

Memoria Proyecto

DEMO de Gestor de contenido
Tienda ONLINE de cartas TCG

ALEJANDRO LEAL LASERNA

Resumen	5
Abstract	7
1. Introducción	8
1.1 Justificación del proyecto	9
1.2 Caso de estudio	9
1.3 Estructura del documento	10
1.4 Descripción general	11
1.5 Objetivos	12
1.6 Requisitos	13
2. Análisis	14
2.1 Casos de uso	14
2.2 Diagrama de clases	16
2.3 Diagramas de secuencia	17
3. Diseño	20
3.1 Capa presentación	21
3.1.1 Login	22
3.1.2 Navegador	23
3.1.3 Sección Lista	24
3.1.4 Carta	25
3.1.4 Sección Admin	26
3.2 Capa Lógica	28
3.3 Capa de datos	31
4. Implementación	32
4.1 Tecnologías	32
4.1.1 JAVA	32
4.1.2 HTML	33
4.1.3 CSS	34
4.1.4 JAVASCRIPT	35
4.1.5 SQL	36

4.2 Tecnologías	38
4.2.1 Tecnologías visuales	38
4.2.2 Hardware	38
4.3 Detalles de implementación	39
4.3.1 Ejemplo de conexión	39
4.3.2 Ejemplo de Carta	40
4.3.3 Ejemplos de funcionamiento Login Usuario	50
4.3.4 Ejemplos de funcionamiento Login Administrador	53
4.3.5 Ejemplos de funcionamiento de Sliders	56
5. Plan de proyecto	57
5.1 Equipo de trabajo	57
5.2 Plan de trabajo	57
5.3 Análisis de costes	
6. Trabajos futuros	59
7. Conclusiones	60
Anexos	61

ENLACES DE INTERES:

GITHUB: <https://github.com/ALEXLEALdev/Proyecto-Integrador>

YOUTUBE: https://www.youtube.com/watch?v=LptHEDX5J_U&ab_channel=AlejandroLeal

RESUMEN

Nuestra tienda online de cartas de Digimon s cuenta con un gestor totalmente editable a través de menús y formularios, diseñados con las tecnologías web más actuales como HTML, JAVA, JAVASCRIPT y MySQL. Hemos priorizado el dinamismo en su contenido y personalización, asegurando así una experiencia única para nuestros usuarios. Además, nos hemos esforzado por posicionar el portal de manera óptima en internet.

Buscamos que nuestro sitio sea escalable, permitiendo la fácil inserción de nuevos objetivos y elementos. Aunque en este caso presentamos una tienda online de cartas como ejemplo, estamos comprometidos a adaptar nuestra plataforma para satisfacer las necesidades específicas de los amantes de Digimon. ¡Explora nuestro catálogo y únete a la aventura digital!

ABSTRACT

Our Digimon card online store features a fully editable manager through menus and forms, designed with the latest web technologies such as HTML, JAVA, JAVASCRIPT and MySQL. We have prioritized dynamism in its content and customization, ensuring a unique experience for our users. Additionally, we have made efforts to position the portal optimally on the internet.

We aim for our site to be scalable, allowing for easy insertion of new modules that may be created in the future. While in this case, we showcase a online cards shop as an example, we are committed to adapting our platform to meet the specific needs of Digimon enthusiasts. Explore our catalog and join the digital adventure!

1. Introducción

En el mundo actual del comercio electrónico, las empresas necesitan mantener una presencia en Internet sólida y actualizada. Aunque existen numerosas soluciones para establecer una tienda en línea, es común recurrir a plataformas de comercio electrónico como Shopify, CardMarket, entre otras. Estas plataformas ofrecen la facilidad de actualizar y gestionar productos mediante paneles de administración, pero en ocasiones requieren de cierta capacitación previa para su manejo eficiente.

En este proyecto, se propone el desarrollo de un gestor de tiendas en línea con requisitos específicos que no se encuentran completamente satisfechos por las plataformas existentes. Por un lado, se busca que el uso del gestor sea intuitivo y accesible incluso para usuarios sin experiencia previa en el manejo de tiendas en línea. Se pretende que la gestión de productos sea sencilla y directa, sin necesidad de una curva de aprendizaje extensa.

Además, se plantea la posibilidad de crear múltiples cartas de otras colecciones en línea con estructuras y diseños personalizados, alojadas en diferentes carpetas del servidor. Esto permitiría dirigir cada tienda mediante dominios individuales, proporcionando una experiencia única para cada producto o línea de productos. Cada producto podría tener su propia identidad visual y funcionalidades específicas, adaptadas a las necesidades de los clientes o productos que representan.

El gestor de tiendas en línea también contaría con una página principal que funcionaría como un catálogo centralizado, mostrando una lista de productos disponibles y un sistema de búsqueda para facilitar la navegación de los usuarios."

1.1 Justificación del proyecto.

Un proyecto de tienda en línea de venta de cartas de Digimon sería justificado por el creciente interés en la nostalgia de los años 90 y el resurgimiento de la franquicia Digimon. La popularidad de los juegos, series de televisión y películas recientes de Digimon ha creado una demanda renovada de productos relacionados, como las cartas coleccionables. Una tienda en línea dedicada a la venta de cartas de Digimon proporcionaría a los aficionados un acceso conveniente a una amplia variedad de cartas para coleccionar, intercambiar y jugar. Además, aprovecharía el crecimiento del mercado de coleccionistas y aficionados a los juegos de cartas en línea, ofreciendo una plataforma para comprar productos auténticos y de alta calidad. La tienda podría diferenciarse ofreciendo cartas raras, ediciones limitadas y paquetes temáticos exclusivos para atraer a clientes y fomentar la lealtad a la marca. Además, podría incluir contenido adicional como guías de juego, noticias sobre eventos y torneos, y tutoriales para ayudar a los clientes a sacar el máximo provecho de su experiencia con las cartas de Digimon. La tienda en línea también sería una oportunidad para crear una comunidad en línea donde los aficionados puedan compartir sus colecciones, intercambiar estrategias y conectarse con otros fans de Digimon de todo el mundo. En resumen, un proyecto de tienda en línea de venta de cartas de Digimon capitalizaría el interés creciente en la franquicia, proporcionando a los aficionados un destino centralizado y accesible para satisfacer su pasión por coleccionar y jugar con las cartas de Digimon.

1.2 Caso de estudio.

Después de detectar un resurgimiento en el interés por la franquicia Digimon debido al lanzamiento de una nueva serie de televisión y varios juegos populares, una empresa de comercio electrónico especializada en productos de entretenimiento decide lanzar una tienda en línea dedicada exclusivamente a la venta de cartas de Digimon. Utilizando técnicas de marketing digital dirigidas a la comunidad de fans existente y aprovechando las redes sociales para generar anticipación y promoción antes del lanzamiento, la tienda en línea se abre con una amplia selección de cartas de Digimon, incluyendo raras y ediciones limitadas.

1.3 Estructura del documento.

Esta memoria se ha estructurado de la siguiente forma:

Introducción: En este segmento se detalla el proyecto y se explican en términos generales sus elementos, así como sus objetivos.

Análisis: Explicación de la estructuración del proyecto y sus capacidades, junto con los esquemas UML más destacados.

Diseño: Estratos de la arquitectura de la aplicación (interfaz, lógico y de almacenamiento en base de datos).

Ejecución: Las tecnologías aplicadas y la descripción de las herramientas utilizadas en la construcción.

Conclusiones: Reflexión desde una perspectiva personal acerca de la aplicación y su propósito.

Referencias: Listado de los recursos documentales consultados para la elaboración de este informe.

Apéndices: Datos adicionales para ampliar alguno de los aspectos tratados en los apartados previos.

Manual del usuario: Descripción sobre cómo emplear el gestor, con asistencia e ilustraciones para facilitar su comprensión.

1.4 Descripción general

La estructura de la tienda online se basa en una parte pública que es el front-end y una parte privada o administrativa que es el back-end.

El front-end es la parte de un sistema informático o sitio web que los usuarios ven y con la que interactúan directamente. Es la interfaz gráfica y funcionalidad que se muestra en el navegador web o en una aplicación, permitiendo a los usuarios realizar acciones y recibir información de manera visual.

El back-end es la parte de un sistema informático o sitio web que opera detrás de escena y no es visible para los usuarios. Se encarga de procesar datos, gestionar la lógica de la aplicación y comunicarse con la base de datos. En resumen, es el "cerebro" de la aplicación, maneja la lógica y la funcionalidad que permite que el front-end funcione correctamente.



1.5 Objetivos

El objetivo de crear una tienda en línea de cartas es satisfacer la demanda de los clientes que buscan adquirir cartas de manera conveniente y accesible a través de Internet, creando modificando y borrando los diferentes elementos que tenemos a disposición.

- Creación de una interfaz fácil, sencilla y amigable.
- Dar visibilidad al producto de manera directa.
- Dar soporte a diferentes dispositivos.
- Facilitar la edición y manipulación de la información.



1.6 Requisitos

1. Conexión a Internet: Se requiere acceso a una conexión de Internet estable para cargar y navegar por la página web de la tienda de cartas en línea.
2. Dispositivo compatible: Es necesario contar con un dispositivo compatible, como una computadora, tableta o smartphone, con un navegador web actualizado para acceder a la tienda en línea y visualizar su contenido.
3. Capacidad de navegación segura: Los usuarios deben tener la capacidad de navegar de manera segura en la página web de la tienda de cartas en línea, lo que implica contar con un navegador web que admita protocolos de seguridad como HTTPS y SSL para proteger la información personal y financiera.
4. Habilitar JavaScript: La funcionalidad de la tienda en línea puede depender del uso de JavaScript, por lo que los usuarios deben tener habilitado este lenguaje de programación en su navegador para una experiencia completa.
5. Método de pago: Para realizar compras en la tienda de cartas en línea, los usuarios necesitan contar con un método de pago válido, como tarjetas de crédito/débito, PayPal u otros sistemas de pago en línea aceptados por la tienda.

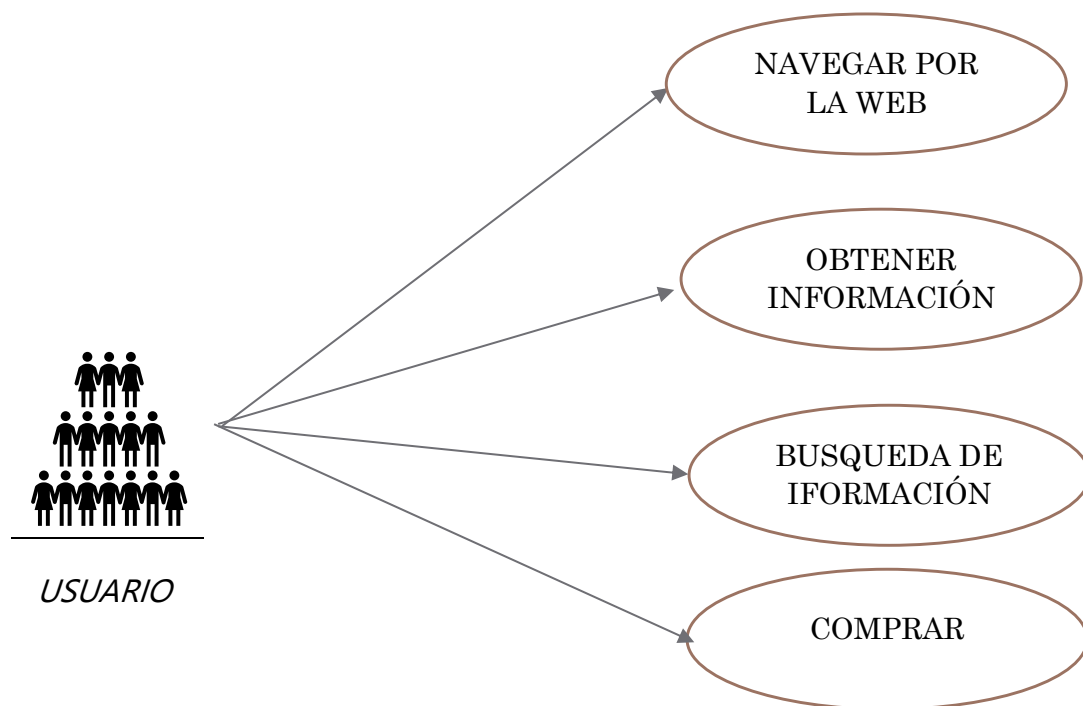
2. Análisis

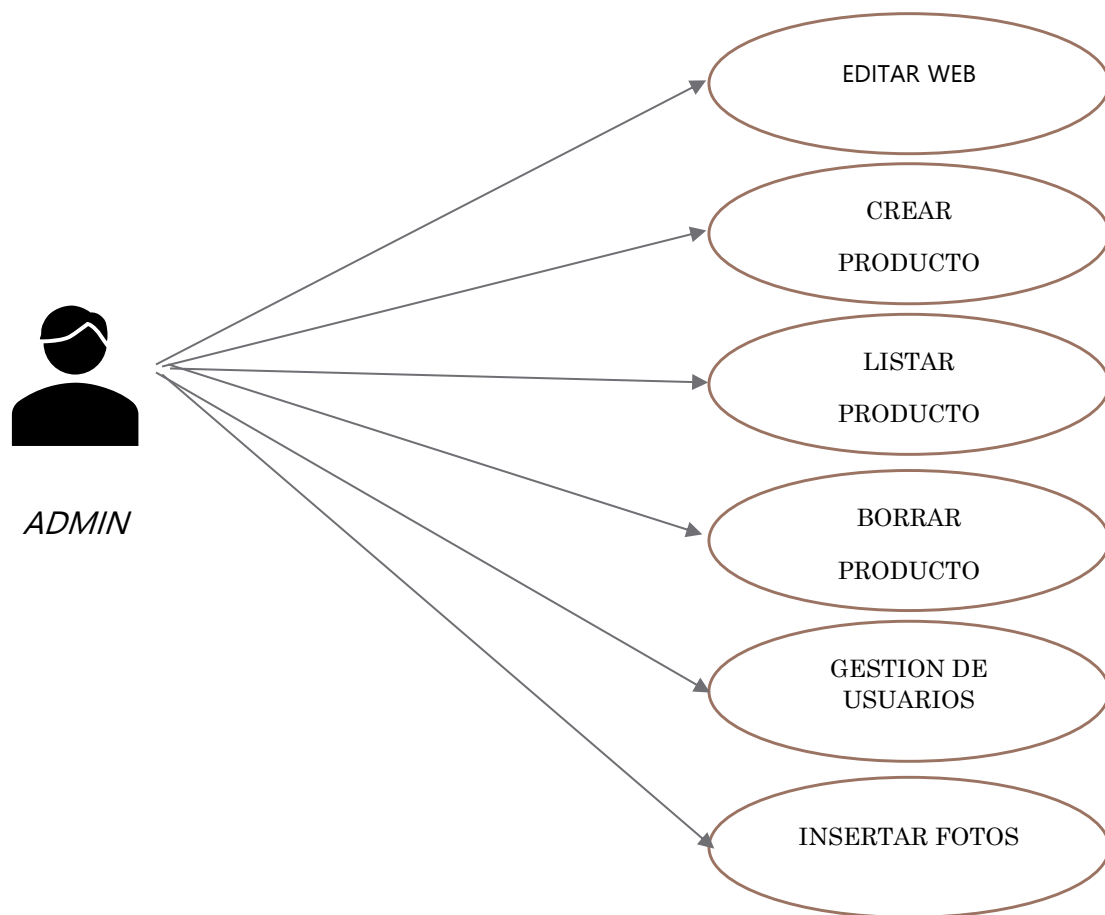
En este capítulo se comenta el análisis del proyecto con diagramas UML de los diferentes actores que interactúan con el proyecto y sus acciones con los elementos que veremos a continuación.

2.1 Casos de uso

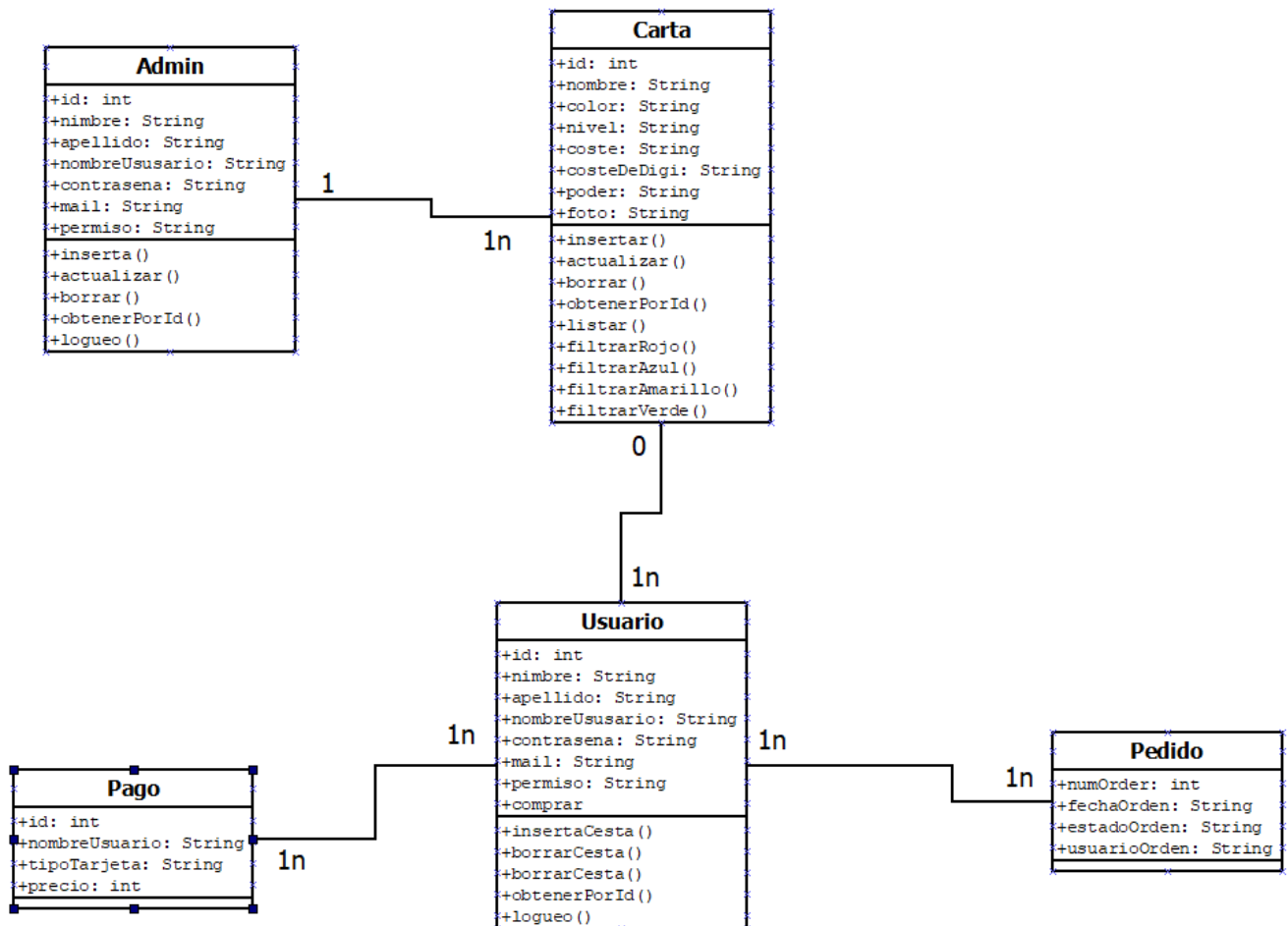
Un escenario de uso describe las acciones o actividades requeridas para ejecutar un determinado proceso. Los roles o entidades que intervienen en un escenario de uso se conocen como actores. En el ámbito de la ingeniería de software, un escenario de uso representa una serie de interacciones planificadas entre un sistema y sus actores, desencadenadas por un evento iniciado por un actor principal, y dirigidas al sistema en sí.

ACTOR ANÓNIMO NIVEL USUARIO



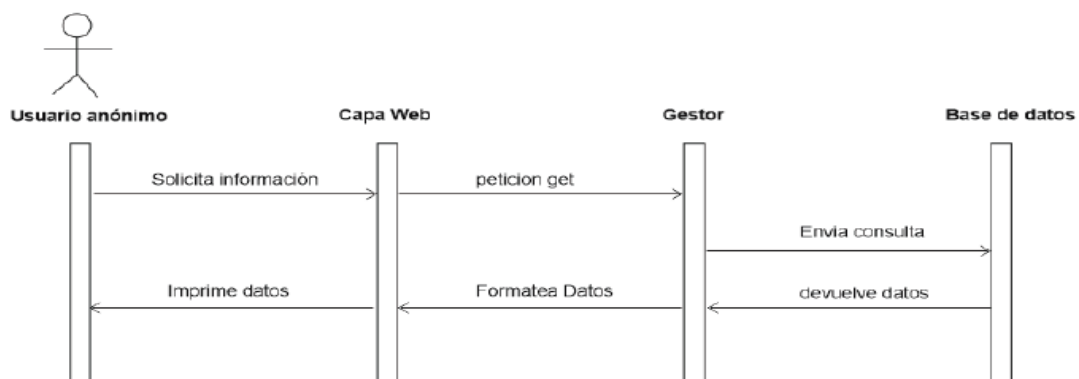


2.2 Diagrama de clases

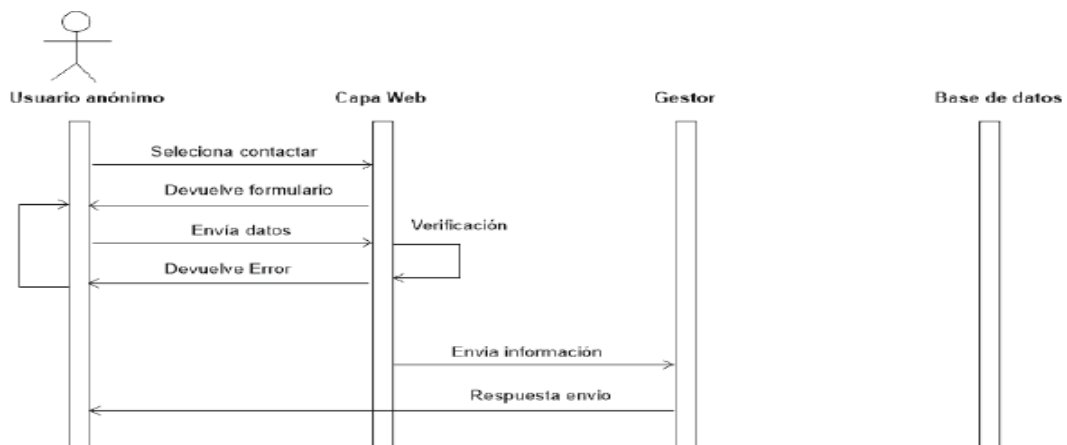


2.3 Diagrama de secuencia

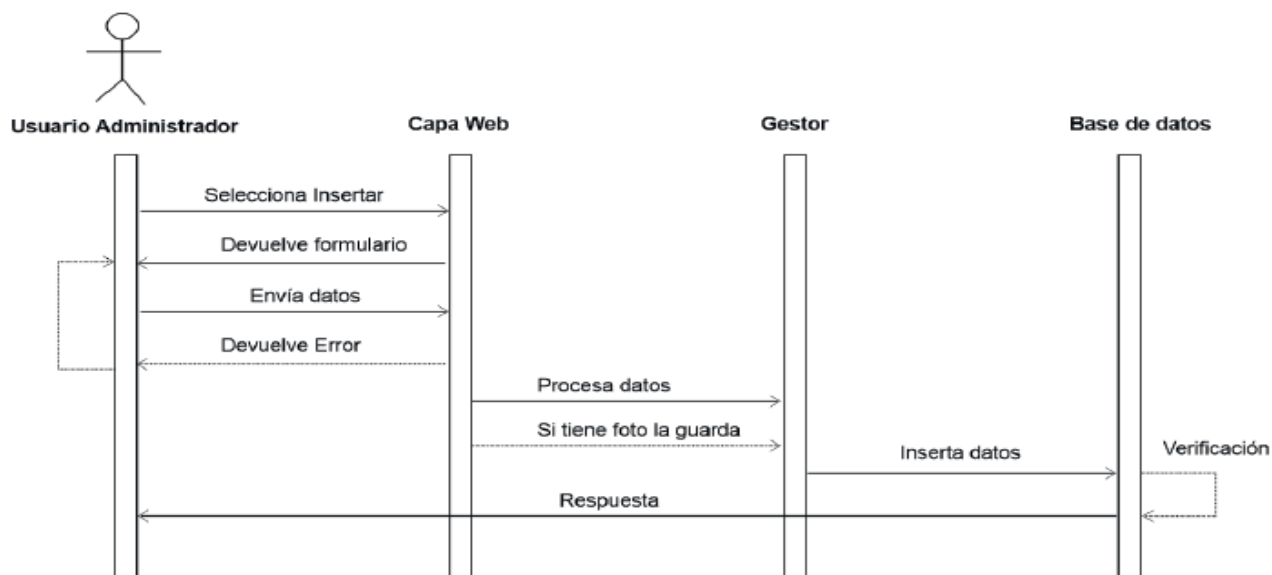
Un diagrama de secuencia es una herramienta de modelado en UML que muestra cómo los objetos interactúan en una secuencia de eventos a lo largo del tiempo. Representa la interacción entre diferentes partes de un sistema a través de mensajes enviados entre objetos. Los objetos se colocan en una línea de tiempo vertical y las flechas indican la dirección de los mensajes enviados entre ellos, mostrando el orden en que ocurren las interacciones. Es útil para visualizar el flujo de control entre objetos y entender el comportamiento dinámico de un sistema.



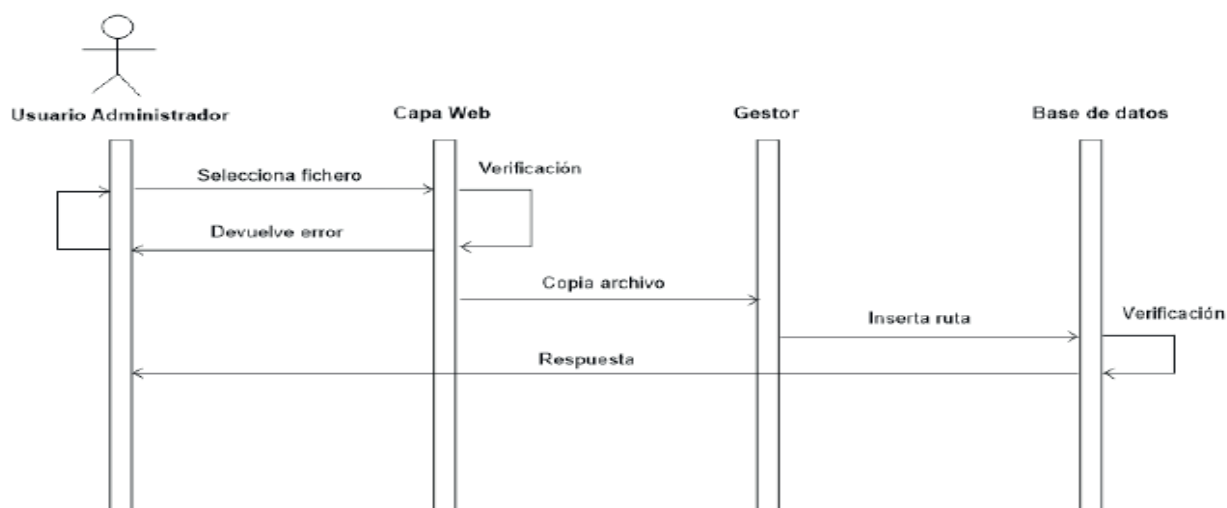
Visualizar contenido.



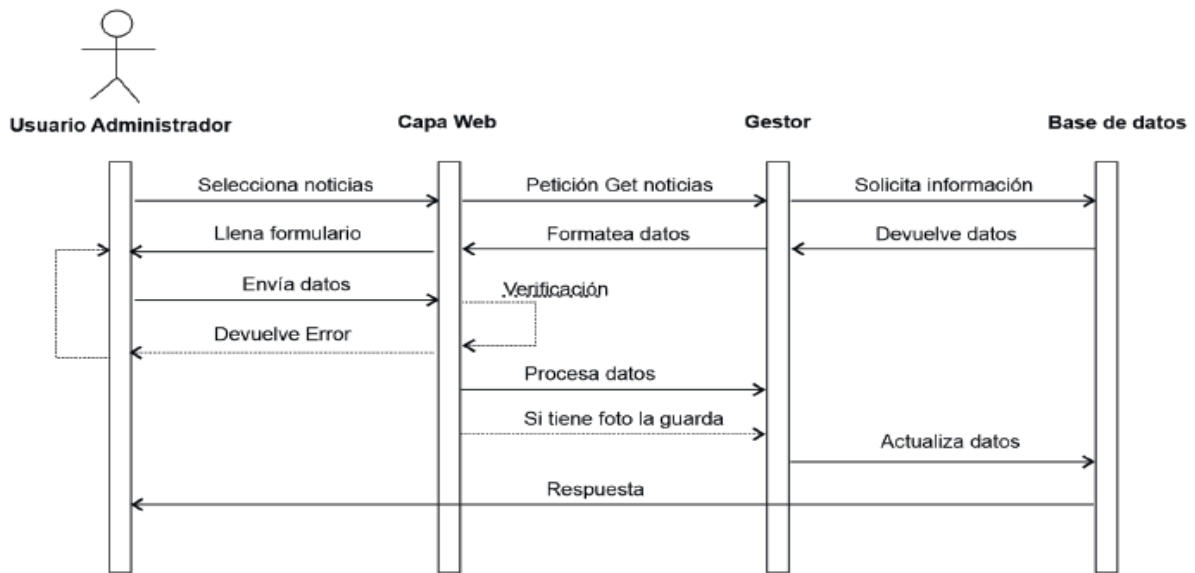
Solicitud de información.



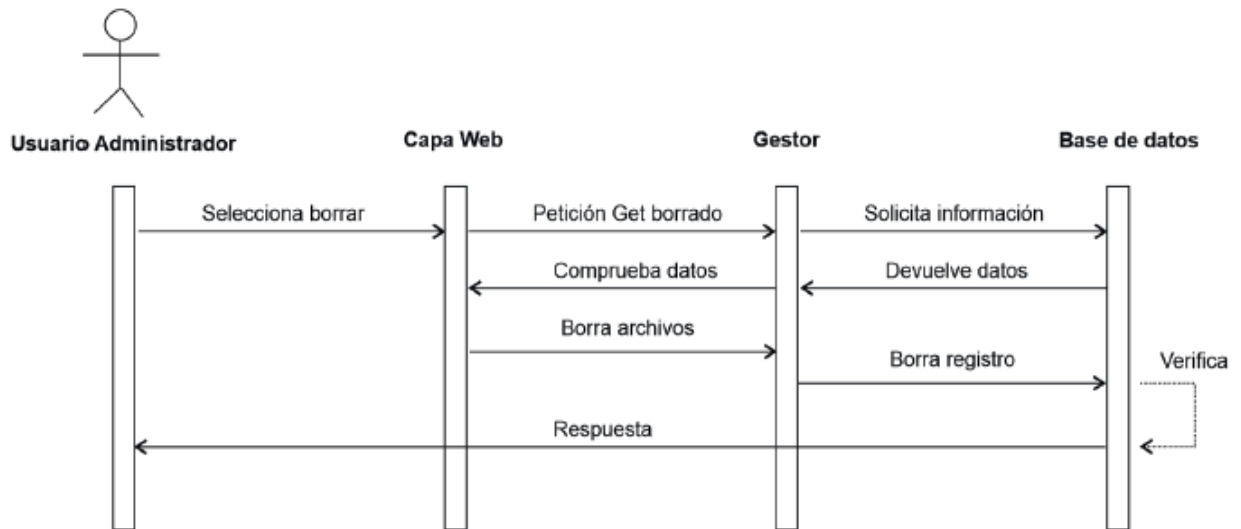
Insertar contenido.



Subir archivo foto.



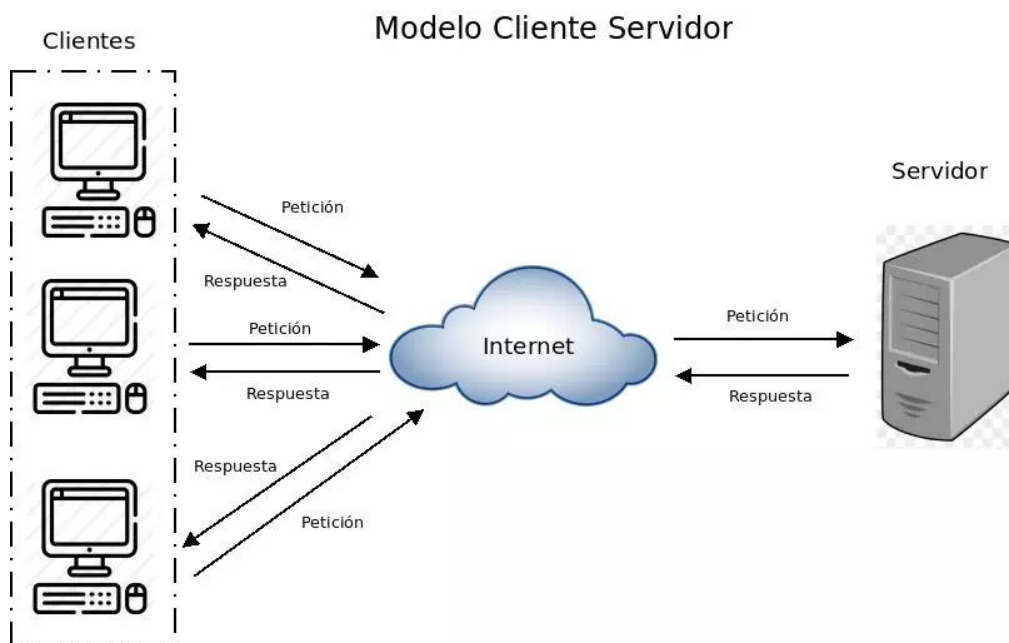
Actualizar datos.



Borrar datos.

3. Diseño

El portal utiliza un sistema donde hay una división clara de responsabilidades entre el servidor y los usuarios. El servidor provee los servicios y recursos solicitados por los usuarios, quienes son los clientes. Los usuarios, a través de sus terminales, hacen demandas al servidor, que las procesa y devuelve resultados o servicios según lo solicitado. Este modelo distribuido asegura una distribución eficiente de las tareas y recursos, permitiendo una interacción fluida entre los usuarios y el sistema.



La arquitectura de tres capas incluye un nivel inicial (Capa de presentación) que representa la parte visual y gráfica.

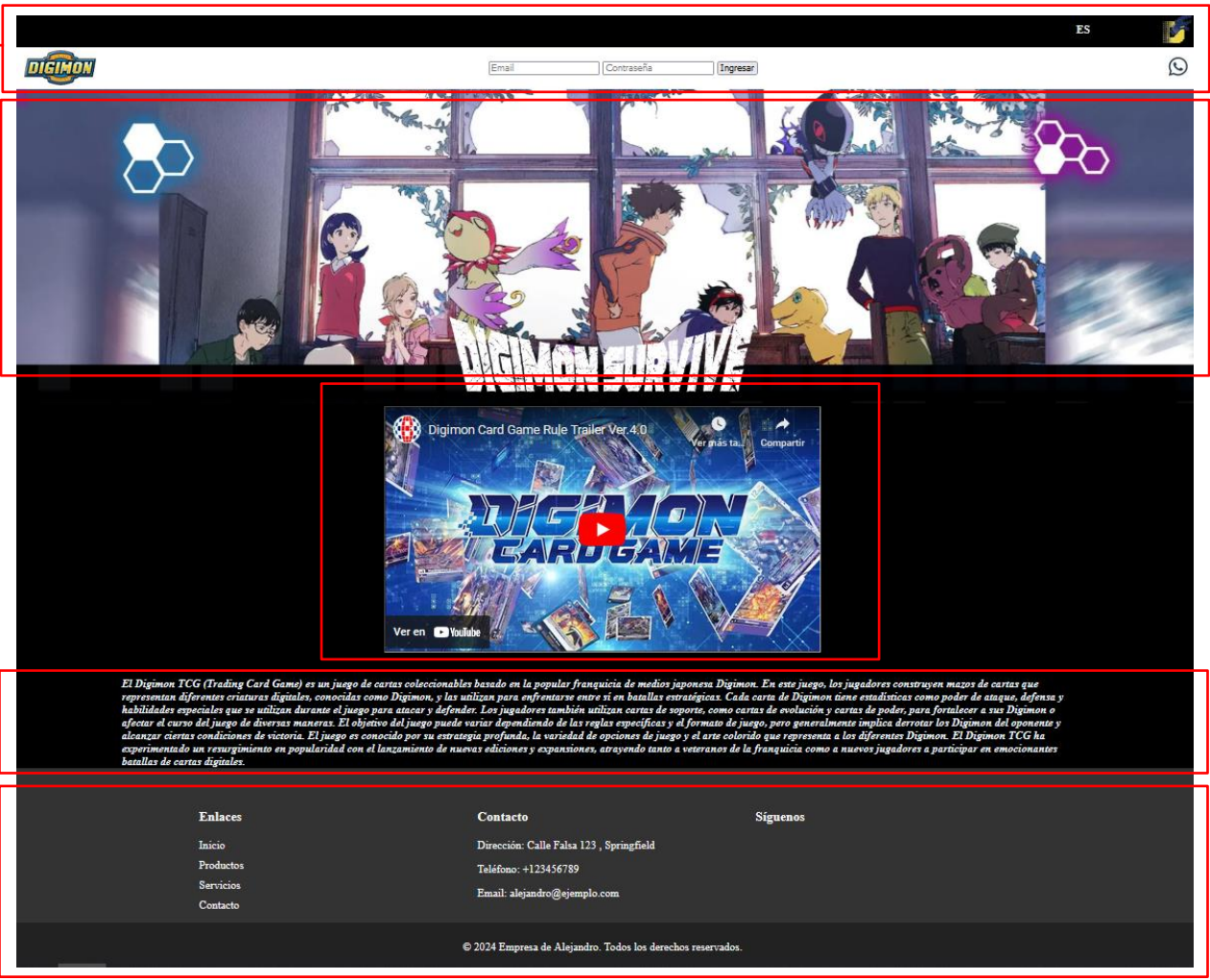
Un nivel intermedio (Capa lógica) que conecta al cliente, que solicita recursos a través de una interfaz de usuario o un navegador, con el servidor de datos. Este nivel intermedio, llamado software intermedio, facilita los recursos solicitados al cliente utilizando otro servidor.

El servidor de datos, la última capa (Capa de datos), suministra los datos necesarios para que el servidor de aplicaciones pueda procesar y proporcionar el servicio solicitado por el cliente.

3.1 Capa de presentación

La capa de presentación es la parte de un sistema o aplicación que se encarga de interactuar directamente con el usuario.

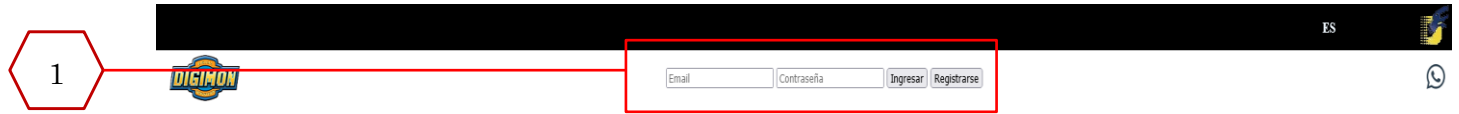
Mostrare capturas de pantalla de la interfaz de usuario y explicare las diferentes partes parte de la página según vaya navegando.



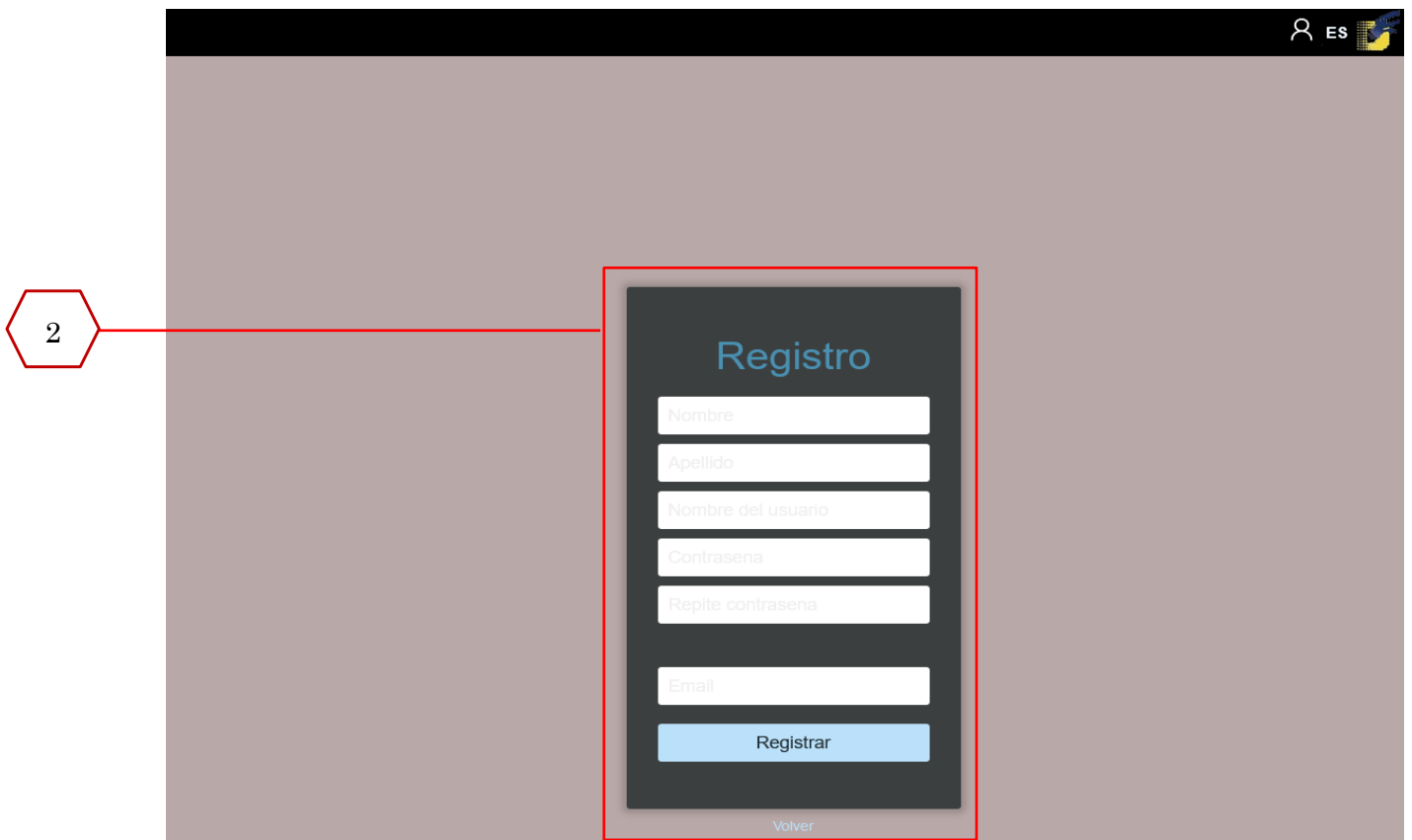
1. Navegador: En el que se mostrara el menú de navegación por el que usuario interactuara de forma rápida por la página.
2. Foto principal: Muestra una ilustración que va a caracterizar a la tienda online.
3. Video principal: Enlace de un video para explicar las mecánicas del juego de las cartas que se van a poder comprar.
4. Pie de página: Enlaces e información de la empresa o tienda física.

3.1.1 Login.

Lo primero que debería hacer el usuario sería insertar su Email y contraseña para tener permiso de ver las diferentes opciones de navegación.

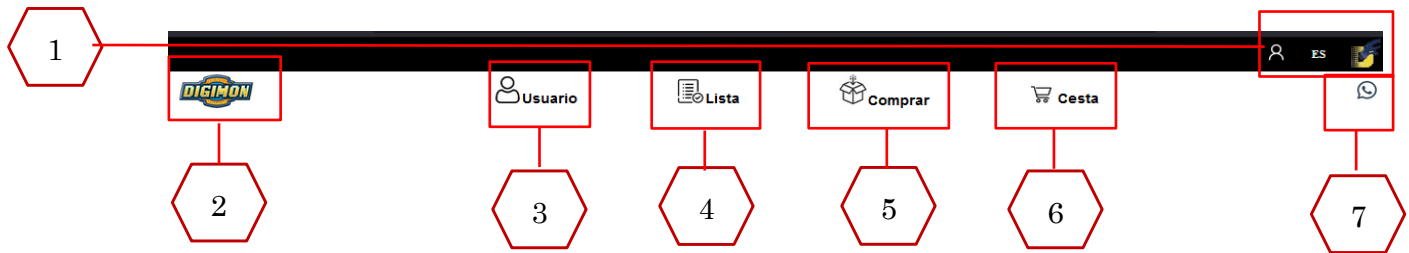


1. Registro (Login): El usuario tendrá la obligación de estar en la base de datos. Si no estuviera aún registrado, tendría que pulsar el botón de REGISTRARSE. Una vez pulsado pasaría a la siguiente web. Si el usuario tiene permiso de "Admin" se redirigira a otra web personal.



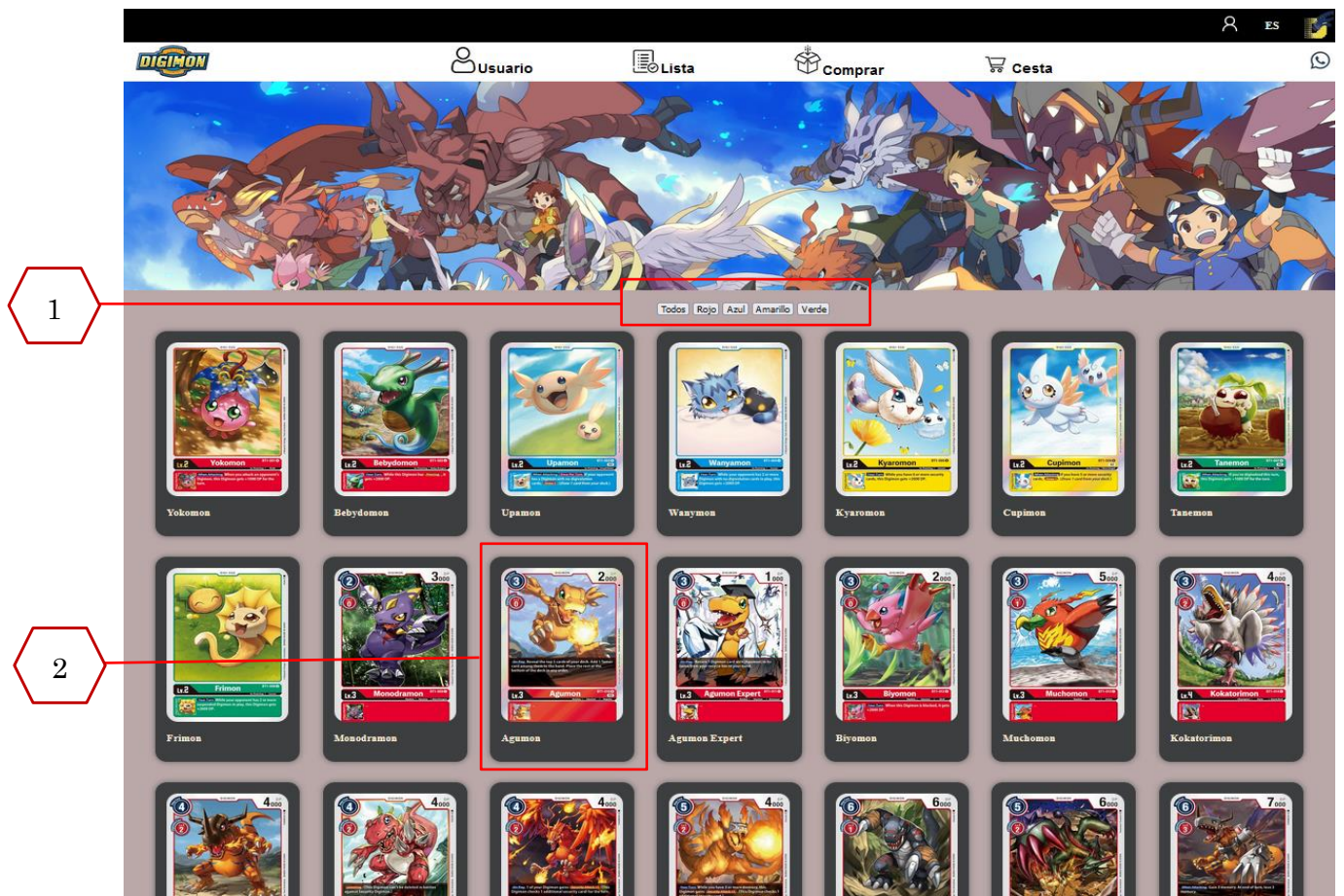
2. Nuevo Usuario: El usuario se registra en la base de datos y será redirigido a la página principal para ya ingresar mail y contraseña.

3.1.2 Navegador.



1. Índice de navegación: El usuario podrá cerrar sesión, cambiar el idioma e ir al índice de nuevo.
2. Icono principal: Dirige a la página principal.
3. Web usuario: El usuario podrá modificar sus datos personales.
4. Lista de cartas: Información de todas las cartas hasta el momento metidas en la base de datos.
5. Comprar: Lista de cartas en formato columna para ingresar la carta en la cesta del usuario.
6. Cesta: Una vez metidas las cartas que el usuario necesite, puedas comprarlas.
7. Social: Enlace a la aplicación de Telegram de la Tienda.

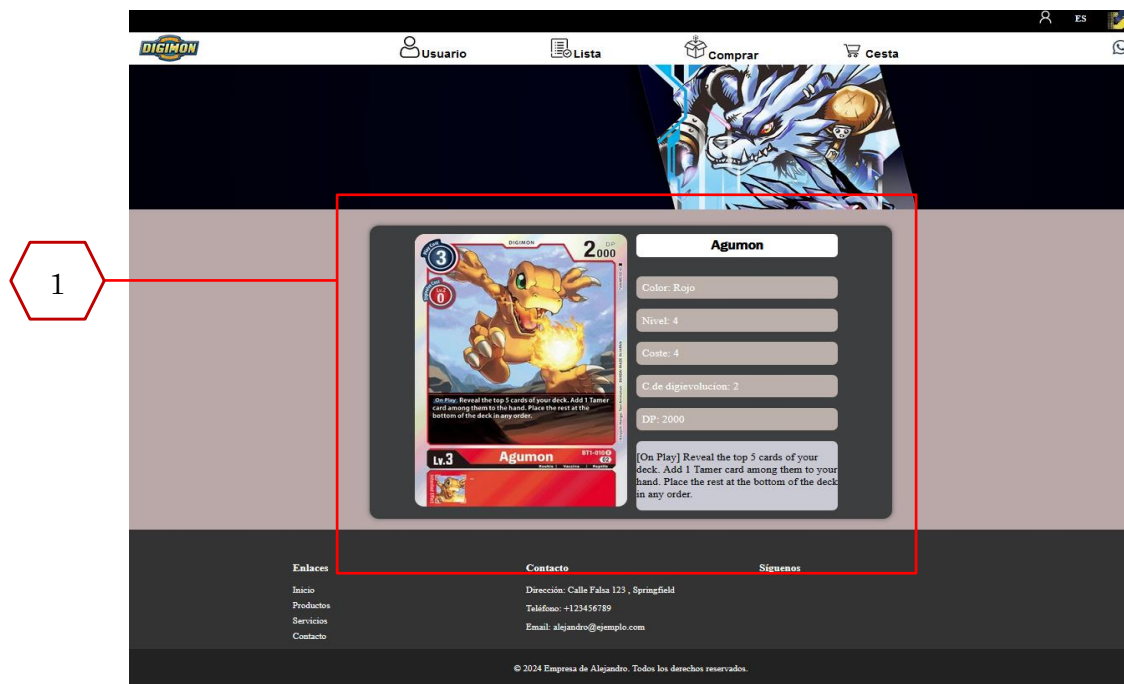
3.1.3 Sección Lista



1.Filtro: El usuario podrá filtrar entre las cartas según su atributo color.

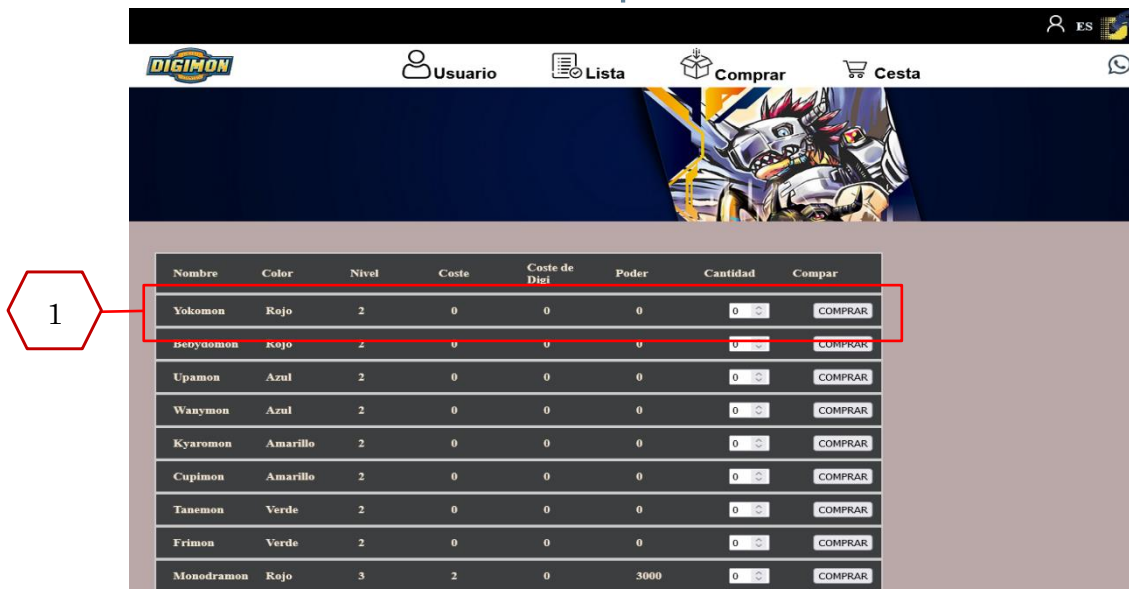
2.Carta: Imagen de la carta. Lleva al enlace con información específica de esta.

3.1.4 Carta



1.Carta: Muestra atributos y características de la carta de la lista y su descripción.

3.1.5 Sección Comprar

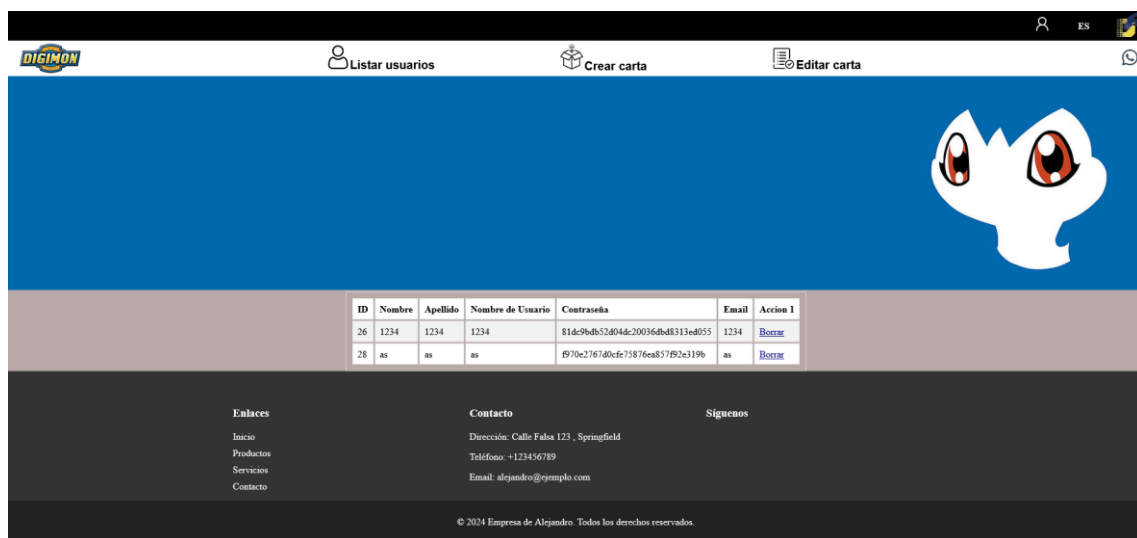


1.Compra de la carta: Se introduce de la carta que se desea, una cantidad de cartas para meter en la cesta.


3.1.6 Sección de Admin


Por la parte de back-end tenemos diferentes gestores de creación, edición y borrado tanto de los usuarios registrados como de las cartas listadas.


La barra de navegación cambiara según los permisos del usuario. Estas son las 3 interfaces que necesita el administrador.





La creación y edición se hacen en dos páginas diferentes en las que además de permitir meter información de la carta, también permite adjuntar la foto necesaria e insertarla tanto a la base de datos como a nuestro proyecto desde cualquier directorio.




[Listar usuarios](#)


[Crear carta](#)


[Editar carta](#)


ES

Nombre:

Color:

Nivel:


Coste:


Coste de digievolución:


Poder:


Foto:


No se ha seleccionado ningún archivo.





[Listar usuarios](#)


[Crear carta](#)


[Editar carta](#)

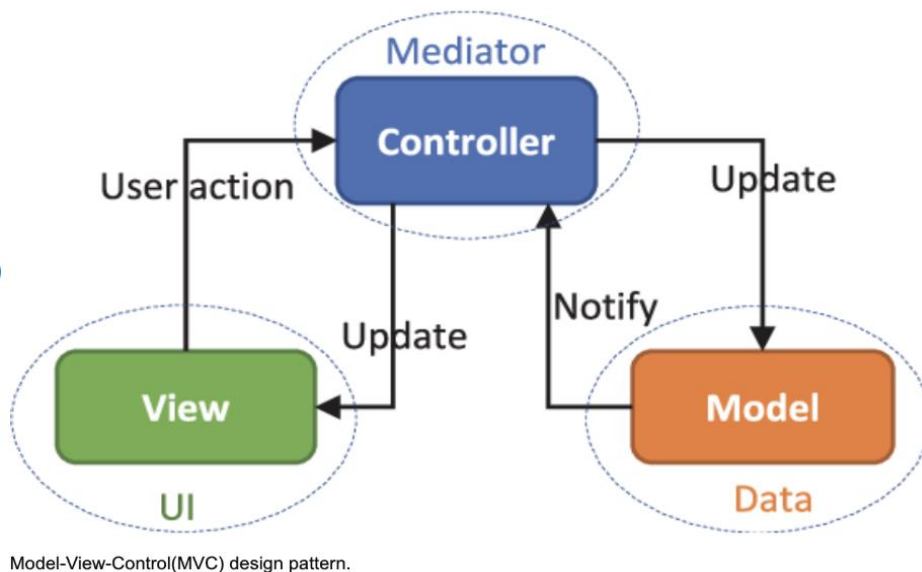

ES



ID	Nombre	Color	Nivel	Coste	Coste de Digi	Poder	Foto	Accion 1	Accion 2
1	Yokomon	Rojo	2	0	0	0	BT1-001.jpg	Editar	Borrar
2	Bebydemon	Rojo	2	0	0	0	BT1-002.jpg	Editar	Borrar
3	Upamon	Azul	2	0	0	0	BT1-003.jpg	Editar	Borrar
4	Wanymon	Azul	2	0	0	0	BT1-004.jpg	Editar	Borrar
5	Kyaromon	Amarillo	2	0	0	0	BT1-005.jpg	Editar	Borrar
6	Cupimon	Amarillo	2	0	0	0	BT1-006.jpg	Editar	Borrar
7	Tanemon	Verde	2	0	0	0	BT1-007.jpg	Editar	Borrar
8	Frimon	Verde	2	0	0	0	BT1-008.jpg	Editar	Borrar
9	Monodramon	Rojo	3	2	0	3000	BT1-009.jpg	Editar	Borrar
10	Agumon	Rojo	3	3	0	2000	BT1-010.jpg	Editar	Borrar
11	Agumon Expert	Rojo	3	3	0	1000	BT1-011.jpg	Editar	Borrar
12	Biyomon	Rojo	3	3	0	2000	BT1-012.jpg	Editar	Borrar

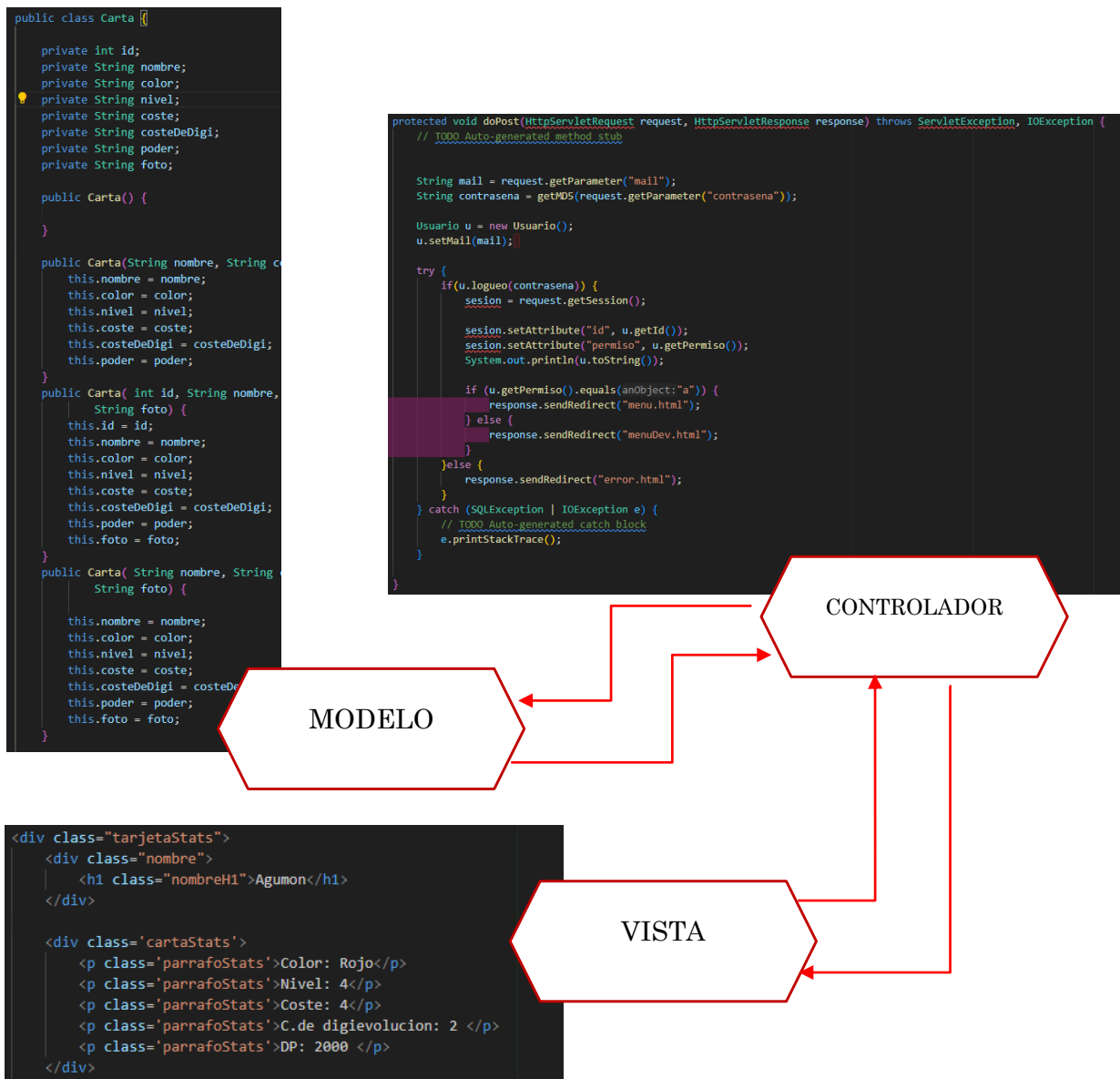
3.2 Capa lógica o controlador

La capa lógica, dentro del contexto de desarrollo de software, es una parte fundamental de la arquitectura de un sistema. Esta capa se encarga de definir los comportamientos, reglas y procesos que gobiernan la interacción entre la interfaz de usuario y los datos almacenados en la capa de persistencia (como una base de datos)



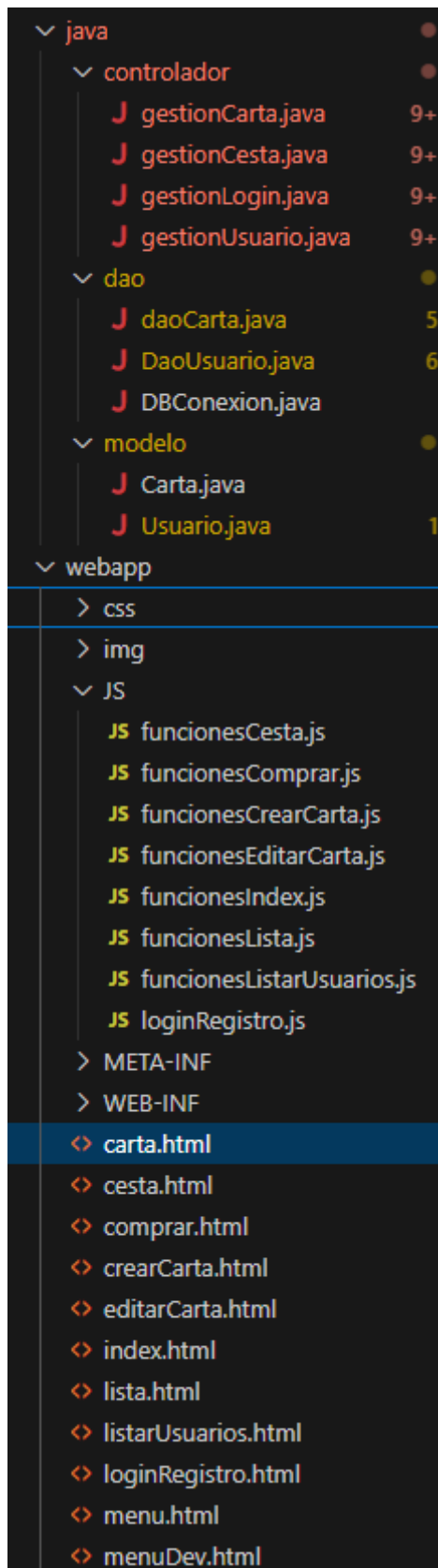
En el contexto de un patrón de diseño como Modelo-Vista-Controlador (MVC), la capa lógica se separa de la capa de presentación (interfaz de usuario) y la capa de datos (persistencia). En el MVC, las clases y métodos dentro de la capa lógica representan la lógica de negocio y la manipulación de datos independiente de cómo se muestran o almacenan los datos.

Ejemplos.



Con estos tres un ejemplo de código se ve como interactúan entre las tres partes el MVC.

Esto es un ejemplo que desgranaremos más adelante.



Archivos de la parte
CONTROLADOR del
proyecto



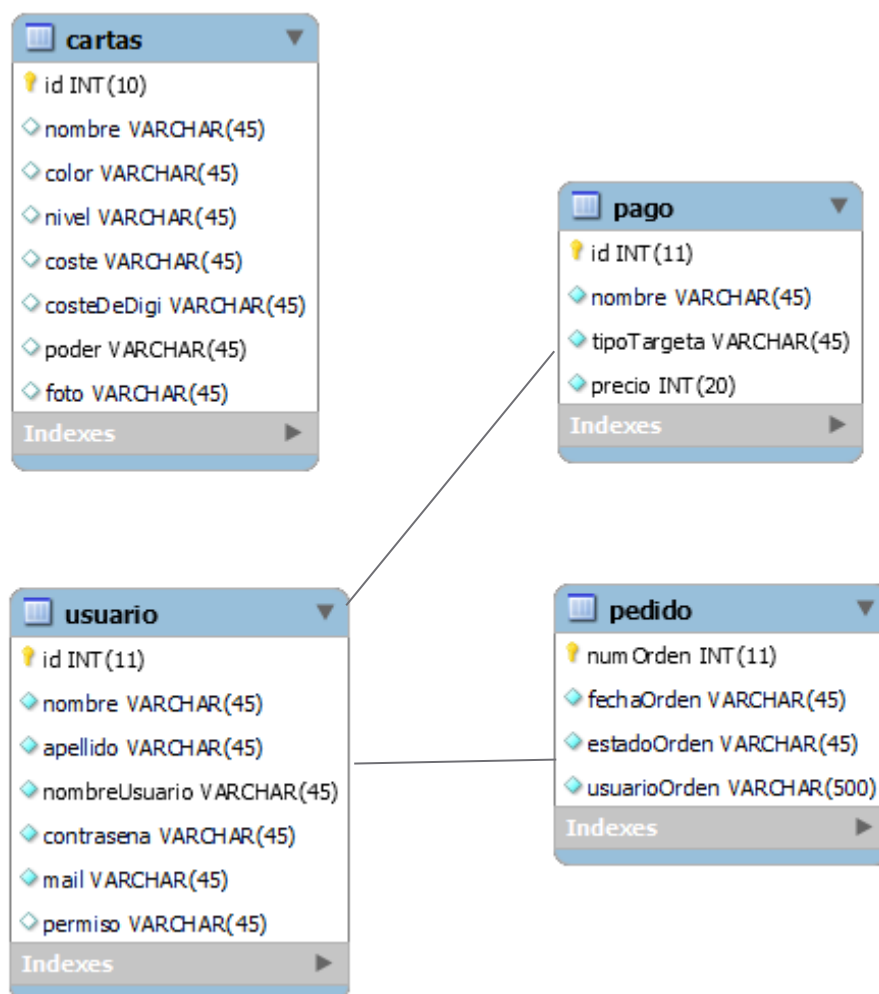
Archivos de la parte
MODELO del
proyecto



Archivos de la parte
VISTA del proyecto

3.3 Capa de datos

La capa de datos es la parte de la arquitectura de software donde se almacenan y manipulan los datos de la aplicación, separada de la lógica de negocio y la interfaz de usuario, asegurando la integridad y persistencia de la información.



4. Implementación

4.1 Tecnologías.

4.1.1 JAVA.

La tecnología de Java es una plataforma de desarrollo versátil y robusta que permite crear una amplia gama de aplicaciones, desde aplicaciones de escritorio hasta sistemas empresariales y aplicaciones web. Utiliza un enfoque de "write once, run anywhere" (escribe una vez, ejecuta en cualquier lugar), lo que significa que el código Java puede ejecutarse en múltiples plataformas sin necesidad de modificaciones. Java se ha destacado por su seguridad, portabilidad y rendimiento, lo que lo convierte en una opción popular para desarrolladores en una variedad de industrias.

Además, la comunidad de Java es extensa y activa, lo que garantiza un amplio soporte y una gran cantidad de recursos disponibles para los desarrolladores.

```
public class HelloWorld {  
    public static void main(String[] args) {  
        System.out.println("¡Hola, mundo!");  
    }  
}
```



4.1.2 HTML.

HTML, o HyperText Markup Language, es el lenguaje estándar utilizado para crear y diseñar páginas web. Utiliza una estructura de marcado que define la estructura y el contenido de una página web mediante etiquetas. Estas etiquetas permiten al desarrollador especificar diferentes elementos como encabezados, párrafos, enlaces, imágenes y más. HTML es un lenguaje de marcado simple y fácil de aprender, lo que lo convierte en la base fundamental para la creación de sitios web. Junto con CSS (Cascading Style Sheets) y JavaScript, HTML forma la tríada básica de tecnologías utilizadas en el desarrollo web moderno.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Ejemplo de HTML Básico</title>
</head>
<body>

  <h1>¡Hola, mundo!</h1>

  <p>Este es un ejemplo básico de una página web utilizando HTML.</p>

</body>
</html>
```



4.1.3 CSS

CSS, o Cascading Style Sheets, es un lenguaje utilizado para describir la presentación de un documento HTML (o XML). Permite definir el aspecto visual de los elementos de una página web, como el color, la tipografía, el diseño y otros aspectos relacionados con la apariencia. CSS utiliza reglas que se aplican a elementos específicos o conjuntos de elementos en un documento HTML, lo que proporciona un control preciso sobre cómo se visualiza el contenido en diferentes dispositivos y tamaños de pantalla. Al separar el contenido de su presentación, CSS promueve la creación de sitios web más accesibles, mantenibles y consistentes en términos de diseño. Con la evolución de CSS y sus capacidades, los desarrolladores pueden crear diseños más complejos y adaptativos, lo que contribuye a una experiencia de usuario más rica y atractiva en la web.

```
<style>
  body {
    font-family: Arial, sans-serif;
    background-color: #f0f0f0;
    color: #333;
    margin: 0;
    padding: 20px;
  }

  h1 {
    color: #0066cc;
  }

  p {
    font-size: 18px;
    line-height: 1.5;
  }
</style>
```



4.1.4 JavaScript

JavaScript es un lenguaje de programación de alto nivel, interpretado por el navegador web, que se utiliza para agregar interactividad y dinamismo a las páginas web. A diferencia de HTML y CSS, que se centran en la estructura y el estilo, respectivamente, JavaScript se enfoca en la funcionalidad y el comportamiento de una página web. Permite a los desarrolladores crear aplicaciones web complejas, manipular el contenido de la página en tiempo real, responder a acciones del usuario, realizar solicitudes de red asincrónicas y mucho más. JavaScript es un lenguaje versátil que se ha expandido más allá del navegador web, con frameworks y librerías que permiten el desarrollo tanto de aplicaciones web como de aplicaciones móviles y de escritorio. Su capacidad para interactuar con HTML y CSS lo convierte en una herramienta poderosa para el desarrollo web moderno.

```
<body>
  <script>
    alert("¡Hola, mundo!");
  </script>
</body>
```



4.1.5 SQL

SQL, o Structured Query Language, es un lenguaje de programación diseñado para gestionar y manipular bases de datos relacionales. Proporciona un conjunto de comandos que permiten realizar diversas operaciones, como la creación, modificación y eliminación de tablas y datos, así como la consulta y la actualización de información almacenada en la base de datos. SQL se utiliza en una amplia gama de aplicaciones y entornos, desde sistemas de gestión de bases de datos (DBMS) como MySQL, PostgreSQL y Oracle, hasta aplicaciones web y empresariales. Su sintaxis es clara y fácil de aprender, lo que lo hace accesible tanto para desarrolladores novatos como experimentados. Además, SQL es un estándar ampliamente aceptado en la industria, lo que garantiza la portabilidad y la interoperabilidad entre diferentes sistemas y tecnologías de bases de datos.

```
1 • create database clinica;
2 • use clinica;
3 • create table cliente(
4     cod_cliente char(5) not null,
5     nombre varchar(30) not null,
6     apellidos varchar(30) not null,
7     direccion varchar(30),
8     telefono varchar(15),
9     primary key(cod_cliente)
10 );
11
12 • INSERT INTO CLIENTE(COD_CLIENTE,NOMBRE,APELLIDOS,TELEFONO)
13 VALUES ('C_1','Luis','Pérez Pérez',33344455);
14 • INSERT INTO CLIENTE(COD_CLIENTE,NOMBRE,APELLIDOS,DIRECCION)
15 VALUES ('C_2','Juan','García López','Del Valle');
16 • INSERT INTO CLIENTE(COD_CLIENTE,NOMBRE,APELLIDOS,DIRECCION,TELEFONO)
17 VALUES ('C_3','Rosa','Sánchez Gil','Paseo Recoletos',8888888);
18 • INSERT INTO CLIENTE(COD_CLIENTE,NOMBRE,APELLIDOS,DIRECCION,TELEFONO)
19 VALUES ('C_4','Isabel','Rodríguez Pla','De la Luz',5555555);
20
```



4.2 Tecnologías.

4.2.1 Tecnologías visuales.

Para este proyecto he utilizado la los exploradores web que normalmente se utilizan. Para el testeo de la tecnología JAVA he utilizado Firefox y para el testeo y desarrollo de la tecnología JAVASCRIPT HTML y CSS he utilizado el explorador Chrome.

Esto me ha facilitado el trabajo por ejemplo en el vaciado de cache en diferentes lapsus de tiempo.



4.2.2 Hardware.

El proyecto está corriendo en un ordenador personal CPU con los siguientes componentes:

- Microprocessor: AMD Ryzen 7 1800X Eight-Core Processor 3.60 GHz.
- RAM: 16,0 GB DDR4.
- Tarjeta gráfica: NVIDIA 2070 Super.
- Memoria: SSD 1 T.
- Placa base: MSI AB350-Gaming 3.
- SO: Microsoft Windows 10 Pro

4.3 Tecnologías.

He realizado el proyecto utilizando estas herramientas principales:

- XAMPP: para emular un servidor local.
- Photoshop: Para la edición de imágenes.
- Eclipse-Workspace y VSC: Como IDE para JAVA, JAVASCRIPT, HTML Y CSS.
- Exploradores como Firefox y Chrome: Para realizar las pruebas del contenido.
- MySQL: Como interfaz de base de datos.

4.3.1 Ejemplos de conexión.

El proceso de conexión que se utiliza para comunicarnos con la base de datos hacia el Front-end.

```
public class DBConexion {  
    //Conexion de Otero//  
  
    public static final String JDBC_URL = "jdbc:mysql://localhost:3306/digimon";  
    public static Connection instance = null;  
  
    public static Connection getConexion() throws SQLException {  
        if(instance == null) {  
            //opcional  
            Properties props = new Properties();  
            props.put("user", "root");  
            props.put("password", "");  
            props.put("charset", "UTF-8");  
            instance = DriverManager.getConnection(JDBC_URL, props);  
        }  
        return instance;  
    }  
}
```

Cada vez que llamemos a la función del ejemplo nos conectara con la base de datos.

4.3.2 Ejemplos de Carta

Con ejemplo "carta" voy a explicar un CRUD completo del proyecto.

```
public class Carta {  
    private int id;  
    private String nombre;  
    private String color;  
    private String nivel;  
    private String coste;  
    private String costeDeDigi;  
    private String poder;  
    private String foto;  
  
    public Carta() {  
    }  
  
    public Carta(String nombre, String color, String nivel, String coste, String costeDeDigi, String poder) {  
        this.nombre = nombre;  
        this.color = color;  
        this.nivel = nivel;  
        this.coste = coste;  
        this.costeDeDigi = costeDeDigi;  
        this.poder = poder;  
    }  
  
    public Carta(int id, String nombre, String color, String nivel, String coste, String costeDeDigi, String poder,  
        String foto) {  
        this.id = id;  
        this.nombre = nombre;  
        this.color = color;  
        this.nivel = nivel;  
        this.coste = coste;  
        this.costeDeDigi = costeDeDigi;  
        this.poder = poder;  
        this.foto = foto;  
    }  
  
    public Carta(String nombre, String color, String nivel, String coste, String costeDeDigi, String poder,  
        String foto) {  
        this.nombre = nombre;  
        this.color = color;  
        this.nivel = nivel;  
        this.coste = coste;  
        this.costeDeDigi = costeDeDigi;  
        this.poder = poder;  
        this.foto = foto;  
    }  
}
```

Ejemplo del objeto
Carta.java

CODIGO DE CREAR CARTA

Como usuario administrador, en la parte de crear carta podremos rellenar el formulario e insertar nuestra carta nueva en la base de datos.

Nombre:

Color:

Nivel:

Coste:

Coste de digievolución:

Poder:

Foto:

Examinar... No

Guardar

```
<form class="formulario" name="altaCarta" action="gestionCarta" method="post" id="Carta" enctype="multipart/form-data" >  
    <ul>  
        <input type="hidden" id="id" name="id">  
        <li><label>Nombre: </label> <input type="text" name="nombre" id="nombre"></li>  
        <li><label>Color: </label> <input type="text" name="color" id="color"></li>  
        <li><label>Nivel: </label> <input type="text" name="nivel" id="nivel"></li>  
        <li><label>Coste: </label> <input type="text" name="coste" id="coste"></li>  
        <li><label>Coste de digievolución: </label> <input type="text" name="costeDeDigi" id="costeDeDigi"></li>  
        <li><label>Poder: </label> <input type="text" name="poder" id="poder"></li>  
        <li><label>Foto: </label> <input type="file" name="foto" id="foto"></li>  
        <li><input type="submit" value="Guardar" onclick="validarFormulario()"></li>  
    </ul>  
</form>
```

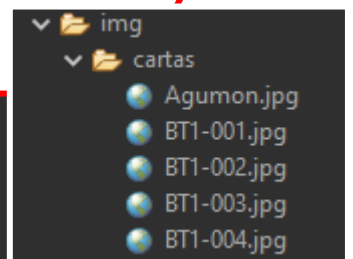
Los datos del formulario pasan gracias a JAVASCRIPT al Back-End.

```
function validarFormulario(){  
  
    let nombre = document.getElementById('nombre').value;  
    let color = document.getElementById('color').value;  
    let nivel = document.getElementById('nivel').value;  
    let coste = document.getElementById('coste').value;  
    let costeDeDigi = document.getElementById('costeDeDigi').value;  
    let poder = document.getElementById('poder').value;  
    let foto = document.getElementById('foto').value;  
  
    let ok = true;  
    if(nombre == ""){  
        ok = false;  
        document.getElementById('nombre').style.background = "red";  
    }  
  
    if(color == ""){  
        ok = false;  
        document.getElementById('color').style.background = "red";  
    }  
  
    if(nivel == ""){  
        ok = false;  
        document.getElementById('nivel').style.background = "red";  
    }  
    if(coste == ""){  
        ok = false;  
        document.getElementById('coste').style.background = "red";  
    }  
    if(costeDeDigi == ""){  
        ok = false;  
        document.getElementById('costeDeDigi').style.background = "red";  
    }  
    if(poder == ""){  
        ok = false;  
        document.getElementById('poder').style.background = "red";  
    }  
    if(foto == ""){  
        ok = false;  
        document.getElementById('foto').style.background = "red";  
    }  
    if(ok){  
        // ...  
    }  
}
```

En el Back-End recogemos los datos mediante el método doPost de gestionCarta.java

```
protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
    //TODO Auto-generated method stub  
  
    String nombre = request.getParameter("nombre");  
    String color = request.getParameter("color");  
    String nivel = request.getParameter("nivel");  
    String coste = request.getParameter("coste");  
    String costeDeDigi = request.getParameter("costeDeDigi");  
    String poder = request.getParameter("poder");  
    String id = request.getParameter("id");  
  
    Part part = request.getPart("foto");  
    Path path = Paths.get(part.getSubmittedFileName());  
    String fileName = path.getFileName().toString();  
  
    InputStream input = part.getInputStream();  
    File file = new File(uploads, fileName);  
  
    try {  
        Files.copy(input, file.toPath());  
    } catch (Exception e) {  
        // TODO: handle exception  
        System.out.println("Error en la copia del archivo");  
        PrintWriter error = response.getWriter();  
        error.print("<h4> Se ha producido un error</h4>");  
    }  
  
    Carta c = new Carta(nombre, color, nivel, coste, costeDeDigi, poder, fileName);  
    try {  
        if(id == "") {  
            c.insertar();  
            response.sendRedirect("crarCarta.html");  
        } else {  
            int idInt = Integer.parseInt(id);  
            c.setId(idInt);  
            c.actualizar();  
            response.sendRedirect("lista.html");  
        }  
    } catch (SQLException e) {  
        // TODO Auto-generated catch block  
        e.printStackTrace();  
    }  
    //response.sendRedirect("listado.html");  
    System.out.println(c.toString());  
}
```

En el método doPost se ha añadido el código para que al ingresar un archivo foto, se copie automáticamente a la carpeta: img/cartas



Mediante la función inserta metemos los parámetros del objeto Carta a la base de datos cartas.sql

El código de los métodos de insertado y actualizado se encuentran en daoCarta.java que a su vez es llamada en el objeto Carta.java que a su vez es llamado por el servlet gestionCarta.java

```
public void insertar() throws SQLException {  
    daoCarta dao = new daoCarta();  
    dao.insertar(this);  
}
```

```
public void actualizar() throws SQLException {  
    daoCarta dao = new daoCarta();  
    dao.actualizar(this);  
}
```

Carta.java

```
public void insertar(Carta n) throws SQLException {  
    PreparedStatement ps = con.prepareStatement("INSERT INTO cartas (nombre, color ,nivel , coste, costeDeDigi, poder, foto) VALUES (?,? ,?,? ,?,?)");  
    ps.setString(1, n.getNombre());  
    ps.setString(2, n.getColor());  
    ps.setString(3, n.getNivel());  
    ps.setString(4, n.getCoste());  
    ps.setString(5, n.getCosteDeDigi());  
    ps.setString(6, n.getPoder());  
    ps.setString(7, n.getFoto());  
  
    int filas = ps.executeUpdate();  
    ps.close();  
}
```

daoCarta.java

BASE DE
DATOS

El método insertar hace una conexión con la base de datos y le envía la Query a la base de datos para insertar los parámetros obtenidos desde el form(HTML)>JavaScript>doPost(servlet)>Objeto>DAO> Base de datos.

CODIGO DE LISTAR CARTA

El orden para listar sería exactamente el contrario salvo por alguna diferencia.

En este caso sería:

Base de datos>DAO>Objeto>doGet(servlet)>JavaScript>pintar formulario(HTML).

```
public ArrayList <Carta> listar() throws SQLException {  
  
    String sql = "SELECT id,nombre,color,nivel,coste,costeDeDigi,poder,foto FROM cartas";  
    PreparedStatement ps = con.prepareStatement(sql);  
    ResultSet rs = ps.executeQuery();  
  
    ArrayList <Carta> ls = null;  
  
    while(rs.next()) {  
        if(ls == null) {  
            ls = new ArrayList<>();  
        }  
        ls.add(new Carta(rs.getInt(1),rs.getString(2),rs.getString(3),rs.getString(4),  
            rs.getString(5),rs.getString(6),rs.getString(7),rs.getString(8)));  
    }  
    return ls;  
}  
  
public String listarJson() throws SQLException {  
    String json = "";  
    Gson gson = new Gson();  
    json = gson.toJson(this.listar());  
    return json;  
}
```

Función para crear un array del
con todos los usuarios en la base
de datos y crear un JSON

```
if (opParam != null && !opParam.isEmpty()) {  
    int opcion = Integer.parseInt(opParam);  
  
    try {  
        switch (opcion) {  
            case 1:  
                daoCarta cartas1 = new daoCarta();  
                out.print(cartas1.listarJson());  
                break;  
        }  
    }  
}
```

Esta opción será llamada desde
JavaScript (opcion = 1)

En JavaScript hacemos la llamada de la opción 1 en este caso y pintamos la tabla.

```
function llamada(){  
    fetch('gestionCarta?op=1')  
    .then(response => response.json())  
    .then(data => pintarTabla(data))  
}
```

```
function pintarTabla(datos) {  
    let html = "<section class='section2'>";  
  
    for (let i = 0; i < datos.length; i++) {  
        html += "<div class='carta'>";  
        html += "<a href = 'carta.html'><img class='Imagen' src='img/cartas/' + datos[i].foto + '' alt='Imagen de la carta'></a>";  
        html += "<div class='texto'><h2 class='titulo'> + datos[i].nombre + "</h2></div>";  
  
        html += "</div>";  
    }  
  
    html += "</section>";  
    document.getElementById("listadoCartas").innerHTML = html;  
}  
  
window.onload = function() {  
    llamada();  
}
```

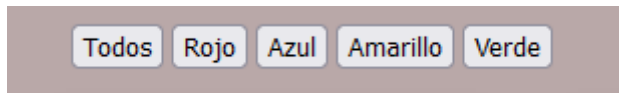
Llamada al servlet mediante un fetch e incorporamos código HTML con los parámetros que extraíamos en base de datos.

```
<section>  
    <div id="listadoCartas"></div>  
</section>
```

Front-End



CODIGO DE FILTRADO DE CARTA



Otros métodos de listado serían mediante el filtrado según parámetros del objeto Carta. Un ejemplo de código de filtrado de "Color: ROJO".

```
public ArrayList <Carta> filtroRojo() throws SQLException {  
    String sql = "SELECT id,nombre,color,nivel,coste,costeDeDigi,poder,foto FROM cartas WHERE color='rojo'";  
    PreparedStatement ps = con.prepareStatement(sql);  
    ResultSet rs = ps.executeQuery();  
  
    ArrayList <Carta> ls = null;  
  
    while(rs.next()) {  
        if(ls == null) {  
            ls = new ArrayList<>();  
        }  
        ls.add(new Carta(rs.getInt(1),rs.getString(2),rs.getString(3),rs.getString(4),  
            rs.getString(5),rs.getString(6),rs.getString(7),rs.getString(8)));  
    }  
    return ls;  
}  
  
public String listarRojos() throws SQLException {  
    String json = "";  
    Gson gson = new Gson();  
    json = gson.toJson(this.filtroRojo());  
    return json;  
}
```

Mismo método que listar, pero cambiando la query de SQL y creando un nuevo JSON.



```
case 4:  
    daoCarta cartas4 = new daoCarta();  
    out.print(cartas4.listarRojos());  
    break;
```

DaoCarta con opcion = 4 que es llamada por JavaScript:



```
function verRojos(){
  fetch('gestionCarta?op=4')
  .then(response => response.json())
  .then(data => pintarTabla(data))
}
```

DaoCarta con opcion
= 4 que es llamada
por JavaScript:

```
function pintarTabla(datos) {
  let html = "<section class='section2'>";

  for (let i = 0; i < datos.length; i++) {
    html += "<div class='carta'>";
    html += "<a href = 'carta.html'><img class='Imagen' src='img/cartas/' + datos[i].foto + '' alt='Imagen de la carta'></a>";
    html += "<div class='texto'><h2 class='titulo'> + datos[i].nombre + "</h2></div>";

    html += "</div>";
  }

  html += "</section>";
  document.getElementById("listadoCartas").innerHTML = html;
}

window.onload = function() {
  llamada();
}
```



CODIGO EDITAR DE CARTA

Como usuario administrador se puede editar cada carta que este en la base de datos. Vemos un ejemplo de la forma en la que el administrador ver la tabla de cartas de forma en que sea más intuitiva para el rol asignado en este caso.

[Gestionar usuarios](#) [Crear carta](#) [Editar carta](#) [Crear Slider](#)

ID	Nombre	Color	Nivel	Coste	Coste de Digi	Poder	Foto	Accion 1	Accion 2
1	Yokomon	Rojo	2	0	0	0	BT1-001.jpg	Editar	Borrar
2	Bebydomon	Rojo	2	0	0	0	BT1-002.jpg	Editar	Borrar
3	Upamon	Azul	2	0	0	0	BT1-003.jpg	Editar	Borrar
4	Wanymon	Azul	2	0	0	0	BT1-004.jpg	Editar	Borrar
5	Kyaromon	Amarillo	2	0	0	0	BT1-005.jpg	Editar	Borrar
6	Cupimon	Amarillo	2	0	0	0	BT1-006.jpg		
7	Tanemon	Verde	2	0	0	0	BT1-007.jpg		
8	Frimon	Verde	2	0	0	0	BT1-008.jpg		
9	Monodramon	Rojo	3	2	0	3000	BT1-009.jpg		
10	Agumon	Rojo	3	3	0	2000	BT1-010.jpg		
11	Agumon Expert	Rojo	3	3	0	1000	BT1-011.jpg		
12	Biyomon	Rojo	3	3	0	2000	BT1-012.jpg		

Nombre:

Color:

Nivel:

Coste:

Coste de digievolución:

Poder:

Foto:

Examinar...

 No se ha seleccionado ningún archivo.

Guardar

En este caso, en el Script asignado a la página hay un cambio a la hora de pintar la tabla:

```
function pintarTabla(datos) {  
    let html = "<table border='2' >";  
  
    html += "<tr>";  
    html += "<th>ID</th>";  
    html += "<th>Nombre</th>";  
    html += "<th>Color</th>";  
    html += "<th>Nivel</th>";  
    html += "<th>Coste</th>";  
    html += "<th>Coste de Digi</th>";  
    html += "<th>Poder</th>";  
    html += "<th>Foto</th>";  
    html += "<th>Accion 1</th>";  
    html += "<th>Accion 2</th>";  
    html += "</tr>";  
  
    for (let i = 0; i < datos.length; i++) {  
        html += "<tr><td>" + datos[i].id + "</td>";  
        html += "<td>" + datos[i].nombre + "</td>";  
        html += "<td>" + datos[i].color + "</td>";  
        html += "<td>" + datos[i].nivel + "</td>";  
        html += "<td>" + datos[i].coste + "</td>";  
        html += "<td>" + datos[i].costeDeDigi + "</td>";  
        html += "<td>" + datos[i].poder + "</td>";  
        html += "<td>" + datos[i].foto + "</td>";  
        html += "<td><a href='crearCarta.html?id=" + datos[i].id + "&op=2'>Editar</a></td><td><a href='javascript:borrar(" + datos[i].id + ")'>Borrar</a></td>";  
        html += "</tr>";  
    }  
  
    html += "</table>";  
    document.getElementById("listadoCartas").innerHTML = html;  
}
```

Al pinchar en Acción 1 [Editar](#) nos redirecciona a la página de creación de "carta" (método **doPOST**) pero añadiendo el parámetro id de la carta y la opción 2 para actualizar.

localhost:14621/DIGIMON/crearCarta.html?id=1&op=2

Pero esta vez ya que tenemos un id pasaría la función por el ELSE:

```
Carta c = new Carta(nombre, color, nivel, coste, costeDeDigi, poder, fileName);
try {
    if(id == "") {
        c.insertar();
        response.sendRedirect("crarCarta.html");
    }
    else {
        int idInt = Integer.parseInt(id);
        c.setId(idInt);
        c.actualizar();
        response.sendRedirect("lista.html");
    }
} catch (SQLException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}
//response.sendRedirect("listado.html");
System.out.println(c.toString());
}
```

gestionCarta.java

carta.java

```
public void actualizar() throws SQLException {
    daoCarta dao = new daoCarta();
    dao.actualizar(this);
}
```

```
public void actualizar(Carta n) throws SQLException {
    PreparedStatement ps = con.prepareStatement("UPDATE cartas SET nombre=?, color=?, nivel=?, coste=?, costeDeDigi=?, poder=?, foto=? WHERE id=?");
    ps.setString(1, n.getNombre());
    ps.setString(2, n.getColor());
    ps.setString(3, n.getNivel());
    ps.setString(4, n.getCoste());
    ps.setString(5, n.getCosteDeDigi());
    ps.setString(6, n.getPoder());
    ps.setString(7, n.getFoto());
    ps.setInt(8, n.getId());
    int filas = ps.executeUpdate();
    ps.close();
}
```

daoCarta.java

CODIGO DE BORRADO DE CARTA

Al pinchar en la Acción 2 [Borrar](#) (método **doGET**) hacemos una llamada con el id de la carta y la llamada con opción 3 desde el JavaScript.

```
function borrar(id) {  
    if (confirm("Seguro que quieres borrar")) {  
        fetch('gestionCarta?id=' + id + '&op=3')  
            .then(response => response.json())  
            .then(data => pintarTabla(data))  
    } else {  
        // Uso de errores//  
    }  
}
```

```
case 3:  
    int id3 = Integer.parseInt(request.getParameter("id"));  
    daoCarta c3 = new daoCarta();  
    c3.borrar(id3);  
    System.out.println("Estoy borrando " + id3);  
    System.out.println("Estoy opcion " + opcion);  
    out.print(c3.listarJson());  
    break;
```

```
public void borrar(int id) throws SQLException {  
    String sql = "DELETE FROM cartas WHERE id=?";  
    PreparedStatement ps = con.prepareStatement(sql);  
    ps.setInt(1,id);  
  
    int filas = ps.executeUpdate();  
    ps.close();  
}
```

Tanto en el código de editar y borrar siempre se hace un uso inmediato de listar la en tabla todos los datos de la base de datos para una mejor visualización.

En resumen:

- Crear carta: Front => Back.
- Listar carta: Back => Front
- Editar carta: Front => Back y vuelta para listar.
- Borrar carta: Front => Back y vuelta para listar.

4.3.3 Ejemplos de funcionamiento

Login(Usuario).

Para ingresar en la página web el usuario tendrá que estar registrado previamente. Una vez ingresado el Email y la contraseña el usuario será redirigido a la página correspondiente en función de su grado Usuario anónimo o Administrador.

CODIGO DE NUEVO REGISTRO y ENCRIPCIÓN

Empezamos rellenando el formulario desde el front con el método POST de nuestro servlet gestiónUsuario.java



The image displays a web registration form on the left, its HTML structure in the top center, and a JavaScript validation function on the right. A red arrow points from the 'password' input field in the HTML code to the 'compararPassword()' function. A red hexagonal callout at the bottom left explains the purpose of the script.

Registro

Nombre

Apellido

Nombre del usuario

Contraseña

Repite contraseña

Email

Registrar

Volver

```
<form id="loginform" class="formularioLogin" name="formularioLogin" action="gestionUsuario" method="post">
  <input type="hidden" id="id" name="id">
  <input type="text" id="nombre" name="nombre" placeholder="Nombre" required>
  <input type="text" id="apellido" name="apellido" placeholder="Apellido" required>
  <input type="text" id="nombreUsuario" name="nombreUsuario" placeholder="Nombre del usuario" required>
  <input type="password" id="contrasena" name="contrasena" placeholder="Contraseña" required=""
    onkeyup="compararPassword()">
  <input type="password" id="repPassword" placeholder="Repite contraseña" name="repPassword"
    required="" onkeyup="compararPassword()">
  <label class="errorPassword" id="errorPassword"></label>
  <br></br>
  <input type="text" id="mail" name="mail" placeholder="Email" required>
  <button type="submit" title="Registrar" name="Ingresar">Registrar</button>
</form>
```

```
function compararPassword(){
  var password = document.getElementById("password");
  var repPassword = document.getElementById("repPassword");
  var errorPassword = document.getElementById("errorPassword");

  if(password.value.length == 0 || repPassword.value.length == 0){
    errorPassword.innerHTML = "La contraseña no puede estar vacía";
    errorPassword.style.color = "blue";
  }else if(password.value != repPassword.value){
    errorPassword.innerHTML = "La contraseña tiene que ser igual";
    errorPassword.style.color = "red";
  } else if(password.value == repPassword.value){
    errorPassword.innerHTML = "La contraseña coincide";
    errorPassword.style.color = "green";
  }
}
```

Con un script nos aseguramos que la contraseña se inserta de forma correcta.

En el método doPost cogemos los parámetros que capturamos del formulario de Registro. Y llamamos a la función insertar ya que no tenemos ningún id asignado.

```
protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
    // TODO Auto-generated method stub

    String nombre = request.getParameter("nombre");
    String apellido = request.getParameter("apellido");
    String nombreUsuario = request.getParameter("nombreUsuario");
    String contrasena = request.getParameter("contrasena");
    String mail = request.getParameter("mail");
    String permiso = request.getParameter("permiso");//
    String id = request.getParameter("id");

    Usuario u = new Usuario(nombre, apellido, nombreUsuario, contrasena, mail,permiso);

    try {
        if(id == null || id.isEmpty()) {
            u.insertar();
            response.sendRedirect("index.html");
        }
        else {
            int idInt = Integer.parseInt(id);
            u.setId(idInt);
            u.actualizar();
            response.sendRedirect("listarUsuarios.html");
        }
    } catch (SQLException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
    //response.sendRedirect("listado.html");
    System.out.println(u.toString());
}
```

En el back-end utilizaríamos la siguiente función con petición del lenguaje de base de datos .

```
public static Connection con = null;

public DaoUsuario() throws SQLException {

    this.con = DBConexion.getConnection();
}

public void insertar(Usuario u) throws SQLException {
    gestionLogin login = new gestionLogin();
    String contrasenaCifrada = login.getMd5(u.getContrasena());

    PreparedStatement ps = con.prepareStatement(
        "INSERT INTO usuarios (nombre,apellido,nombreUsuario,contrasena,mail) VALUES (?,?,,?,?) ");
    ps.setString(1, u.getNombre());
    ps.setString(2, u.getApellido());
    ps.setString(3, u.getNombreUsuario());
    ps.setString(4, contrasenaCifrada);
    ps.setString(5, u.getMail());

    int filas = ps.executeUpdate();
    ps.close();
}
```

Conectamos llamando desde la función de DBConexion.java, e insertamos en la base de datos de forma que la contraseña esté encriptada.

```
public static String getMD5(String input) {
    try {
        MessageDigest md = MessageDigest.getInstance("MD5");
        byte[] messageDigest = md.digest(input.getBytes());
        BigInteger number = new BigInteger(1, messageDigest);
        String hashtext = number.toString(16);

        while (hashtext.length() < 32) {
            hashtext = "0" + hashtext;
        }
        return hashtext;
    } catch (NoSuchAlgorithmException e) {
        throw new RuntimeException(e);
    }
}
```

CODIGO DE INGRESO DE USUARIO

<input type="text" value="Email"/>	<input type="text" value="Contraseña"/>	<input type="button" value="Ingresar"/>	<input type="button" value="Registrarse"/>
------------------------------------	---	---	--

Una vez creado el usuario nos registramos en el formulario del FRONT de forma que el usuario con permisos de persona anónima en este caso. El formulario de ingreso de usuario estará gestionado por el servlet: gestionLogin.java

```
public Usuario logueando(Usuario u, String contrasena) throws SQLException {
    String sql = "SELECT * FROM usuarios WHERE mail=? AND contrasena=?";
    PreparedStatement ps = con.prepareStatement(sql);
    ps.setString(1, u.getMail());
    ps.setString(2, contrasena);

    ResultSet rs = ps.executeQuery();

    rs.next();

    Usuario aux = new Usuario(rs.getInt(1), rs.getString(2), rs.getString(3), rs.getString(4), rs.getString(5),
        rs.getString(6), rs.getString(7));

    return aux;
}
```

DaoUsuario.java

```
public boolean logueo (String contrasena) throws SQLException {
    boolean ok = false;
    DaoUsuario dao = new DaoUsuario();

    Usuario aux = dao.logueando(this, contrasena);

    if(aux != null) {
        ok=true;
        this.setId(aux.getId());
        this.setNombre(aux.getNombre());
        this.setApellido(aux.getApellido());
        this.setNombreUsuario(aux.getNombreUsuario());
        this.setContrasena(aux.getContrasena());
        this.setMail(aux.getMail());
        this.setPermiso(aux.getPermiso());
    }

    return ok;
}
```

Usuario.java

```
String mail = request.getParameter("mail");
String contrasena = getMD5(request.getParameter("contrasena"));

Usuario u = new Usuario();
u.setMail(mail);

try {
    if(u.logueo(contrasena)) {
        session = request.getSession();


        session.setAttribute("id", u.getId());
        session.setAttribute("permiso", u.getPermiso());
        System.out.println(u.toString());


        if (u.getPermiso().equals("a")) {
            response.sendRedirect("menu.html");
        } else {
            response.sendRedirect("menuDev.html");
        }
    } else {
        response.sendRedirect("error.html");
    }
} catch (SQLException | IOException e) {
    System.out.println("Error 1");
    // TODO Auto-generated catch block
    e.printStackTrace();
}
```


gestionLogin.java


Ejemplos de funcionamiento Login (Administrador).


El administrador una vez ingresado con Email y contraseña y su correspondiente permiso podrá borrar los diferentes usuarios que estén creados en la base de datos mediante el siguiente código.

 **Listar usuarios**

 **Crear carta**

 **Editar carta**

 **Crear Slider**

 **Editar Sliders**

ID	Nombre	Apellido	Nombre de Usuario	Contraseña	Email	Accion 1
26	1234	1234	1234	81[REDACTED]055	[REDACTED]	Borrar
28	as	as	as	f97[REDACTED]9b	[REDACTED]	Borrar

CODIGO DE NUEVO LISTADO DE USUARIO

En este caso obtenemos con una consulta con el id de los usuarios en la base de datos. En este sentido la dirección se efectúa desde el back-end al front-end. Recogemos los datos que hay en la base de datos y los representamos de forma visual en el entorno gráfico.

```
public ArrayList<Usuario> listar() throws SQLException {  
    String sql = "SELECT * FROM usuarios ";  
    PreparedStatement ps = con.prepareStatement(sql);  
  
    ResultSet rs = ps.executeQuery();  
    ArrayList<Usuario> ls = null;  
  
    while (rs.next()) {  
        if (ls == null) {  
            ls = new ArrayList<Usuario>();  
        }  
  
        ls.add(new Usuario(rs.getInt(1), rs.getString(2), rs.getString(3), rs.getString(4), rs.getString(5),  
            rs.getString(6), rs.getString(7)));  
    }  
    return ls;  
}  
  
public String listarJson() throws SQLException {  
    String json = "";  
    Gson gson = new Gson();  
    json = gson.toJson(this.listar());  
    return json;  
}
```

Función para crear un array del con todos los usuarios en la base de datos y crear un JSON para posterior mente pintarlos en HTML pasando por JAVASCRIPT

En JavaScript recuperamos el Json creado y lo pintamos al front.

```
function llamada() {  
    fetch('gestionUsuario?op=1')  
    .then(response => response.json())  
    .then(data => pintarTabla(data))  
}
```

```
window.onload = function() {  
    llamada();  
}
```

```
function pintarTabla(datos) {
```

```
    let html = "<table border='2' >";
```

```
    html += "<tr>";
```

```
    html += "<th>ID</th>";
```

```
    html += "<th>Nombre</th>";
```

```
    html += "<th>Apellido</th>";
```

```
    html += "<th>Nombre de Usuario</th>";
```

```
    html += "<th>Contraseña</th>";
```

```
    html += "<th>Email</th>";
```

```
    html += "<th>Accion 1</th>";
```

```
    html += "</tr>";
```

```
    for (let i = 0; i < datos.length; i++) {
```

```
        html += "<tr><td>" + datos[i].id + "</td>";
```

```
        html += "<td>" + datos[i].nombre + "</td>";
```

```
        html += "<td>" + datos[i].apellido + "</td>";
```

```
        html += "<td>" + datos[i].nombreUsuario + "</td>";
```

```
        html += "<td>" + datos[i].contrasena + "</td>";
```

```
        html += "<td>" + datos[i].mail + "</td>";
```

```
        html += "<td><a href='javascript:borrar(" + datos[i].id + ")'>Borrar</a></td>";
```

```
        html += "</tr>";
```

```
    }
```

```
    html += "</table>";
```

```
    document.getElementById("listadoUsuarios").innerHTML = html;
```

```
}
```

```
window.onload = function() {
```

```
    llamada();
```

```
}
```

```
switch (opcion) {  
    case 1:  
        DaoUsuario usuarios1 = new DaoUsuario();  
        out.print(usuarios1.listarJson());  
        break;
```

Llamada a la opción 1 del servlet por método doGet desde JavaScript

Creamos tabla con todos los usuarios con la información del Json.

CODIGO DE BORRADO DE USUARIO

Función de borrado con consulta SQL de borrado de datos.

```
public void borrar(int id) throws SQLException {  
    String sql = "DELETE FROM usuarios WHERE id=?";  
    PreparedStatement ps = con.prepareStatement(sql);  
    ps.setInt(1, id);  
  
    int filas = ps.executeUpdate();  
    ps.close();  
}
```

En este caso no editamos los usuarios desde el administrador ya que sería el propio usuario quien tendría la función de modificar dichos datos.

Ejemplos de funcionamiento de Sliders

En todo el Front existe un apartado de sliders. Ejemplo:



Estos tienen la función de merchandising y de darle dinamismo a la página de forma visual.

```
window.addEventListener("DOMContentLoaded", function(){

    let sliderPrincipal = document.getElementById("sliderPrincipal");
    let flechaIzq = document.getElementById("flechaIzq");
    let flechaDer = document.getElementById("flechaDer");

    let rutas = ["img/sliders/slider1.jpg", "img/sliders/slider2.jpg", "img/sliders/slider3.jpg"];
    let posicionActual = 0;

    flechaIzq.addEventListener("click", retroceder);
    flechaDer.addEventListener("click", avanzar);

    function retroceder() {
        posicionActual--;
        if (posicionActual < 0) {
            posicionActual = rutas.length - 1;
        }
        cambiarSlider();
    }

    function avanzar() {
        posicionActual++;
        if (posicionActual >= rutas.length) {
            posicionActual = 0;
        }
        cambiarSlider();
    }

    function cambiarSlider() {
        sliderPrincipal.src = rutas[posicionActual];
    }

    // Función para avanzar automáticamente cada 4 segundos
    setInterval(avanzarAutomaticamente, 4000);

    function avanzarAutomaticamente() {
        avanzar();
    }

});
```

Tiene dos funciones, la de avanzar y retroceder cuando el usuario quiera, y la de cambiar la foto en intervalos de segundos.

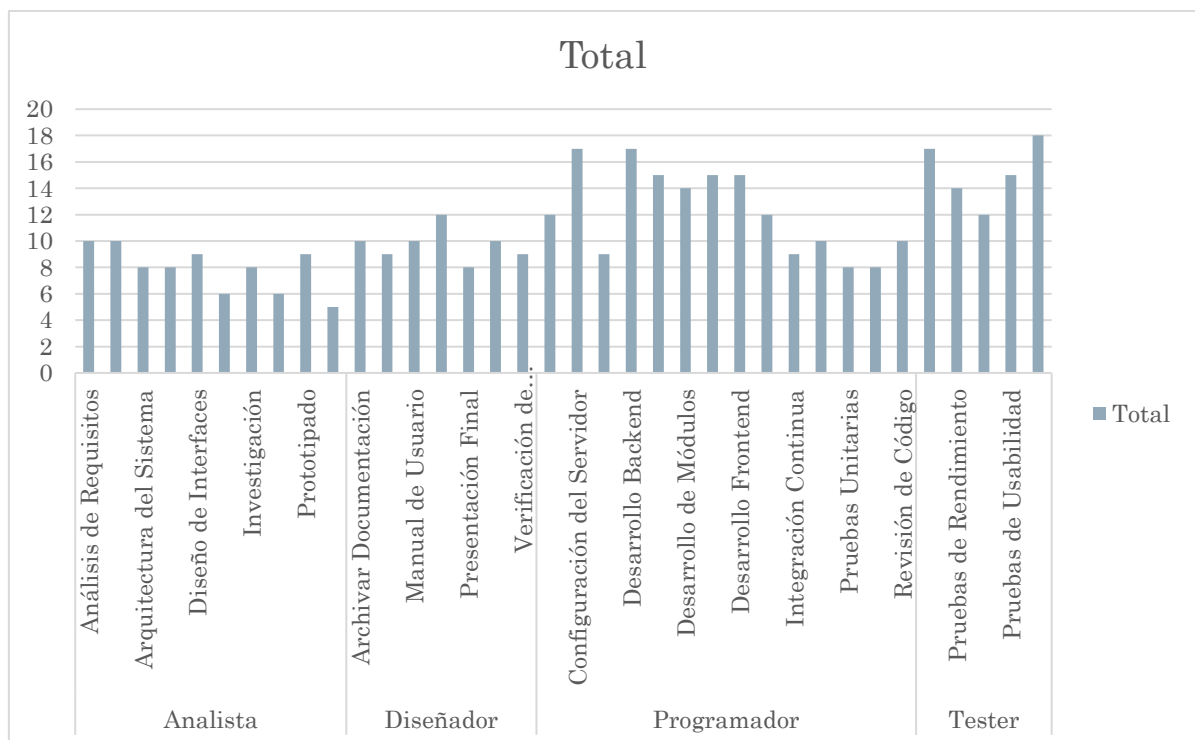
5. Plan de proyecto

5.1 Equipo de trabajo

Este proyecto constaría de:

- Jefe de proyecto: Programador senior encargado de la organización de proyecto con más de 5 años de experiencia tanto en Back como Front-End.
- Programadores Back-End: 2 personas encargadas de la parte servidor-usuario con conocimientos en análisis, modela y desarrollo del lenguaje JAVA.
- Programadores Front-End: 2 personas con experiencia en el desarrollo web y conocedor de lenguajes como HTML y JAVASCRIPT y 1 persona encargada del uso del lenguaje CSS.
- Tester: Ingeniero informático con experiencia de 3 años en desarrollo de software y servidores Apache.
- Diseñador: Persona con experiencia de un año diseño e ilustración. Conocimientos en Photoshop.

5.2 Plan de trabajo



Fases y subfases de desarrollo.	Rol	Horas	Fecha inicio	Fecha Final
Análisis y Diseño				
Análisis de Requisitos	Analista	10	16/03/2024	20/03/2024
Arquitectura del Sistema	Analista	8	26/03/2024	29/03/2024
Diagramas UML	Analista	8	26/03/2024	29/03/2024
Diseño de Interfaces	Analista	9	23/03/2024	25/03/2024
Modelado de Datos	Analista	6	20/03/2024	22/03/2024
Prototipado	Analista	9	21/03/2024	25/03/2024
Estudio del Proyecto				
Análisis de Viabilidad	Analista	10	10/03/2024	15/03/2024
Estimación de Costos	Analista	6	16/03/2024	19/03/2024
Investigación	Analista	8	10/03/2024	12/03/2024
Revisión de Requisitos	Analista	5	13/03/2024	15/03/2024
Implementación				
Base de Datos	Programador	12	05/04/2024	09/04/2024
Configuración del Servidor	Programador	17	30/03/2024	04/04/2024
Corrección de Errores	Programador	9	05/05/2024	07/05/2024
Desarrollo Backend	Programador	17	30/03/2024	03/04/2024
Desarrollo de API	Programador	15	10/04/2024	14/04/2024
Desarrollo de Módulos	Programador	14	15/04/2024	19/04/2024
Desarrollo de Servicios Web	Programador	15	25/04/2024	30/04/2024
Desarrollo Frontend	Programador	15	04/04/2024	08/04/2024
Integración	Programador	12	09/04/2024	12/04/2024
Integración Continua	Programador	9	16/04/2024	18/04/2024
Optimización del Código	Programador	10	20/04/2024	24/04/2024
Pruebas Unitarias	Programador	8	13/04/2024	15/04/2024
Refactorización	Programador	8	08/05/2024	11/05/2024
Revisión de Código	Programador	10	19/04/2024	22/04/2024
Memoria y Documentación				
Archivar Documentación	Diseñador	10	26/05/2024	31/05/2024
Documentación Técnica	Diseñador	9	12/05/2024	14/05/2024
Manual de Usuario	Diseñador	10	15/05/2024	18/05/2024
Preparación de Informes	Diseñador	12	16/05/2024	20/05/2024
Presentación Final	Diseñador	8	19/05/2024	22/05/2024
Revisión de Documentación	Diseñador	10	11/05/2024	15/05/2024
Verificación de Documentos	Diseñador	9	21/05/2024	25/05/2024
Pruebas				
Pruebas de Integración	Tester	17	06/05/2024	10/05/2024
Pruebas de Rendimiento	Tester	14	27/04/2024	30/04/2024
Pruebas de Seguridad	Tester	12	01/05/2024	04/05/2024
Pruebas de Usabilidad	Tester	15	01/05/2024	05/05/2024
Pruebas Funcionales	Tester	18	23/04/2024	26/04/2024
Total, general		394		

5.3 Análisis de costes

Analizamos el coste de del proyecto.

- * Coste personal:
- * Coste hardware:
- * Coste software:
- * Costes generales:

COSTE PERSONAL:

Jefe de proyecto: 40 horas x 50€ = 2000€
Analista: 79 horas x 20 € = 1580 €
Programador y diseñador: 236 horas x 15 € = 3540 €
Tester: 76 horas x 30 euros = 2280 €
TOTAL: 9400 €

COSTE HARDWARE:

Equipo PC 1.500 €
Hosting, Servidor dedicado 120 €/mes.
3 meses para la creación 360 €.

COSTE SOFTWARE:

Todo el software utilizado en gratuito u open source.

TOTAL:

14620 €

6. Trabajos futuros.

- Por parte del usuario:
 - Modificación de sus datos personales.
 - Creación de su propia lista de compra y cartas deseadas.
- Por parte del administrador:
 - Modificar los sliders de cada página desde una menú lista.
 - Creación de un pago online.
 - Mayor seguridad en el código y base de datos.
 - Mejoras en la parte visual de la página.

7. CONCLUSIONES

A lo largo del desarrollo de mi tienda online de cartas de Digimon, he logrado importantes avances que sientan las bases para una plataforma robusta y eficiente. Hasta la fecha, he implementado un sistema de gestión de inventario que permite la entrada y salida de productos de manera eficiente, así como una interfaz de usuario amigable que facilita la navegación y la búsqueda de cartas específicas. Además, se han establecido funcionalidades básicas para el proceso de compra, incluyendo el carrito de compras.

Sin embargo, el proyecto aún no está finalizado. Quedan pendientes varias mejoras y adiciones clave, como la optimización de la experiencia del usuario, la implementación de un sistema de recomendaciones personalizadas y la integración completa de una plataforma de atención al cliente. Asimismo, es crucial realizar pruebas exhaustivas para garantizar la estabilidad y seguridad del sistema antes de un lanzamiento.

En resumen, aunque hemos avanzado significativamente, aún queda trabajo por hacer para completar nuestra tienda online de cartas de Digimon. Con una planificación cuidadosa y un enfoque en la calidad, estoy en el buen camino para crear una plataforma que no solo satisfaga las necesidades de los coleccionistas y jugadores de Digimon, sino que también establezca un nuevo estándar en el comercio electrónico de cartas coleccionables.

Anexos

Java

Tutoriales Javanet: Proporciona tutoriales detallados sobre programación en Java, desde lo básico hasta lo avanzado.
Tutoriales Javanet

Aprende.org: Curso gratuito de Java, cubriendo conceptos básicos y avanzados. Curso de Java en Aprende.org

JavaScript

Mozilla Developer Network (MDN): Documentación oficial de JavaScript, muy completa y en español. JavaScript en MDN

Desarrolloweb.com: Serie de tutoriales y artículos sobre JavaScript para todos los niveles. JavaScript en Desarrolloweb.com

HTML

HTML.net: Tutorial completo de HTML5 en español, ideal para principiantes. Tutorial de HTML5 en HTML.net

W3schools: Tutoriales y ejemplos interactivos de HTML. HTML en w3schools

CSS

CSS en MDN: Documentación oficial de CSS en español, con guías y tutoriales. CSS en MDN

Desarrolloweb.com: Tutoriales y artículos sobre CSS, desde lo básico hasta técnicas avanzadas. CSS en Desarrolloweb.com

Cursos en línea

CódigoFacilito: Plataforma con cursos gratuitos y de pago sobre desarrollo web, incluyendo Java, JavaScript, HTML y CSS.
CódigoFacilito

Udemy: Ofrece cursos en español sobre Java, JavaScript, HTML y CSS, con lecciones en video. Udemy

