
106             }
107             ImageConverter converter = new ImageConverter();
108
109             imagen.ImagenByte = (byte[])converter.ConvertTo(inImage, typeof(byte[]));
110         }
```

Figura 2.15. Código `ImagenController` para el procesamiento de la imagen

Una vez realizado el procesamiento de la imagen, se comprueba que el modelo se validó. En la Figura 2.16 se realiza el almacenamiento de la imagen en la base de datos en caso de que el modelo sea válido y se devuelve la vista correspondiente (líneas 116 a 118), si el modelo no es válido en la línea 122 se devuelve la vista `CargarImagen`.

```
112             //Si el modelo es valido, se almacena en la base de datos
113             //Si el modelo no es valido, se retorna a la vista CargarImagen
114             if (ModelState.IsValid)
115             {
116                 ImagenCapaNegocio cp = new ImagenCapaNegocio();
117                 cp.GuardarImagen(imagen);
118                 return RedirectToAction("Lista");
119             }
120             else
121             {
122                 return View("CargarImagen");
123             }
124         }
125     }
126 }
```

Figura 2.16. Almacenamiento de la imagen

En la Figura 2.17 se observa el método de acción `Lista` del controlador `ImagenController`, este método se encarga de presentar la vista con la lista de imágenes. Desde la línea 29 a 34 se obtienen las imágenes de la base de datos usando el método `ObtenerImagenes`. En las líneas 37 a 50 se usa la sentencia de código `foreach` para recorrer las imágenes obtenidas y presentarlas en una lista de la vista correspondiente.

```
27      public ActionResult Lista()
28      {
29          ListaImagenViewModel mostrarLista = new ListaImagenViewModel();
30
31          ImagenCapaNegocio negocio = new ImagenCapaNegocio();
32          List<Imagen> imagenes = negocio.ObtenerImagenes();
33
34          List<ImagenViewModel> imagenViewModels = new List<ImagenViewModel>();
35
36          //Recorre la lista de imagenes
37          foreach (Imagen im in imagenes)
38          {
39              ImagenViewModel imagenVM = new ImagenViewModel();
40              imagenVM.alto = im.Alto;
41              imagenVM.ancho = im.Ancho;
42              imagenVM.imagenByte = im.ImagenByte;
43
44              //Muestra todos los detalles de las imagenes agregandolos a una lista
45              imagenViewModels.Add(imagenVM);
46          }
47
48          mostrarLista.Imagenes = imagenViewModels;
49          mostrarLista.nombreLista = "Imagenes ingresadas";
50          return View(mostrarLista);
51      }
```

Figura 2.17. Método de `Lista` acción del controlador `ImagenController`

3 RESULTADOS, CONCLUSIONES Y RECOMENDACIONES

En este capítulo se presentan las pruebas de funcionamiento y los resultados obtenidos, de igual manera se presentan las conclusiones y recomendaciones producto de este Trabajo de integración curricular

3.1 Resultados

Se realizaron pruebas de funcionalidad con el objetivo de comprobar el correcto funcionamiento del prototipo realizado. Para esto se realizó la adquisición de imágenes y archivos de diferentes formatos para comprobar el funcionamiento de las validaciones. Adicionalmente, se realizaron pruebas de procesamiento de imágenes, para comprobar que se obtiene información de las mismas y se la almaceno en la base de datos para observar su correcto funcionamiento. Finalmente, se realizaron pruebas del comportamiento responsivo del prototipo y así comprobar que se adapta ante tamaños de navegador web reducidos.

3.1.1 Interfaz de usuario compartida

La interfaz de usuario generada por la vista `_Layout` presenta un menú desplegable en el que se tiene acceso a las funcionalidades de los diferentes subsistemas, como se observa en la Figura 3.1.

Figura 3.1. Interfaz de usuario generada por la vista `_Layout`

Al presionar en la opción de menú denominada Cargar Imagen, se muestra la interfaz de usuario generada por la vista `CargarImagen`.

Se aclara que las opciones Identificar lego y Búsqueda no fueron desarrolladas debido a que no son parte de este componente, ya que representan el *stub* que emula las tareas que los subsistemas de clasificación y búsqueda deben realizar.

En la Figura 3.2 se observa que la interfaz de usuario generada por la vista `_Layout` es compartida para toda la aplicación independientemente del botón presionado en el menú lateral.

Figura 3.2. Interfaz de usuario para el subsistema de clasificación

3.1.2 Adquisición de imagen

Al presionar en el botón Cargar imagen que se observa en la Figura 3.1, la aplicación presenta la interfaz de usuario generada por la vista `CargarImagen` que permite adquirir la imagen, visualizarla, procesarla, reingresar una nueva imagen en caso de una selección errónea y mostrar un listado de imágenes.

El botón Listado de imágenes dirige a la interfaz de usuario generada por la vista `Lista`, en la cual se muestra un listado de las imágenes procesadas.

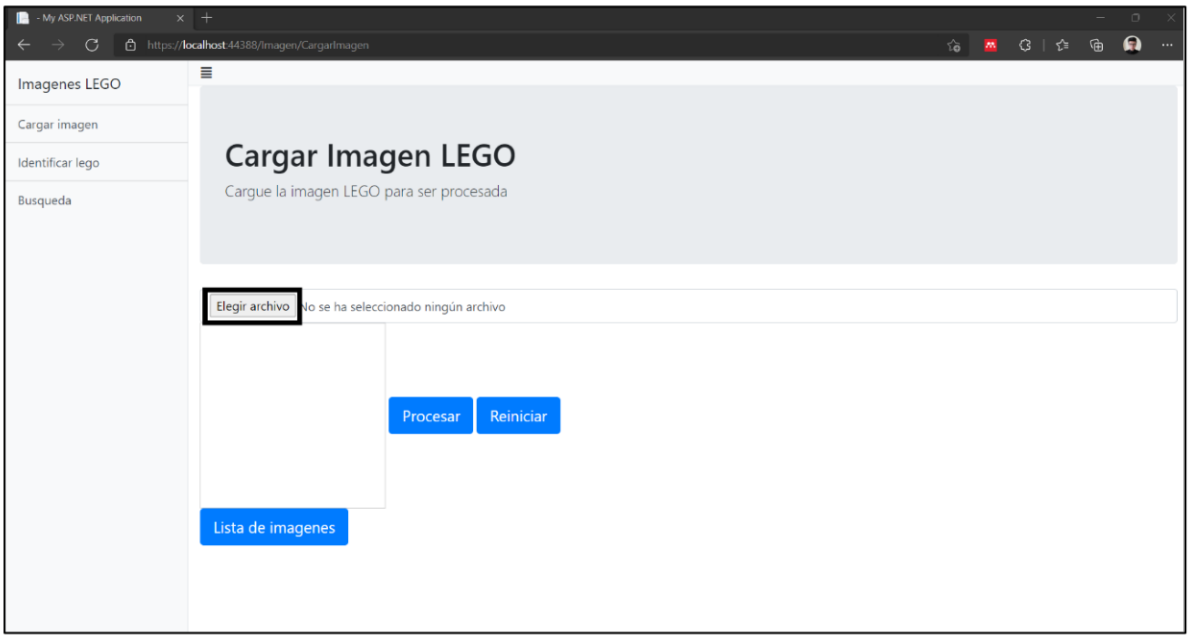

Figura 3.2. Interfaz de usuario compartida

Al presionar en el botón Elegir archivo de la Figura 3.2, se abre un explorador de archivos de Windows desde el cual se adquiere la imagen.

Una vez adquirida la imagen, esta es visualizada, dicha interfaz de usuario se observa en la Figura 3.3.

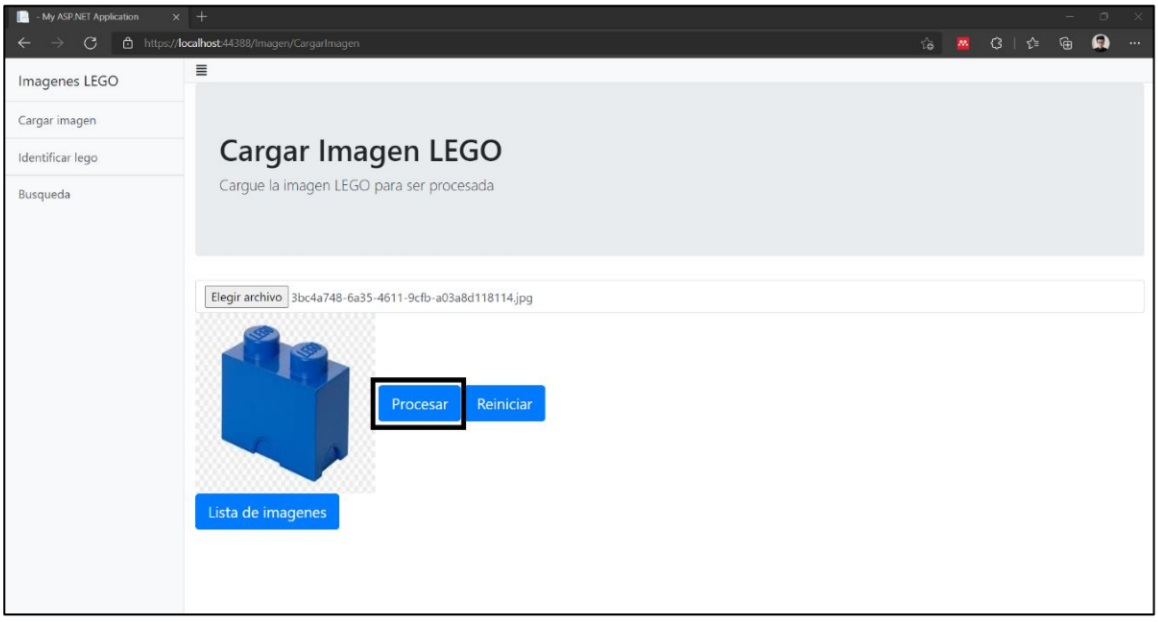

Figura 3.3. Imagen adquirida

3.1.3 Procesamiento de la imagen

Una vez adquirida la imagen, se presiona en el botón Procesar de la Figura 3.3 para que se envíe la información al servidor web, y la información de la imagen sea almacenada en la base de datos.

El resultado del procesamiento de la imagen se puede observar en la Figura 3.4 correspondiente a la interfaz de usuario generada por la vista `Lista`, en la cual se muestra un listado de imágenes obtenido de la base de datos, dicho listado presenta información de las imágenes como ancho y alto.

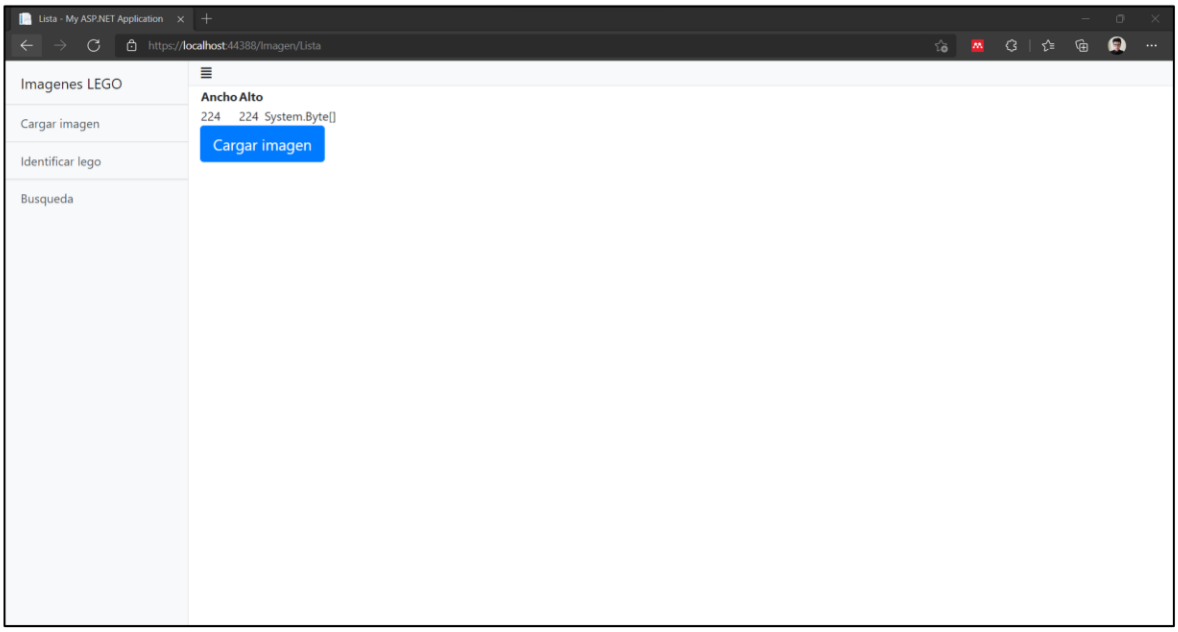

Figura 3.4. Interfaz de usuario generada por la vista `Lista` con la información de las imágenes procesadas

3.1.4 Validaciones

Se realizaron validaciones para permitir el ingreso de archivos de formato JPG y que exista una imagen cargada previo al procesamiento.

Para comprobar la validación de formato, se intenta ingresa un archivo de formato PNG y un archivo de formato PDF.

La aplicación no permite adquirir archivos diferentes al formato JPG, por lo cual muestra una alerta informando al usuario, como se observa en la Figura 3.5.

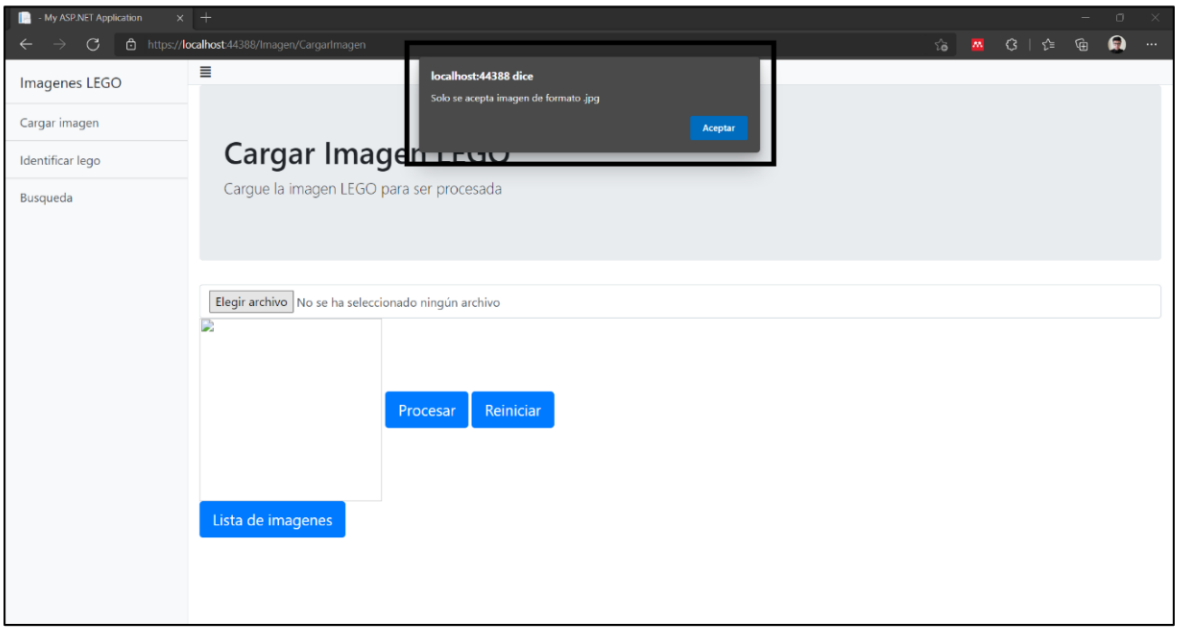

Figura 3.5. Validación de formato de la imagen

Se realizó la validación de que exista una imagen cargada previo a su procesamiento.

En la Figura 3.6 se muestra el caso en donde el usuario presiona en el botón Procesar sin haber ingresado una imagen,

Dicha acción provoca que se muestre una alerta al usuario informando que debe cargar una imagen cuando se presiona en el botón Procesar.

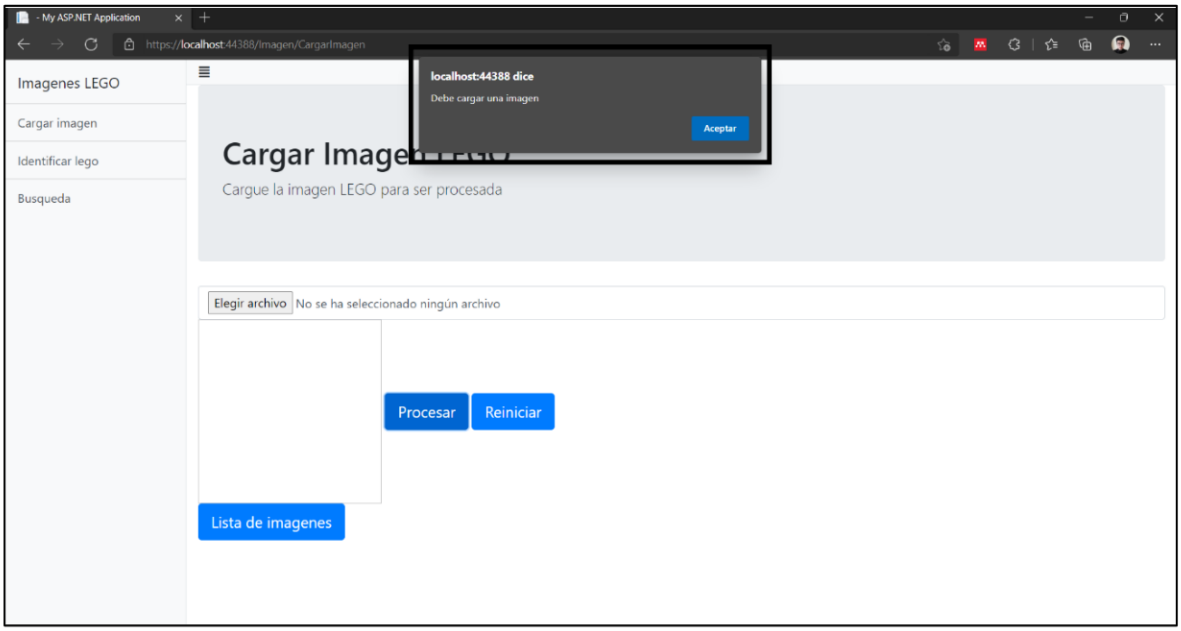

Figura 3.6. Validación de imagen cargada

3.1.5 Comportamiento responsivo de la aplicación

El comportamiento responsivo se verificó reduciendo el tamaño del explorador web, en la Figura 3.7 se observa que menú se minimiza cuando ocupa un espacio considerable en el navegador web.

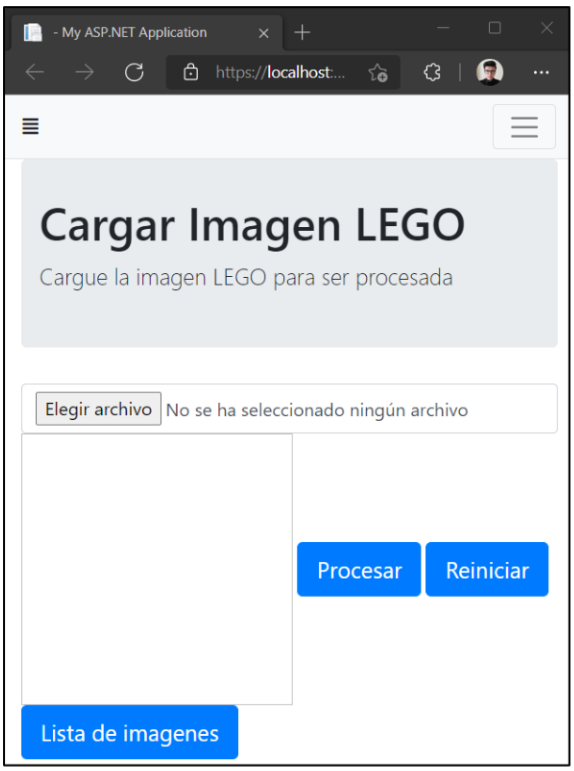

Figura 3.7. Comportamiento responsivo de la aplicación

3.2 Conclusiones

- Al finalizar con el presente Trabajo de Integración Curricular se dispone de un subsistema capaz de adquirir y procesar imágenes de fichas Lego para un sistema de clasificación de fichas Lego basado en imágenes.
- Para el procesamiento de la imagen se empleo la clase `Bitmap` del espacio de nombres `System.Drawing`, con la cual se redimensionó la imagen, y con esto se pudo obtener la información que será entregada al subsistema de clasificación.
- Para incluir características de responsividad y menú desplegable en las interfaces de usuario, se empleó Bootstrap el cual proporcionó archivos para el manejo de las mismas, sin requerir la escritura de código CSS o JavaScript.

- Se creó el modelo `Imagen` para simular al subsistema de clasificación al que se le entregará la información del procesamiento de la imagen y así comprobar el correcto funcionamiento del subsistema de adquisición de imágenes, además se creó una base de datos en SQL Server en donde se almacenó la información de las imágenes y permitió disponer de la misma para que estas puedan ser listadas por el subsistema de adquisición.
- Se creó una base de datos en SQL Server para simular al subsistema de clasificación al que se le entregará la información del procesamiento de la imagen y así comprobar el correcto funcionamiento del subsistema de adquisición de imágenes, además de que permitió disponer de la información de las imágenes para que estas puedan ser listadas por el subsistema de adquisición.

3.3 Recomendaciones

- Dado que el subsistema de adquisición acepta imágenes únicamente en formato JPG, se recomienda continuar con el desarrollo para que acepte diversos formatos de imágenes.
- Se recomienda continuar con el desarrollo en el aspecto visual del subsistema mediante el uso de características proporcionadas por Bootstrap que no fueron tomadas en cuenta en el desarrollo de este Trabajo de integración Curricular.
- Se recomienda usar la biblioteca `ImageSharp` para simplificar el procesamiento de imágenes, debido a que cuenta con mejores características que las definidas el espacio de nombres `System.Drawing`.
- En un futuro, se puede añadir nuevos criterios de procesamiento de la imagen como filtros para remover el ruido o variación de contraste para mejorar la visibilidad de algunos elementos dentro de la imagen, de esta formar se entregaría información más completa y detallada de la imagen al subsistema de clasificación.
- Se podría considerar nuevas validaciones como peso mínimo o máximo de la imagen ingresada.
- Se recomienda implementar una funcionalidad que permita al usuario escoger el tipo de compresión de la imagen procesada.

- Se recomienda desarrollar un módulo que permita redimensionar la imagen a valores de ancho y alto, que puedan ser ingresados a través de una interfaz de usuario.

4 REFERENCIAS BIBLIOGRÁFICAS

[1] S. Smith, “Información general de ASP.NET Core MVC | Microsoft Docs,” 03 Octubre 2021. Disponible en: https://docs.microsoft.com/es-es/aspnet/core/mvc/overview?view=aspnetcore-6.0. [Último acceso: 29 11 2021].

[2] J. Galloway, B. Wilson, and S. Allen, *Professional ASP.NET MVC 5*, 1st ed. Indianapolis, 2014.

[3] S. Smith, “Vistas de ASP.NET Core MVC | Microsoft Docs,” 10 Diciembre 2021. Disponible en: https://docs.microsoft.com/es-es/aspnet/core/mvc/views/overview?view=aspnetcore-6.0. [Último acceso: 29 12 2021].

[4] “ASP.NET MVC Tutorials,” 2020. Disponible en: https://www.tutorialsteacher.com/mvc. [Último acceso: 29 11 2021].

[5] S. Walther, “Introducción al controlador MVC deC#ASP.net () | Microsoft Docs,” 13 Mayo 2021. Disponible en: https://docs.microsoft.com/es-es/aspnet/mvc/overview/older-versions-1/controllers-and-routing/aspnet-mvc-controllers-overview-cs. [Último acceso: 29 11 2021].

[6] “Bitmap Class (System.Drawing) | Microsoft Docs.” Disponible en: https://docs.microsoft.com/en-us/dotnet/api/system.drawing.bitmap?view=dotnet-plat-ext-6.0. [Último acceso: 29 11 2021].

[7] mozilla.org, “JavaScript | MDN,” 21 12 2021” 21 Diciembre 2021. Disponible en: https://developer.mozilla.org/es/docs/Web/JavaScript. [Último acceso: 30 12 2021].

[8] EALDE, “La metodología ágil Kanban: en qué consiste y para qué sirve,” 04 Agosto 2020. Disponible en: https://www.ealde.es/metodologia-agil-kanban/. [Último acceso: 30 11 2021].

[9] I. Cañete, “Qué es un ‘product backlog’ y cuál es su función | BBVA,” 11 Marzo 2019. Disponible en: https://www.bbva.com/es/que-es-un-product-backlog-y-cual-es-su-funcion/. [Último acceso: 30 11 2021].

[10] “¿Qué es un tablero Kanban? | Kanbanize,” *2021*. Disponible en: https://kanbanize.com/es/recursos-de-kanban/primeros-pasos/que-es-tablero-kanban. [Último acceso: 30 11 2021].

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