ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA

DESARROLLO DE UN PROTOTIPO DE APLICACIÓN WEB PARA LA GESTIÓN DE PEDIDOS PARA UN RESTAURANTE

TRABAJO DE TITULACIÓN PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN ELECTRÓNICA Y REDES DE INFORMACIÓN

BRYAN ALEXIS SANTANDER ARAGÓN

bryan.santander@epn.edu.ec

DIRECTOR: RAÚL DAVID MEJÍA NAVARRETE, M.Sc

david.mejia@epn.edu.ec

Quito, agosto 2022

AVAL

Certifico que el presente trabajo fue desarrollado por Bryan Alexis Santander Aragón, bajo mi supervisión y tutoría.

Ing. Raúl David Mejía Navarrete, M.Sc.
DIRECTOR DEL TRABAJO DE TITULACIÓN

DECLARACIÓN DE AUTORIA

Yo, Bryan Alexis Santander Aragón, declaro bajo juramento que el trabajo 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.

A través de la presente declaración cedo mis derechos de propiedad intelectual correspondientes a este trabajo, a la Escuela Politécnica Nacional, según lo establecido por la Ley de Propiedad Intelectual, por su Reglamento y por la normatividad institucional vigente.

_____

Bryan Alexis Santander Aragón

DEDICATORIA

A mi padre, el hombre más maravilloso y valiente que conozco, que a pesar de las dificultades y pruebas que le ha puesto la vida nunca se ha rendido y ha salido adelante.

A mi madre, que en paz descanse, la persona que me dio la vida, que me inculcó la importancia del estudio y que ha sido mi mayor motivación para seguir adelante.

A mi hermano Freddy, que además de hermano ha sido el mejor amigo y confidente que la vida me regalo, compañero interminable de aventuras y la persona que siempre ha estado ahí para apoyarme en los buenos y malos momentos.

Al mejor regalo que me dio la vida mis sobrinos Julián y Adrián que son las personitas que alegran mis días con sus ocurrencias y travesuras, los quiero sobrinos.

A mis amigos de la universidad con los que conviví y compartí miles de experiencias y anécdotas a lo largo de todo el camino universitario y de la vida.

A mis tíos, primos y amigos que han formado parte mi vida y que siempre me han estado ahí para mí, gracias por todo.

Finalmente, a la Escuela Politécnica Nacional por formarme como profesional.

AGRADECIMIENTO

Gracias a mis padres Alberto Mesías Santander Flores y Norma Ximena Aragón Ruiz, quienes fueron las personas que me dieron la vida y me enseñaron a ser una persona de bien y que me han inculcado todos los valores que me definen como persona, me han enseñado a luchar por mis sueños y cumplir mis metas.

A mi hermano Freddy que siempre ha estado guiándome y que me enseñó a seguir sus pasos, a quien considero mi mejor amigo, que siempre ha estado ahí para mí en los buenos y malos momentos que hemos pasado, gracias por todo.

A mis tías Hilda, Lucía, Mercedes, Carmen y Enma que siempre estuvieron pendientes de mi progreso académico, pero por sobre todo siempre me brindaron su cariño y apoyo, decir gracias es poco.

Gracias a la querida Escuela Politécnica Nacional, institución que me brindó la oportunidad de prepararme como un profesional que aporte a la sociedad y permitirme cumplir uno de los más grandes objetivos de mi vida. También un especial agradecimiento al Colegio Alfonso del Hierro “La Salle”, institución que me formó durante 13 años y a la cual guardo un cariño especial.

A todos los profesores de la Escuela Politécnica Nacional por aportar cada uno con sus enseñanzas y experiencias a nivel profesional pero también personal, en especial a mi tutor, David Mejía M.Sc., por sus guía, apoyo y paciencia en el desarrollo del presente Trabajo de Titulación.

A los grandes amigos que la EPN me entregó: Gabriel Guañuna, David Proaño, Francisco Salazar, Leonardo Fonseca, Joseph Bravo, Esteban Machado, Matías Romero y en especial a Bryan López el primer amigo que tuve desde prepo, gracias a todos por esos días llenos de estudio, risas, comidas, conversaciones, giras, definitivamente sin ustedes la vida universitaria no hubiera sido igual.

Finalmente, a toda mi familia y amigos que han formado parte de mi vida, infinitas gracias.

ÍNDICE DEL CONTENIDO

RESUMEN

El desarrollo del presente Trabajo de Titulación consiste en el diseño e implementación de un prototipo que permite gestionar la toma de pedidos y visualizar las ventas que genera un restaurante buscando automatizar algunas tareas y procesos que se desarrollan en el mismo.

El prototipo permite realizar tareas como gestionar los productos del menú del restaurante, gestionar los usuarios registrados en el sistema, realizar procesos de autenticación, gestión de los pedidos de los clientes, visualizar las ventas generadas por el restaurante, entre otras.

Este documento está formado por 4 capítulos. El capítulo 1 abarca un resumen de los principales conceptos de la web y el protocolo HTTP, además incluye conceptos sobre el patrón arquitectónico MVC Modelo-Vista-Controlador, la metodología Kanban y una recopilación de las tecnologías y herramientas utilizadas en el desarrollo de aplicaciones web.

En el capítulo 2 se presenta el levantamiento de los requerimientos funcionales y no funcionales del prototipo, el *backlog* de tareas Kanban, la arquitectura del prototipo, la fase de diseño que incluyen todos los diagramas y los *sketches* de las vistas. Adicionalmente se presenta un resumen de la fase de implementación mediante la codificación del modelo, los controladores y las vistas necesarias para el correcto funcionamiento del prototipo.

En el capítulo 3 se presentan las pruebas de funcionamiento del prototipo, necesarias para garantizar que el mismo cumple con los requerimientos levantados, además, en este capítulo también se muestran los resultados de las encuestas y las correcciones finales realizadas al prototipo.

El capítulo 4 contiene las conclusiones y recomendaciones una vez culminado el presente Trabajo de Titulación.

Para finalizar los Anexos contienen las historias de usuario, los *sketches* de las vistas, la encuesta del uso de la aplicación, el código fuente del prototipo, el manual de usuario y el manual de instalación.

PALABRAS CLAVE: MVC, backlog, Kanban, arquitectura de software, sketches.

ABSTRACT

The development of this Degree Project consists of the design and implementation of a prototype that allows managing order taking and visualizing the sales generated by a restaurant, seeking to automate some tasks and processes that are developed in it.

The prototype allows tasks such as managing the products on the restaurant menu, managing users registered in the system, performing authentication processes, managing customer orders, viewing the sales generated by the restaurant, among others.

This document consists of 4 chapters. Chapter 1 covers a summary of the main concepts of the web and the HTTP protocol, it also includes concepts about the MVC Model-View-Controller architectural pattern, the Kanban methodology and a compilation of the technologies and tools used in the development of web applications.

Chapter 2 presents the lifting of the functional and non-functional requirements of the prototype, the backlog of Kanban tasks, the architecture of the prototype, the design phase that includes all the diagrams and the sketches of the views. Additionally, a summary of the implementation phase is presented through the coding of the model, the controllers, and the views necessary for the proper functioning of the prototype.

Chapter 3 presents the functional tests of the prototype, necessary to guarantee that it meets the requirements raised, in addition, this chapter also shows the results of the surveys and the final corrections made to the prototype.

Chapter 4 contains the conclusions and recommendations once this Degree Project has been completed.

Finally, the Annexes contain the user stories, the view sketches, the application usage survey, the prototype source code, the user manual, and the installation manual.

KEY WORDS: MVC, backlog, Kanban, software architecture, sketches

1. INTRODUCCIÓN

Debido a la nueva realidad que está atravesando el mundo en general a causa de la pandemia del COVID-19, muchos restaurantes han tenido que cambiar la forma de realizar sus actividades diarias, algunas de ellas han encontrado en el software una solución para poder automatizar algunos procesos como la toma o entrega de los pedidos.

En el caso del restaurante "Don Ruiz", no cuenta con un sistema adecuado que permita automatizar algunas tareas, por lo que el proceso de toma de órdenes que realizan los meseros se los hace en papel y a mano interactuando directamente con los clientes, para posteriormente pasarlos a la cocina donde se interactúa con el personal de la cocina; al momento de solicitar la cuenta los clientes se acercan a la caja donde se interactúa con la persona a cargo de los cobros.

Todo este procedimiento aumenta el riesgo de contagio de COVID-19 entre todos los actores mencionados; además, los registros de ventas del día se los almacena de forma tradicional en un cuaderno. Por lo anteriormente expuesto se plantea el desarrollo de un prototipo que presente todas las opciones del menú disponibles a ser elegidas por los clientes, automatice y personalice la toma de pedidos de los clientes, gestione el pedido por parte del personal del restaurante, solicite la cuenta de los clientes, y permita llevar un registro de las ventas del día.

1.1. OBJETIVOS

El objetivo general del presente Trabajo de Titulación es:

- Desarrollar un prototipo de aplicación web para la gestión de pedidos para un restaurante.

Los objetivos específicos del presente Trabajo de Titulación son:

- Analizar las herramientas que se van a utilizar para el desarrollo del presente trabajo de titulación.
- Diseñar en función de los requerimientos establecidos cada uno de los componentes necesarios para el prototipo.
- Implementar el prototipo con base en los componentes diseñados.
- Evaluar los resultados con base a las pruebas de funcionalidad.

1.2. ALCANCE

En este trabajo se plantea desarrollar un prototipo de aplicación web para el restaurante "Don Ruiz" que permita automatizar la toma de los pedidos, gestionar los pedidos por parte del personal del restaurante, solicitar la cuenta y llevar un registro de las ventas del día. Las herramientas que se planea utilizar para realizar el prototipo son Visual Studio, ASP.NET Core MVC, Microsoft Entity Framework, SQL Server, Bootstrap, GitHub, Scaffolding, Razor entre otros

Se realizará una entrevista con los propietarios y empleados del restaurante con el objetivo de determinar los requerimientos funcionales y no funcionales del prototipo y la información obtenida será recopilada mediante historias de usuario. El prototipo contará con 3 roles de usuarios: administrador, empleado y cliente, que determinarán la funcionalidad a la cual los usuarios tendrán acceso en la aplicación web.

Para el usuario administrador existirán vistas que le permita realizar las siguientes acciones:

- Iniciar sesión una vez autenticado.
- Realizar un CRUD (crear, mostrar, actualizar y eliminar) para gestionar los productos del menú que ofrece el restaurante.
- Realizar un CRUD (crear, mostrar, actualizar y eliminar) de los empleados del restaurante.
- Gestionar los cobros:
  - Visualizar las solicitudes de cobro de los clientes con el valor total a pagar.
  - Confirmar el pago de los clientes.
- Gestionar las ventas del día:
  - Mostrar el total de las ventas del día.

Para el usuario empleado existirán vistas que le permita realizar las siguientes acciones:

- Iniciar sesión una vez autenticado.
- Gestionar los pedidos:
  - Visualizar los pedidos de los clientes asociados al número de mesa correspondiente.
  - Indicar cuando el pedido esté listo.
  - Indicar los pedidos entregados.

Para el usuario cliente existirán vistas que le permita realizar las siguientes acciones:

- Realizar el pedido:

- Visualizar las opciones del menú disponibles con su respectivo precio.
- Escoger el producto que desee.
- Confirmar el pedido.
- Solicitar la cuenta.

Se empleará como base la metodología Kanban en el desarrollo de este trabajo de titulación. Además, durante cada una de las etapas de desarrollo del prototipo se realizarán pruebas de funcionalidad y validación con ayuda de los empleados y propietarios del restaurante, y finalmente se aplicará una encuesta, al menos a 5 personas para verificar que el prototipo cumpla con los requerimientos establecidos.

Como parte de este trabajo de titulación se generará un producto final demostrable.

1.3. MARCO TEÓRICO

En este capítulo se presenta un resumen de los principales conceptos de la web y el protocolo HTTP, además incluye conceptos sobre el patrón arquitectónico MVC Modelo-Vista-Controlador, la metodología Kanban y una recopilación de las tecnologías y herramientas utilizadas en el desarrollo de aplicaciones web.

1.3.1. WORLD WIDE WEB

El *World Wide Web* o simplemente la web es un conjunto de páginas interconectadas a las que se puede acceder a través de navegadores web para obtener información, documentos, audios, contenido multimedia y acceder a aplicaciones dinámicas usando dispositivos como computadores, laptops o dispositivos móviles conectados a Internet.

La web nació a principios de los años 90 en un laboratorio del CERN[1] en Ginebra Suiza y su creador fue *Tim Berners-Lee*. La idea de *Berners-Lee* consistía en crear un sistema de red de nodos denominado *mesh* que estuviera formado por una colección de sitios creados mediante el lenguaje de marcado HTML[2] y que se encontrarán alojados en servidores, pudiendo vincular documentos e información con la finalidad de que los usuarios puedan acceder y navegar entre ellos; a este primer sistema también se le conoce como web 1.0 [1].

[1] CERN: Consejo Europeo de Investigación Nuclear, es uno de los centros de investigación más importantes del mundo.

[2] HTML: Lenguaje de Marcado de Hipertexto, es uno de los principales lenguajes utilizados en el desarrollo web.

1.3.1.1. EVOLUCIÓN DE LA WEB

El éxito de la web se debe a tres elementos fundamentales que fueron establecidos por su creador [1]:

a) El protocolo HTTP[3];
b) El lenguaje de marcado HTML; y,
c) La URL[4].

La web 1.0 consistía en un sistema muy básico y lento que permitía a los usuarios únicamente buscar y leer la información a través de un navegador web, los usuarios no podían compartir, enviar o modificar la información de los sitios web, es decir la información se mantenía estática, y para su actualización se requería del administrador de dichos sitios. La web 1.0 fue utilizada hasta el año 2003 y empleaba principalmente los tres elementos fundamentales.

En 1999, Darcy DiNucci anunció la segunda versión de la web, denominado web 2.0 y la misma fue presentada formalmente en el año 2004 por Dale Dougherty. La web 2.0 implementó tecnologías como JavaScript[5], CSS, REST[6], XML[7], entre otros; además, es conocida también como una web de lectura y escritura debido a que los usuarios poseen una mayor interacción con los sitios web, son capaces de modificar y compartir la información, así como la capacidad de comunicarse unos con otros a través de *blogs* y redes sociales.

La web 3.0 apareció en el 2014 y se caracteriza por permitir a los usuarios interactuar con aplicaciones y sitios web dinámicos ya que dota de cierto nivel de inteligencia a las computadoras, laptops y dispositivos móviles, además, la web 3.0 convirtió al Internet en una gran base de datos puesto que es factible manejar e interpretar grandes volúmenes de datos e información. En la web 3.0 se puede obtener la información a grandes velocidades y de distintas fuentes para mostrarla al usuario final por medio de un navegador web mediante un explorador como Google. Otra de las características de la web 3.0 es que las aplicaciones y plataformas se encuentran descentralizadas, es decir, permiten a los usuarios interactuar sin la necesidad de utilizar un servidor central que se encargue de la

[3] HTTP: Protocolo de Transferencia de Hipertexto, es un protocolo de capa aplicación.

[4] URL: Localizador de Recursos Uniforme, es utilizado para localizar recursos en la web.

[5] JavaScript: Es un lenguaje interpretado que permite agregar lógica a la aplicación web que se puede ejecutar en el lado del cliente o en el lado del servidor.

[6] REST: Transferencia de Estado Representacional, es un estilo arquitectónico para el desarrollo de sistemas distribuidos.

[7] XML: Lenguaje de Marcado Extensible, es un lenguaje utilizado para el intercambio de datos.

administración de la plataforma o aplicación. En la Figura 1.1 se presenta un breve resumen de los cambios de la web.

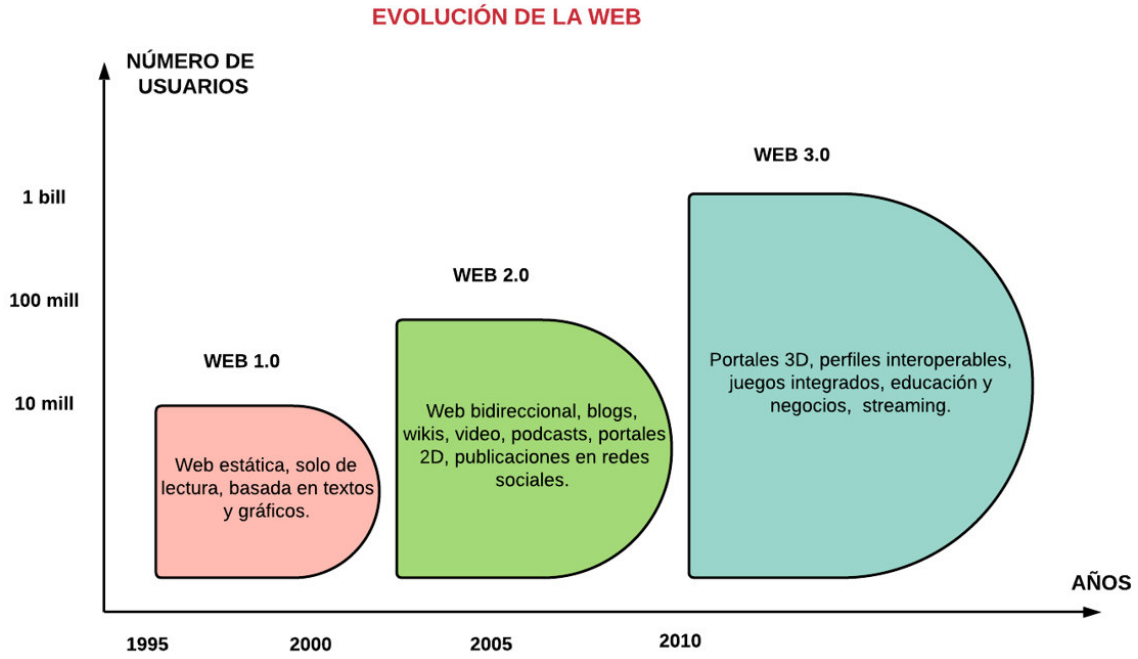


Figura 1.1. Evolución de la web [1]

1.3.2. DEFINICIONES GENERALES

1.3.2.1. PROTOCOLO HTTP

HTTP es un protocolo orientado a conexión. Este protocolo es el encargado del intercambio de recursos web mediante mensajes generados por un cliente denominados peticiones y mensajes generados por un servidor denominados respuestas. El protocolo HTTP es un protocolo de la capa aplicación y se basa en una arquitectura cliente-servidor. Los servidores que lo emplean atienden las solicitudes en el puerto 80 de TCP[8]. Además, HTTP es capaz de transportar documentos escritos en lenguaje HTML, así como como transportar datos, documentos, contenido multimedia e incluso puede enviar formularios con datos.

El protocolo HTTPS[9] es una variante del protocolo HTTP que ha ganado mucho terreno en los últimos tiempos debido a la necesidad de mantener la confidencialidad de la información y permitir que los usuarios gocen de una experiencia segura y privada cuando se encuentran navegando en la web. El uso de HTTPS permite proteger la información

[8] TCP: Protocolo de Transmisión de Control: protocolo de la capa transporte.

[9] HTTPS: Protocolo de Transferencia de Hipertexto Seguro, versión segura de HTTP.

mediante la utilización de protocolos como SSL[10] o TLS[11] los cuáles usan los tres pilares fundamentales de la seguridad de la información: el cifrado, la integridad de los datos y la autenticación.

FUNCIONAMIENTO DEL PROTOCOLO HTTP

El protocolo HTTP inicia en el lado del cliente, en el cual mediante un navegador web se realiza una petición a un servidor web, solicitándole un recurso definido a través de una URL, el servidor web gestiona la petición solicitada y responde con los datos solicitados por el cliente; adicionalmente el cliente puede generar más peticiones para solicitar datos adicionales que requiera el recurso inicialmente definido, sobre todo si este es un documento HTML, como hojas de estilo CSS[12], *scripts*, contenido multimedia, anuncios, entre otros. Una vez que toda la información se recopila en el lado del cliente, el navegador web se encarga de hacer el *parsing* respectivo y combinar todos estos elementos dando como resultado una página web con una interfaz amigable para que el usuario final pueda interactuar con la misma [2]. En la Figura 1.2 se presenta un ejemplo de lo indicado.

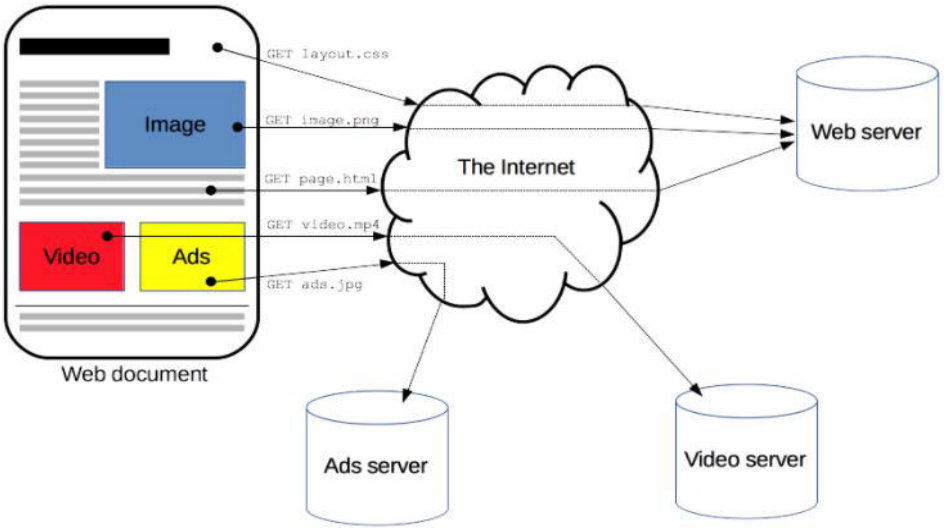


Figura 1.2. Funcionamiento del protocolo HTTP [2]

MÉTODOS USADOS EN HTTP

Los métodos o verbos de HTTP son operaciones que intervienen en la comunicación entre el navegador web y el servidor web para permitir realizar distintas acciones sobre un determinado recurso definido a través de una URL, dichos métodos se encuentran

[10] SSL: Capa de Sockets Seguros, es un protocolo criptográfico.

[11] TLS: Seguridad de la Capa transporte, es un protocolo criptográfico.

[12] CSS: Hojas de estilo en cascada, lenguaje de *front-end* orientado a la parte visual y estética del sitio web.

definidos en la sección 4 de la RFC[13] 7231. Entre los principales métodos HTTP se encuentran los siguientes:

- **`GET`:** Es utilizado para solicitar un recurso.
- **`HEAD`:** Solicita la cabecera de un recurso, es similar al método `GET`.
- **`POST`:** Envía datos hacia el servidor.
- **`PUT`:** Sube o actualiza un recurso especifico.
- **`DELETE`:** Elimina un recurso especifico.
- **`CONNECT`:** Permite establecer un acceso hacia otro servidor que posee un recurso específico.
- **`TRACE`:** Se utiliza como bucle de retorno del mensaje.
- **`OPTIONS`:** Devuelve información sobre todos los métodos HTTP que un servidor web es capaz de utilizar.

FORMATO DE LOS MENSAJES HTTP

El protocolo HTTP utiliza dos tipos de mensajes: las peticiones que son realizadas en el lado de cliente hacia el servidor y las respuestas que las generadas en el lado del servidor y se envían hacia el cliente, cada una cuenta con su propio formato.

Las peticiones emplean el formato presentado en la Figura 1.3.

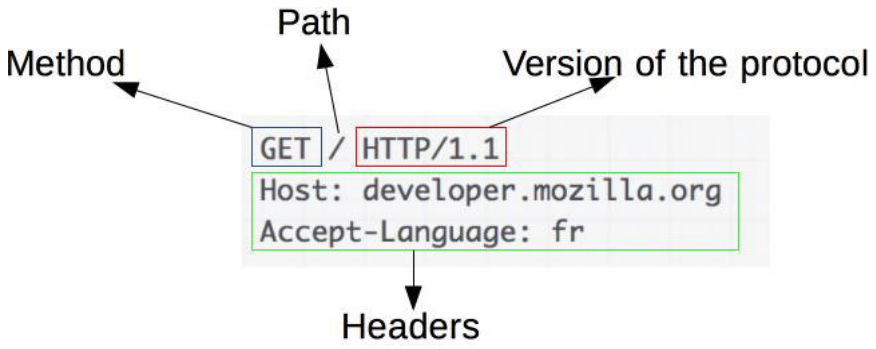


Figura 1.3. Formato de la petición HTTP [2]

En la petición se encuentran los siguientes campos:

- **MÉTODO**: Establece la operación que el cliente desea realizar mediante la utilización de métodos HTTP como `GET`, `POST`, etc.
- **RUTA O RECURSO:** Definido a través de una URL, que indica el recurso sobre el cual se solicita se ejecute el método HTTP.

[13] RFC (*Request for Comments*): es la documentación de los protocolos y tecnologías que se utilizan en internet.

- **VERSIÓN:** Define la versión del protocolo HTTP utilizado, puede ser HTTP/1.0, HTTP/1.1 o HTTP/2.
- **CABECERAS:** Contienen información adicional requerida para procesar la solicitud.

Las respuestas emplean el formato presentado en la Figura 1.4.

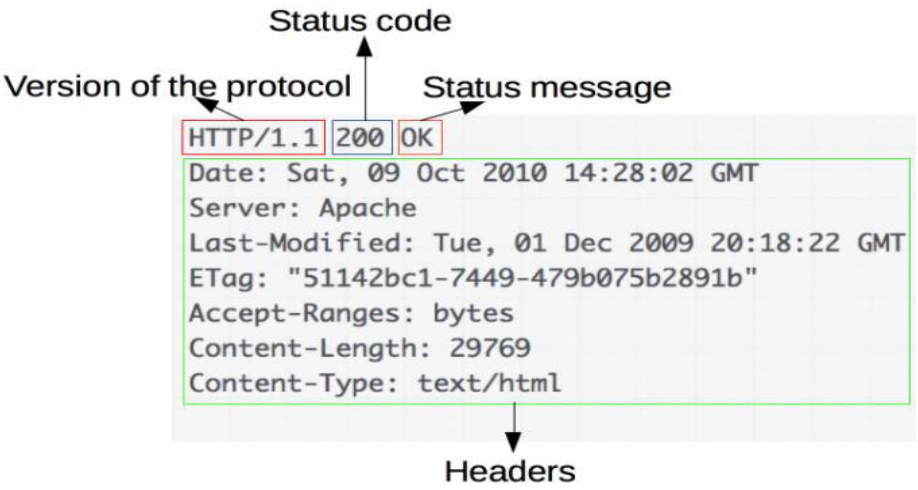


Figura 1.4. Formato de la respuesta HTTP [2]

En la respuesta se encuentran los siguientes campos:

- **VERSIÓN:** Define la versión del protocolo HTTP empleado.
- **CÓDIGO DE ESTADO:** Indica el estado del pedido mediante un código numérico.
- **MENSAJE DE ESTADO:** Indica una descripción sobre el código de estado.
- **CABECERAS:** Contiene información adicional sobre la respuesta.
- **CUERPO:** Contiene la respuesta a la solicitud.

1.3.2.2. SERVIDOR WEB

De acuerdo con [3], un servidor web es un software que se encuentra alojado en un servidor y es el encargado de recibir, almacenar, procesar y despachar recursos como páginas web acorde con las peticiones realizadas desde el cliente. El servidor web proporciona los recursos solicitados haciendo uso del protocolo HTTP o HTTPS para su transporte, para que posteriormente los navegadores web se encarguen de mostrar el resultado a los usuarios finales. En la actualidad existen una gran variedad de servidores web que distan mucho uno de otro y a la hora de escoger uno es muy importante tomar en cuenta el sistema operativo que posee el servidor, para el caso de Windows Server el servidor web

más común es IIS[14] propio de Microsoft, en tanto que para los servidores que tienen un sistema operativo basado en Linux los servidores web más utilizados son Apache, Nginx y LiteSpeed.

INTERNET INFORMATION SERVICES

El servidor IIS propiedad de Microsoft es un servidor que está orientado a ofrecer distintos servicios como FTP[15] para transferencia de archivos, SMTP[16] para correo electrónico, servicios para la web mediante los cuales es factible alojar sitios web, entre otros.

En el año 2016, con el lanzamiento de la versión 10.0, IIS comenzó a experimentar un incremento sustancial en la cuota de mercado del mundo de servidores web llegando a tener 688 millones de instalaciones alrededor del mundo de acuerdo con NETCRAFT[17], seguido de Nginx con alrededor de 358 millones de instalaciones y de Apache con cerca de 313 millones de instalaciones [4]. Aunque el servidor IIS ofrece una variedad de servicios, ha sido utilizado principalmente como servidor web ya sea en Internet, en una intranet o en servidores privados donde posee la mayor cantidad de usuarios debido a la importancia en el mercado empresarial para ser utilizado en aplicaciones propias de las empresas.

1.3.2.3. APLICACIÓN WEB

Las aplicaciones web surgieron ante la necesidad de tener sistemas cada vez más orientados a la computación que sean automatizados, autónomos y desarrollados en plataformas computacionales independientes. Las aplicaciones web consumen los servicios web ubicados en un servidor a través del cual los usuarios tienen acceso para poder conectarse y compartir diferente tipo de información mediante la aplicación utilizando una red como el Internet o una intranet.

Una aplicación web es una tecnología a la que se accede a través de un navegador web que cuente con una conexión a Internet o que se encuentre dentro de una intranet, permitiendo un ahorro de tiempo ya que no necesitan ser instaladas en un computador o dispositivo móvil. Las aplicaciones web son desarrolladas mediante lenguajes que son soportados por los navegadores web como JavaScript, HTML, CSS, entre otros.

---

[14] IIS: *Internet Information Services*, es un servidor web y de otros servicios para Windows.

[15] FTP: Protocolo de transferencia de archivos, es utilizado para brindar un servicio de transferencia de archivos.

[16] SMTP: Protocolo para transferencia simple de correo, es un protocolo utilizado para el intercambio de correos.

[17] NETCRAFT: Es una empresa especializada en el análisis de servidores y alojamiento web

En la actualidad existen una gran variedad de aplicaciones web para realizar muchas actividades como el correo, redes sociales, tiendas en línea, almacenar información, *blogs*, videoconferencias y muchas más; todas estas aplicaciones han automatizado casi por completo muchos procesos que antes tardaban mucho tiempo en realizarse, cambiando la forma de hacer las cosas. Generalmente una aplicación web es un sistema que se basa en la arquitectura cliente-servidor y se comunican entre sí mediante el modelo petición-respuesta usando el protocolo HTTP [5].

En la Figura 1.5 se presenta un esquema de la arquitectura cliente-servidor.

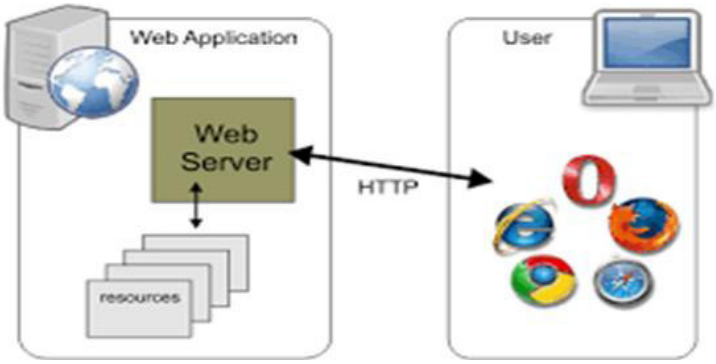


Figura 1.5. Arquitectura cliente-servidor de una aplicación web [5]

1.3.3. PATRONES ARQUITÉCTONICOS UTILIZADOS EN EL DESARROLLO DE APLICACIONES WEB

Un patrón arquitectónico ofrece un diseño de arquitectura estructurada esencial para el desarrollo de una aplicación o software con el objetivo de definir responsabilidades y tareas de cada uno de los componentes, además de definir como se interrelacionan entre sí los componentes mencionados dentro de la aplicación [6].

A continuación, se realizará una revisión del patrón arquitectónico Modelo-Vista-Controlador (MVC) que se utilizará en el presente Trabajo de Titulación.

1.3.3.1. PATRÓN MODELO-VISTA-CONTROLADOR (MVC)

El patrón MVC permite realizar la implementación de una aplicación web dividiendo las tareas y responsabilidades en 3 componentes principales. El concepto del patrón MVC surgió hace muchos años incluso antes de que aparezca la web y en los últimos años su uso se ha incrementado ya que han aparecido en el mercado numerosos *frameworks* para el desarrollo web que emplean este patrón.

MODELO

Este componente se encarga de almacenar, gestionar y actualizar la información que se encuentra ubicada base de datos y que maneja la aplicación web haciendo uso de un enfoque orientado a objetos. En otras palabras, el modelo es el componente encargado de la lógica de los datos. Son responsabilidades del modelo:

- Encapsular el estado de la aplicación.
- Exponer la funcionalidad de la aplicación.
- Notificar los cambios a las vistas.
- Realizar consultas a la base de datos.

Para el desarrollo del modelo, en el presente trabajo de titulación se lo codificará en el lenguaje de programación C# y con la ayuda de Entity Framework los datos se los puede manejar con un enfoque a objetos sin la necesidad de utilizar sentencias escritas en el lenguaje SQL[18], una vez que se codifica las distintas clases que corresponden al modelo, Entity Framework genera el código necesario en SQL Server.

VISTA

La vista es un componente que se encarga de la visualización de la información al usuario mediante una interfaz gráfica, la vista se codificada mediante el lenguaje de marcado HTML. Las vistas en ASP.NET MVC utilizan una sintaxis que es interpretada por el motor Razor, mediante la cual se puede combinar código escrito en el lenguaje de marcado HTML con código escrito en el lenguaje de programación C#. La vista tiene las siguientes responsabilidades:

- Renderizar el modelo.
- Actualizar el modelo con base en la información de las solicitudes.
- Interactuar directamente con el usuario final.

CONTROLADOR

El tercer componente es el controlador que se encarga de la interacción como intermediario entre el Modelo y la Vista, adaptando la información a las necesidades y requerimientos de dichos componentes. El controlador contiene el código C# con cada una de las funcionalidades que necesita la aplicación web. El controlador tiene las siguientes responsabilidades:

- Definir el comportamiento de la aplicación.
- Seleccionar la vista adecuada.

[18] SQL: Lenguaje de Consultas Estructurado, lenguaje utilizado en el manejo de bases de datos.

- Asociar los pedidos de los usuarios con acciones que pueden realizar, por ejemplo, actualizar el modelo.

En la Figura 1.6 se presenta un esquema del patrón MVC.

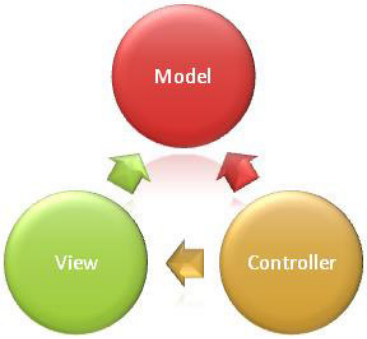


Figura 1.6. Modelo-Vista-Controlador [7]

CICLO DE VIDA DE UNA SOLICITUD HTTP EN ASP.NET CORE MVC

En la Figura 1.7 se presenta un diagrama que representa el flujo que una solicitud HTTP sigue en un servidor IIS con ASP.NET CORE MVC. Inicialmente, se remite la solicitud HTTP a un módulo denominado *routing,* en el cual se analiza la misma y se determina que ruta coincide con la URL del pedido. La ruta asocia una URL y un verbo de HTTP con un controlador y una acción específica de ese controlador. El pedido entonces se remite al controlador, el mismo que debe ejecutar el método que está asociado a la acción específica, este proceso se conoce como *Action Execution* y da como resultado un *Result Execution*, posteriormente el sistema utiliza un motor de búsqueda de vistas denominado *View Engine* que se encarga de encontrar y representar la vista adecuada, finalmente una vez seleccionada la vista se procede a enviar una respuesta HTTP al navegador web del usuario con la vista solicitada [7].

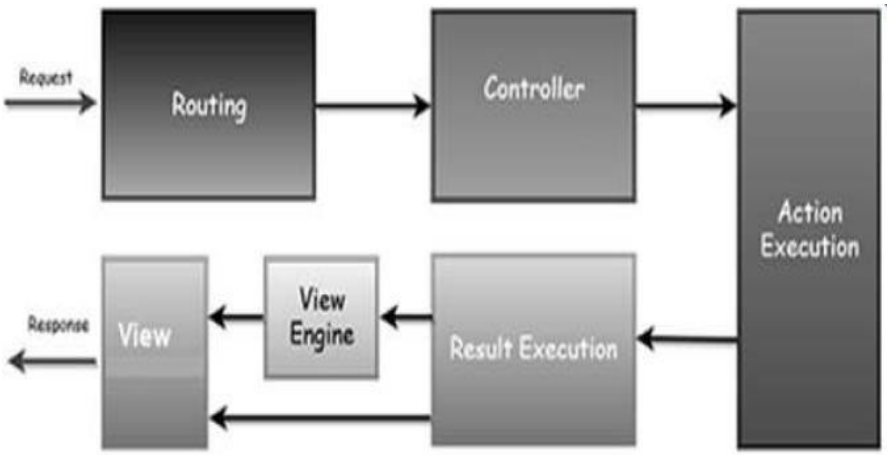


Figura 1.7. Ciclo de vida de una solicitud HTTP con MVC [7]

1.3.4. METODOLOGÍA KANBAN

Es un método que nació en los años 40 de la mano de Toyota. Toyota lo implementó en la producción de sus autos con el objetivo de reducir considerablemente los desperdicios tratando de evitar dejar de ser productivos, brindándole un valor añadido al producto final sin aumentar los costos. A principios del nuevo milenio la metodología Kanban comenzó a aplicarse en el campo de desarrollo de software.

Kanban es una metodología ágil empleada para el desarrollo de software permitiendo gestionar de manera eficaz y eficiente las tareas necesarias para cumplir con un determinado proyecto. Es muy simple y sencilla de usar, además permite enfocar a las tareas desde una parte muy visual teniendo así una idea general del estado del proyecto [8].

Un tablero Kanban básico se encuentra formado por 3 componentes: Tareas Pendientes, Tareas en Proceso y Tareas Hechas facilitando de esta manera el desarrollo de un proyecto, y que a su vez sea más fácil el poder detectar de una manera sencilla y visual las tareas que están generando conflicto dentro del proyecto generando cuellos de botella y a su vez permitiendo buscar soluciones para que el flujo y la carga de trabajo sea continuo con el objetivo de obtener un producto de calidad que sea entregado en el lapso de plazo establecido.

Una metodología ágil se basa en una filosofía de mejora continua que consiste en dividir cada una de las tareas necesarias en el desarrollo de software entre los integrantes del grupo de trabajo con el objetivo de planificar y gestionar de una manera más rápida, pero sobre todo eficiente.

1.3.4.1. PRINCIPIOS DE LA METODOLOGÍA KANBAN

La metodología se basa en los siguientes principios:

1) **Garantizar una alta calidad en el proyecto:** Tareas realizadas con un alto nivel de calidad.
2) **Reducir sustancialmente el desperdicio:** Kanban propone hacer el trabajo justo y necesario, pero que el mismo se lo realice bien y sea entregado dentro de los plazos establecidos.
3) **Mejora continua:** Establecer parámetros que desarrollen y promuevan la mejora continua.
4) **Flexibilidad:** Priorizar ciertas tareas según las necesidades del proyecto en un determinado momento.

1.3.5. TECNOLOGÍAS ADICIONALES QUE SE PUEDEN UTILIZAR EN EL DESARROLLO DE APLICACIONES WEB

1.3.5.1. SISTEMA DE GESTIÓN DE BASE DE DATOS

Un sistema de gestión de base de datos es un software que permite realizar la extracción, el manejo y el almacenamiento de los datos mediante una interfaz que se muestra al usuario permitiendo al mismo un adecuado manejo y control de las bases de datos manteniendo a las mismas seguras, consistentes e íntegras.

Un sistema de gestión de bases de datos se encarga de realizar las siguientes funciones:

- Almacenar los datos.
- Editar los datos.
- Eliminar los datos.
- Manipular los datos.
- Administrar los metadatos.

1.3.5.2. ASOCIACIÓN RELACIONAL DE OBJETOS (ORM)

La asociación relacional permite proporcionar una capa intermedia entre las bases de datos relacionales y un software desarrollado mediante la programación orientada objetos con el objetivo de suprimir el uso de consultas SQL, acelerando el desarrollo de las aplicaciones y reduciendo la complejidad de combinar código con las tablas de una base de datos. Esto hace que los programadores no se preocupen por el uso de sentencias escritas en el lenguaje SQL ya que la implementación, manejo y administración de las bases de datos se los realiza usando el lenguaje orientado a objetos [9].

1.3.5.3. CONTROL DE VERSIONES

Un sistema de control de versiones permite gestionar de manera eficiente la administración de cambios en el código a medida que se desarrolla una aplicación web a lo largo del tiempo, con el objetivo de poder recuperar versiones específicas cuando sea necesario en el futuro. Un control de versiones, además, permite trabajar de manera colaborativa entre múltiples desarrolladores en un mismo proyecto donde cada uno de los mismos guarda una copia del proyecto permitiendo la simultaneidad.

En la Figura 1.8 se presenta un ejemplo del control de versiones, en la cual se pueden apreciar 3 versiones, y en cada una de estas se conoce los cambios que ha sufrido un archivo en particular a medida que se va desarrollando el código.

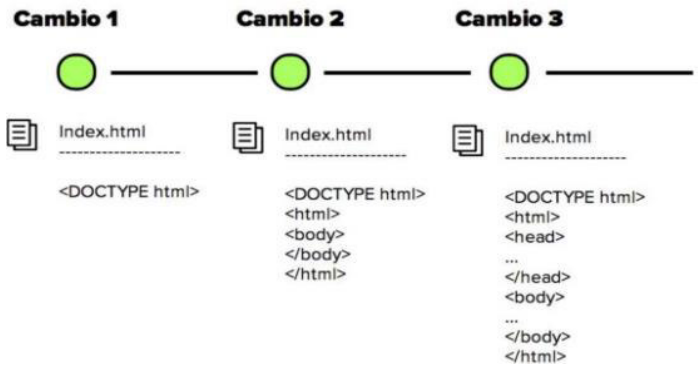


Figura 1.8. Control de versiones [10]

1.3.6. HERRAMIENTAS UTILIZADAS EN EL DESARROLLO DE APLICACIONES WEB

1.3.6.1. ASP .NET CORE

ASP .NET Core es una versión mejorada de ASP .NET Framework. ASP.NET Core es un *framework* de alto rendimiento diseñado y comercializado por Microsoft, utilizado por los programadores para el desarrollo de modernas aplicaciones web dinámicas que se encuentran hospedados en el Internet o en la nube. Los lenguajes de programación que se pueden utilizar en ASP.NET Core son Visual Basic, C# y F#, los cuales se basan en el CLR[19]. ASP.NET Core permite desarrollar aplicaciones que se ejecuten en 3 sistemas operativos diferentes Windows, MacOS o Linux; y el IDE[20] más común que utiliza ASP.NET Core para el desarrollo de aplicaciones web es Visual Studio.

Las aplicaciones web que se desarrollan con ASP.NET Core están completamente orientada a objetos, manejan elementos definidos sean mediante el lenguaje de marcado HTML o un lenguaje de programación como C#, y que, además, utilizan propiedades, métodos y eventos. Visual Studio dispone de diferentes plantillas para implementar aplicaciones web mediante ASP.NET Core de forma sencilla, como: Aplicación web ASP.NET Core, Aplicación web ASP.NET Core utilizando el patrón Modelo-Vista-Controlador (MVC), ASP.NET Core Web API[21], entre otras.

ASP.NET CORE MVC

Es un *framework* que utiliza el patrón arquitectónico Modelo-Vista-Controlador (MVC) con el objetivo de separar y asignar responsabilidades a cada uno de los componentes. Es una

[19] CLR: Entorno en tiempo de ejecución de lenguaje común, entorno de ejecución de .NET.

[20] IDE: Ambiente de desarrollo Integrado, software que cuenta con las herramientas necesarias para el desarrollo de software.

[21] API: Interfaz de programación de aplicaciones, tecnología utilizada para la comunicación en la web.

versión muy utilizada para el desarrollo de aplicaciones web de manera robusta, simplificada, segura y escalable; algunas de las grandes empresas lo utilizan como Microsoft, StackOverflow, Marketwatch, entre otras.

1.3.6.2. PAQUETES NUGET

Para el desarrollo de aplicaciones en las plataformas modernas que existen en la actualidad es necesario utilizar herramientas que permitan crear, compartir y consumir paquetes, mismo que se encuentran formados por líneas de código. En el caso de las plataformas .NET Framework y .NET Core, estas permiten consumir paquetes denominados NuGet dotando a los proyectos de archivos que permiten agregar utilidades y funcionalidades a los mismos, de manera sencilla.

Los paquetes NuGet se encuentran alojados en un repositorio central denominado `nuget.org` que almacena más de 100.000 paquetes únicos que permiten agregar distintas funcionalidades a los proyectos. El proceso de desarrollo de paquetes NuGet se presenta en la Figura 1.9: Un programador crea uno o varios paquetes NuGet que son útiles para el desarrollo de proyectos y los coloca en `nuget.org`, o sitios similares; posteriormente los consumidores buscan los paquetes necesarios para las aplicaciones que se encuentran desarrollando, los descargan e instalan, y los usan en sus aplicaciones [11].

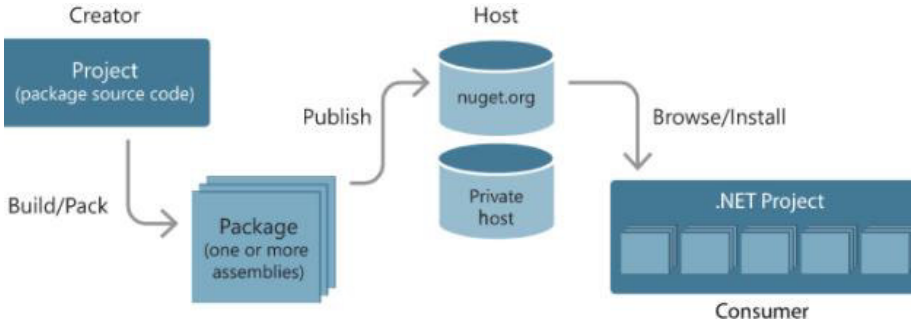


Figura 1.9. Proceso de creación y consumo de paquetes NuGet [11]

1.3.6.3. SQL SERVER MANAGEMENT STUDIO

SQL Server Management Studio (SSMS) es un software desarrollado y comercializado por Microsoft que permite el manejo, la administración y el almacenamiento de bases de datos basadas en el lenguaje SQL. SSMS permite acceder, configurar, gestionar y administrar los módulos de SQL Server para el manejo de bases de datos locales, o remotas mediante Azure SQL, utilizado para el desarrollo de bases de datos en la nube. SSMS es una

plataforma muy completa pero únicamente se ejecuta en computadores que cuenten con un sistema operativo Windows [12].

1.3.6.4. BIBLIOTECA DE CLASES

Una biblioteca de clases .NET Standard permite tener una mayor uniformidad e interoperabilidad en el ecosistema de desarrollo de .NET, al crear bibliotecas de clases reutilizables y que se pueden compartir [13].

.NET Standard se recomienda utilizar en los siguientes escenarios:

- Cuando se requiera tener código compartido que se pueda utilizar desde cualquier plataforma de .NET.
- Para migrar de una manera más fácil entre plataformas por ejemplo migrar una aplicación de .NET Framework a .NET Core.

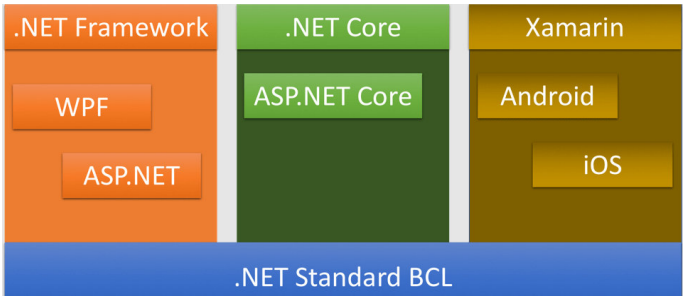


Figura 1.10. .NET Standard y las plataformas de .NET [13]

1.3.6.5. ENTITY FRAMEWORK

Es un ORM utilizado para el desarrollo y la implementación de aplicaciones que utilicen bases de datos relacionales como fuente principal de información, Entity Framework se encuentra disponible para las tecnologías .NET Framework o .NET Core. La implementación de Entity Framework permite tener un mayor nivel abstracción en el manejo y administración de datos, además, permite reducir sustancialmente el número de líneas de código al tratar los datos de una base de datos como objetos, cada uno de estos objetos pertenecen a una clase con sus propias propiedades y atributos dejando de lado la preocupación por el manejo de tablas, columnas y relaciones almacenadas en las bases de datos [14].

En la Figura 1.11 se muestran las 4 opciones que Entity Framework ofrece para gestionar la base de datos en Visual Studio, mientras que en la Tabla 1.1 se presentan las características de cada una de estas opciones.

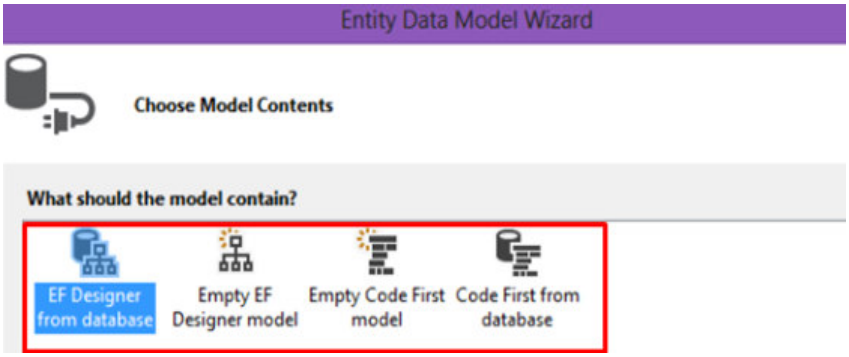


Figura 1.11. Opciones de Entity Framework en Visual Studio

Tabla 1.1. Opciones de Entity Framework

EF Designer desde la base de datos	**EF Designer Modelo vacío**	**EF Modelo Vacío de Code First**	**EF Code First desde la base de datos**
DISEÑO VISUAL		**DISEÑO MEDIANTE CÓDIGO**	
Crea un modelo utilizando EF Designer a partir de una base de datos que ya existe. Se realiza la conexión y configuración con la base de datos y se utilizan clases para interactuar con la aplicación.	Crea un modelo de datos de EF Designer de manera visual definiendo entidades, atributos y relaciones mediante la interfaz gráfica. Luego se creará la base de datos y clases con base al modelo diseñado.	Crea un modelo a partir de las líneas de código, para posteriormente migrar dicho modelo hacia un sistema de gestor de bases de datos como SQL Server Management Studio.	Crea un modelo de datos a partir de líneas de código basado en una base de datos que ya existente. Se puede elegir la conexión y configuración con la base de datos.

En la Figura 1.12 se presenta un ejemplo de EF Designer, como se puede apreciar el modelo se lo construye de una manera muy visual creando entidades con sus respectivos nombres atributos y relaciones de una manera muy similar al diagrama relacional de una base de datos. En tanto que en la Figura 1.13 se presenta un ejemplo de EF Code First, el modelo está compuesto por una clase que cuenta con diferentes tipos de variables, propiedades de tipo `get` y `set`, entre otros elementos, pudiendo trabajar con enfoque orientado a objetos y sin la necesidad de utilizar consultas SQL

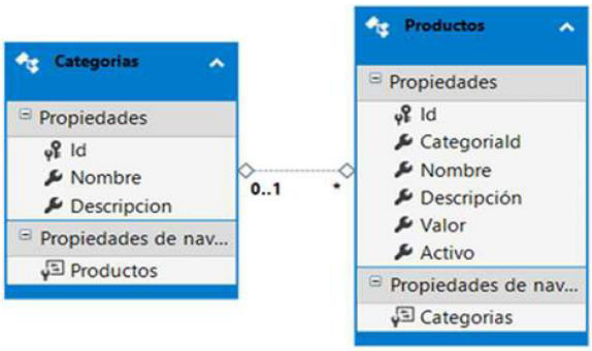


Figura 1.12. EF Designer empleado para diseño visual

```
11      public class OrderDetail
12      {
13          public int Id { get; set; }
14          public Product Product { get; set; }
15
16          public float Quantity { get; set; }
17
18
19          public decimal Price { get; set; }
20
21          [DataType(DataType.MultilineText)]
22          public string Remarks { get; set; }
23          public decimal Value => (decimal)Quantity * Price;
24      }
```

Figura 1.13. EF Code First para diseño con código

1.3.6.6. SCAFFOLDING

Permite realizar acciones CRUD[22] y de esta manera gestionar la información de la base de datos o de modelos definidos de forma simple mediante la generación de código automatizado y se lo ha incluido en diferentes *frameworks* como Django, Laravel, Express Framework, ASP .NET MVC, entre otros. En Visual Studio y para el caso de ASP.NET MVC, dispone de plantillas que permite generar controladores, con sus respectivas vistas y con todos los métodos CRUD para interactuar con la base de datos.

1.3.6.7. GIT Y GITHUB

Git, de acuerdo con [10], es por mucho el sistema de control de versiones más utilizado por los programadores como parte del desarrollo de software. Es un sistema muy moderno y

[22] CRUD *(Create, Read, Update, Delete):* se refiere a operaciones que se realizan en bases de datos para crear, leer, actualizar y eliminar datos.

de código abierto que permite tener una copia de trabajo de cada uno de los desarrolladores del proyecto albergando un historial completo de los cambios y versiones del código fuente de una aplicación. Git cuenta con una arquitectura distribuida que se caracteriza por tener un buen rendimiento, una adecuada seguridad basado en la integridad del código y por ser un sistema muy flexible.

GitHub es un sistema que permite llevar un historial de cambios y versiones del código fuente de una aplicación almacenado en un repositorio de la nube, al estar dicho repositorio en la nube permite a los programadores tener un acceso al historial y las ediciones del código en tiempo real, en cualquier momento y en cualquier lugar. Los repositorios pueden ser públicos para trabajar de manera colaborativa en internet, o privado donde únicamente tengan acceso los desarrolladores que cuenten con los permisos adecuados. GitHub trabaja de manera conjunta con Git.

1.3.6.8. LENGUAJE DE MARCADO DE HIPERTEXTO (HTML)

HTML es el principal lenguaje que se utiliza para el desarrollo de páginas web dinámicas. Es un lenguaje basado en etiquetas que permite colocar párrafos, secciones, listas, enlaces, imágenes y demás contenido que ayude a moldear y dar forma a una página web. El documento HTML está formado por una etiqueta raíz `<html>` y dentro de esta etiqueta se encuentran 2 etiquetas: la etiqueta `<head>` que contiene la información de cabecera de las páginas y la etiqueta `<body>` que contiene el contenido de la página [15].

1.3.6.9. HOJAS DE ESTILO EN CASCADA (CSS)

CSS es un lenguaje básico en el desarrollo de páginas web, es un complemento a HTML y su objetivo es definir la parte de la presentación de la página web, en otras palabras, CSS indica cómo debe alinearse los elementos, su posición, espaciados, forma, colores, entre otros aspectos que forman parte de la página web. CSS está enfocado hacia la parte estética y visual de la página web mediante el uso de los denominados estilos [15].

Los elementos que forman parte del lenguaje CSS son:

- **SELECTORES:** Permiten especificar los elementos de la página.
- **ATRIBUTOS:** Permiten definir las características de los elementos especificados en los selectores.
- **VALORES:** Indica el estilo que se debe aplicar a cada atributo de cada selector. Los valores del estilo con las unidades CSS como píxeles, puntos, etc.

1.3.6.10. JAVASCRIPT (JS)

Es uno de los lenguajes de *front-end* más utilizados en el desarrollo de aplicaciones web. Es un lenguaje que se utiliza mediante secuencias de comandos que permiten agregar muchas más funcionalidades a la presentación y parte visual de la aplicación web como imágenes, mapas interactivos, animaciones 2D-3D interactivas, reproductores de multimedia, audios, entre otros. Es considerada como la tercera tecnología dentro de las tecnologías web de *front-end* por detrás de HTML y CSS [16].

1.3.6.11. JQUERY

JQuery es una librería de código abierto de JavaScript que permite agregar una mejor interactividad en el sitio web. JQuery posee una gran cantidad de *plugins* que permiten estructurar de mejor manera la funcionalidad permitiendo una interacción dinámica con el sitio web [16].

1.3.6.12. JAVASCRIPT ASÍNCRONO Y XML

Asynchronous JavaScript And XML (AJAX) es un conjunto de técnicas utilizadas en el desarrollo web, permite trabajar de manera asíncrona en las aplicaciones procesando información solicitada al servidor en un segundo plano sin la necesidad de recargar la página. Una de las ventajas de utilizar AJAX es mejorar notablemente la experiencia del usuario en el sitio o aplicación web [16].

1.3.6.13. BOOTSTRAP

Bootstrap [17] es el *framework* más utilizado para el desarrollo de *front-end* de aplicaciones web empleando tres tecnologías: HTML, CSS y JavaScript. Es compatible con navegadores web tanto en computadores como dispositivos móviles entre los que encuentran Google Chrome, Mozilla Firefox, Microsoft Edge, Internet Explorer, Opera, Safari, etc. Bootstrap es totalmente gratuito y de código abierto, incluye plantillas en HTML y CSS que facilitan el desarrollo e implementación de formularios, botones, tablas, imágenes y demás elementos necesarios para dar forma al sitio.

1.3.6.14. RAZOR

Es una sintaxis de marcado que se utiliza en el lado del servidor con el objetivo de incrustar código C# o VB.Net dentro de documentos que utilicen otro lenguaje combinando los

mismos. Razor es muy utilizado en ASP.NET MVC ya que permite programar las vistas combinando código escrito en el lenguaje C# con lenguajes de *front-end* como HTML Los archivos que utilizan Razor en el caso de ASP.NET y que emplean código escrito en C# tienen una extensión `.cshtm`, estos archivos, dado que contienen código HTML y código escrito en C# deben ser compilados y ejecutados, lo que permite obtener un documento en HTML que puede ser enviado como respuesta a un pedido HTTP, sin ninguna complicación facilitando por completo la vida del programador.

En la Figura 1.13 se presenta un ejemplo de uso de Razor en una vista, claramente se puede apreciar que en el fragmento presentado se combina código escrito en el lenguaje C# y con código escrito usando el lenguaje de marcado HTML.

```
1  @model IEnumerable<OnSale.Common.Entities.Country>
2  <table>
3      <thead>
4          <tr>
5              <th>Id</th>
6              <th>Nombre</th>
7              <th></th>
8          </tr>
9      </thead>
10     <tbody>
11         @foreach (var item in Model)
12         {
13         <tr>
14             <td>@item.Id</td>
15             <td>@item.Name</td>
16             <td>
17             @Html.ActionLink("Countries", "Edit", new { Id = item.Id })
18             @Html.ActionLink("Countries", "Details", new { Id = item.Id })
19             @Html.ActionLink("Countries", "Delete", new { Id = item.Id })
20             </td>
21         </tr>
22
23         }
24     </tbody>
25 </table>
```

Figura 1.13. Ejemplo de uso de Razor

En el ejemplo en concreto se aprecia la creación de una tabla con base en la información del modelo (línea 1), mediante las etiquetas `<th>` definidas dentro de la etiqueta `<thead>` (línea 3 - línea 9) se definen los títulos de las columnas, para el `<tbody>` se emplea un lazo `foreach` (línea 11 a línea 23) característico del lenguaje C# para iterar los datos que pertenecen al modelo, renderizando esta información en la tabla construida a partir de la combinación de lenguaje HTML con lenguaje C#.

1.3.6.15. TRELLO

Es una aplicación disponible en la web y en dispositivos móviles que permite gestionar de una manera efectiva y adecuada un proyecto a nivel profesional o personal. Es ideal para utilizarlo en el desarrollo de software ya que permite implementar la metodología Kanban

de una manera muy visual. Trello permite crear un tablero con tarjetas que describan cada una de las tareas necesarias del proyecto, dicho tablero contiene al menos 3 columnas con las tareas: Pendientes, en Proceso y Hechas [18].

1.3.7. RELACIÓN CON TRABAJOS ANTERIORES DEL ÁREA

El presente Trabajo de Titulación se encuentra relacionado con el proyecto de titulación titulado" IMPLEMENTACIÓN DE UN PROTOTIPO PARA MANEJO DE MENÚ EN RESTAURANTES USANDO TECNOLOGÍA INALÁMBRICA ZIGBEE"

Este trabajo se encuentra enfocado a trabajar en las crecientes aplicaciones de automatización en hogares y edificios de uso terciario; y tiene como principal objetivo ofrecer una solución mediante la implementación de un prototipo que utiliza tecnología ZigBee para poder realizar la toma interactiva de órdenes que realizan los clientes en un restaurante.

La principal diferencia que existe con el trabajo anteriormente mencionado radica en la tecnología que se emplea, la implementación del prototipo que realiza la toma de órdenes y manejo de menús del restaurante se lo desarrolla utilizando el protocolo de comunicación inalámbrica denominado ZigBee. En el presente trabajo de titulación se desarrollará un prototipo de aplicación web utilizando ASP.NET Core implementando el patrón MVC (Modelo-Vista-Controlador) para que los usuarios puedan acceder a la misma por medio de cualquier navegador web, todo esto estará enfocado a los requerimientos del restaurante "Don Ruiz".

2. METODOLOGÍA

En el presente capítulo, se detallan los requerimientos funcionales y no funcionales del prototipo obtenidos por medio de entrevistas a los propietarios y empleados del restaurante, con la información obtenida se procedió a realizar las historias de usuario y a generar un tablero Kanban que contiene el *backlog* de tareas necesarias para el desarrollo del prototipo y que está conformado por 3 columnas: Tareas Pendientes, Tareas En Proceso y Tareas Hechas.

Posteriormente, como parte de la fase diseño del prototipo se explica la elaboración de los diagramas de casos de uso, el diagrama relacional de la base de datos, el diagrama de clases y los *sketches* de las vistas con las que contará el prototipo.

Después, se detalla la implementación del prototipo para el cual se explica la creación del repositorio en GitHub y su clonación en un repositorio Git local, luego, se realiza la implementación del ambiente de desarrollo por medio de Visual Studio 2019 con todos los componentes, dependencias y librerías necesarias para el desarrollo del prototipo.

Finalmente se realiza la explicación de la codificación del modelo y su migración hacia el gestor de base de datos SQL Server Management Studio, además se revisa la codificación de los controladores que permiten dar el manejo a la lógica del negocio y la codificación de las vistas con las interfaces de usuario con las que cuenta el prototipo.

2.1. DISEÑO DE PROTOTIPO

2.1.1. HISTORIAS DE USUARIO

Una historia de usuario permite tener una descripción general e informal por parte del usuario final o cliente acerca de la funcionalidad y requerimientos que debe tener la aplicación o software que se plantea desarrollar [19].

Una vez realizada la entrevista a los 2 propietarios y 3 empleados del restaurante se procedió a elaborar cada una de las historias de usuario utilizando una plantilla obtenida en [20]. Cada historia de usuario posee un identificador único que permite distinguirla y tiene el siguiente formato `UH<Número de historia>-<Tipo de Usuario>`, además, cada historia de usuario posee un nombre, un rol de usuario, la funcionalidad, el resultado esperado y detalles adicionales.

En la Figura 2.1 se presenta un ejemplo de historia de usuario asociada con los usuarios en general, es decir funcionalidades que compartirán todos los usuarios de la aplicación

sin excepción alguna. En la Figura 2.2 se muestra un ejemplo de historia de usuario asociada al usuario administrador. En la Figura 2.3 se muestra un ejemplo de historia de usuario asociada al usuario cliente. En el anexo A se incluyen todas las historias de usuario.

Id: UH001-USR	
Nombre: Inicio de Sesión	
Cómo	Usuario
Puedo	Iniciar sesión utilizando un nombre de usuario y una contraseña.
Para	Ingresar al sistema.
Detalles	
La contraseña debe ser fuerte.	
Debe permitir modificar la contraseña.	
Debe mostrar un mensaje de error si el usuario o la contraseña es incorrecta.	

Figura 2.1. Historia de usuario asociado a todos los usuarios en general

Id: UH001-ADM	
Nombre Historia: Gestión de los productos del menú.	
Cómo	Administrador
Puedo	Realizar el CRUD de los productos del menú.
Para	Realizar una gestión del menú.
Detalles	
Cada producto debe contener una imagen de este.	
Cada producto debe tener información como el nombre, el precio, descripción, imagen, tipo de producto, entre otros.	

Figura 2.2. Ejemplo de historia de usuario administrador

Id: UH001-CLI	
Nombre: Registrar Cliente	
Cómo	Cliente
Puedo	Registrarme en la aplicación.
Para	Poder ingresar al sistema.
Detalles	
Se debe ingresar un correo válido y existente.	
Cada usuario debe tener información como el nombre, apellido, cédula, teléfono, dirección, imagen, entre otros.	

Figura 2.3. Ejemplo de historia de usuario cliente

2.1.2. REQUERIMIENTOS

Una vez que se realizaron las diferentes historias de usuario, se procedió a establecer los siguientes requerimientos funcionales:

- El prototipo tendrá un mecanismo de autenticación de usuarios utilizando como credenciales un correo válido y una contraseña.

- En el prototipo todos los usuarios serán capaces de cerrar sesión cuando lo requieran.
- En el prototipo se implementarán 3 roles de usuarios: Administrador, Empleado y Cliente.
- El prototipo permitirá al usuario administrador realizar el CRUD de empleados y administradores.
- El prototipo contará con un mecanismo de registro para clientes mediante el uso de un formulario.
- Cuando se registren nuevos usuarios el prototipo enviará un correo de confirmación a la dirección de correo electrónico registrado para garantizar que el mismo exista y sea válido.
- Cuando se registre un nuevo usuario empleado o administrador, el prototipo enviará las credenciales de usuario en el correo de confirmación.
- En el prototipo, la primera vez que inicie sesión un usuario empleado o administrador, se solicitará un cambio de contraseña por seguridad.
- El prototipo solicitará el ingreso de contraseñas fuertes por parte de los usuarios.
- El prototipo contará con un mecanismo que permita recuperar la contraseña en caso de olvido o pérdida de esta.
- El prototipo tendrá un mecanismo que permita cambiar la contraseña y los datos personales cuando el usuario lo requiera.
- El usuario administrador será capaz de realizar el CRUD de productos del menú y del menú del día que ofrece el restaurante.
- El prototipo contará con un mecanismo que permita ingresar una imagen para presentar a los usuarios los productos del menú.
- El usuario cliente podrá ver el menú que ofrece el restaurante.
- El prototipo dispondrá de un mecanismo que permita a los usuarios visualizar productos del menú.
- Se manejarán 5 estados para gestionar el pedido: pendiente, gestionando, entregado, cobro y pagado.
- El prototipo permitirá al usuario cliente realizar el pedido asociando al mismo el tipo de pedido (mesa, llevar) y el número mesa.
- Los usuarios empleados y administradores podrán gestionar los pedidos ingresados.
- El cliente será capaz de solicitar la cuenta cuando lo requiera.
- El prototipo contará con un mecanismo que permita a los usuarios visualizar el estado de los pedidos y su detalle.

- El prototipo tendrá un mecanismo para que un usuario administrador pueda confirmar el pago de las cuentas de los clientes.
- El prototipo deberá permitir al administrador ver las ventas del día o el de una fecha específica.
- El administrador gestionará las mesas, entendiendo por gestionar a crear o eliminar una mesa.
- Los usuarios podrán ver el estado de las mesas para ver si están ocupadas o libres.
- Todas las listas del prototipo contarán con un cuadro de búsqueda para facilitar el filtrado de información.

Los requerimientos no funcionales son:

- Las interfaces de usuario deberán funcionar adecuadamente en al menos dos navegadores web.
- El prototipo tendrá una base de datos generada en SQL Server.
- El prototipo se lo construirá utilizando ASP .NET Core MVC.

2.1.3. BACKLOG DE TAREAS Y TABLERO KANBAN

Con base a los requerimientos levantados en el apartado anterior, en la Tabla 2.1 se muestran todas las tareas necesarias para implementar *backlog*.

Tabla 2.1. Tareas definidas en el *backlog*

Tarea
Implementar un mecanismo de autenticación en el sistema.
Implementar un mecanismo de validación de usuarios por medio de correo electrónico.
Implementar el CRUD de empleados y administradores en el sistema.
Implementar el CRUD de clientes en el sistema.
Implementar un mecanismo de recuperación de contraseña de usuarios.
Implementar un mecanismo para el manejo de contraseñas seguras.
Implementar un mecanismo que permita el cambio de la contraseña de usuario.
Implementar el CRUD de los productos que ofrece el restaurante.
Implementar un mecanismo que permita gestionar el menú del día.
Implementar un mecanismo para gestión de las mesas del restaurante.
Implementar el mecanismo de gestión de los pedidos en el sistema.
Implementar un mecanismo que despliegue la información de las ventas.
Implementar un mecanismo de acceso no autorizado y de errores en el sistema.

Una vez especificado el *backlog* de tareas se procedió a estructurar el tablero Kanban del prototipo con las 3 columnas correspondientes Tareas Pendientes, Tareas en Proceso y Tareas Hechas como se lo muestra en la Figura 2.4.

Para la elaboración del tablero Kanban se utilizó la aplicación denominada Trello mencionada en la sección 1.3.6.15.

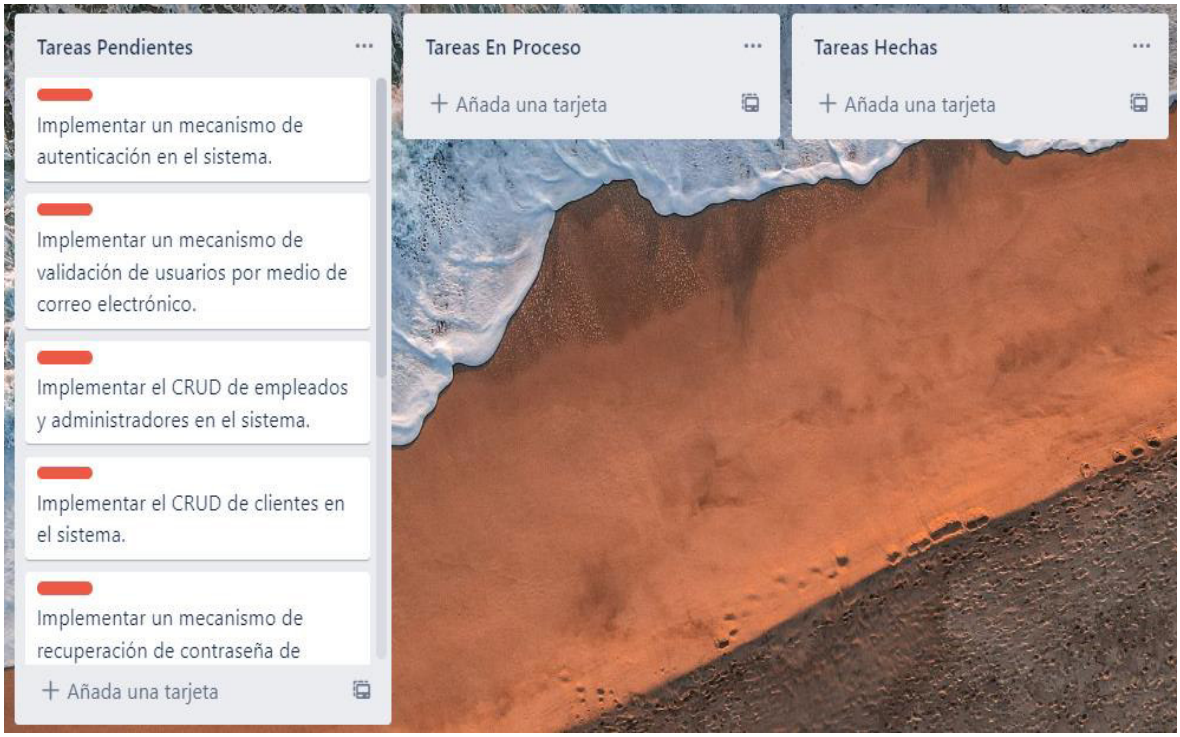


Figura 2.4. Tablero Kanban

2.1.4. ARQUITECTURA DEL PROTOTIPO

El prototipo contará con una arquitectura cliente-servidor utilizando el patrón arquitectónico Modelo-Vista-Controlador. El prototipo se encuentra conformado por los siguientes componentes:

- **Base de datos:** Es el componente que se encarga de almacenar la información del prototipo.
- **Modelo:** Es el componente que brinda el acceso a la información de la base de datos para su manipulación.
- **Controlador:** Es el componente encargado de establecer la lógica del negocio para el funcionamiento adecuado del prototipo.
- **Vista:** Es el componente que renderiza la información obtenida del modelo con el objetivo de mostrarla al usuario final en un navegador web.

En la Figura 2.5 se muestra la arquitectura del prototipo, donde se puede apreciar que el cliente interactúa con el servidor por medio de peticiones y respuestas basadas en el protocolo HTTP, el controlador recibe las peticiones, solicita datos al modelo y recupera la vista con la información renderizada para enviársela al cliente.

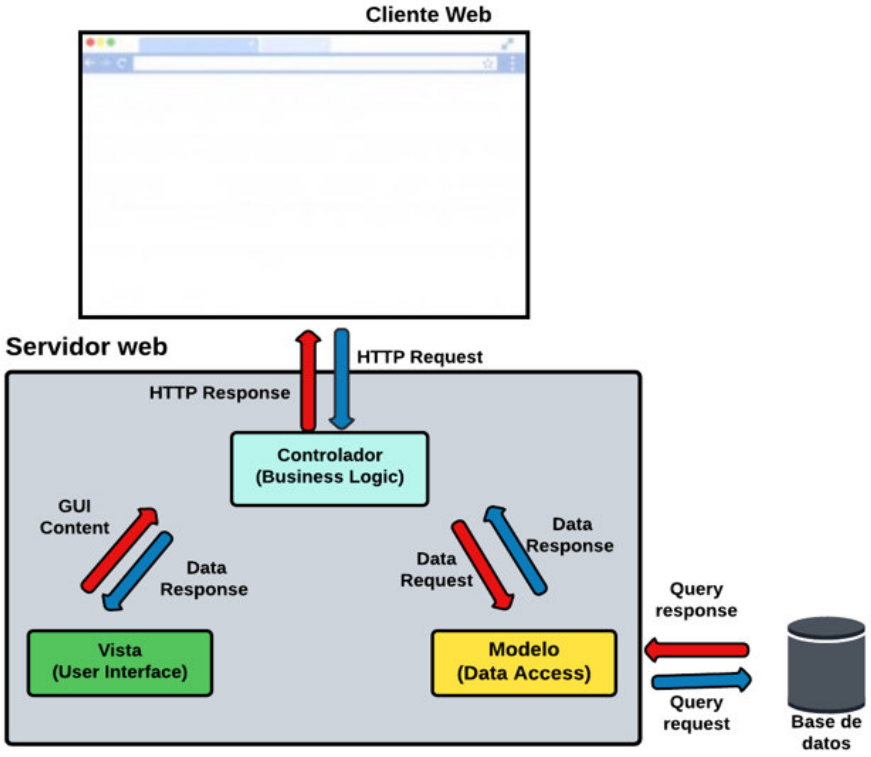


Figura 2.5. Arquitectura del proyecto

2.1.5. DIAGRAMA DE CASOS DE USO

Una vez realizadas las historias de usuario y levantados los requerimientos se han identificado 3 actores que se muestran en la Tabla 2.2 y que van a interactuar con el prototipo que se plantea desarrollar.

Tabla 2.2. Actores del sistema

Actor	Descripción
Administrador	Es el actor encargado de gestionar el menú, gestionar los empleados, gestionar los cobros, visualizar las ventas entre otras actividades. Es el actor que posee mayores responsabilidades en el sistema.
Empleado	Es el actor encargado de gestionar los pedidos ingresados en el sistema.
Cliente	Es el actor encargado de registrarse en el sistema, de realizar el pedido y solicitar los cobros cuando lo requiera.

En la Figura 2.6 se muestra el diagrama de casos de uso, donde se presentan las interacciones de todos los actores con el prototipo.

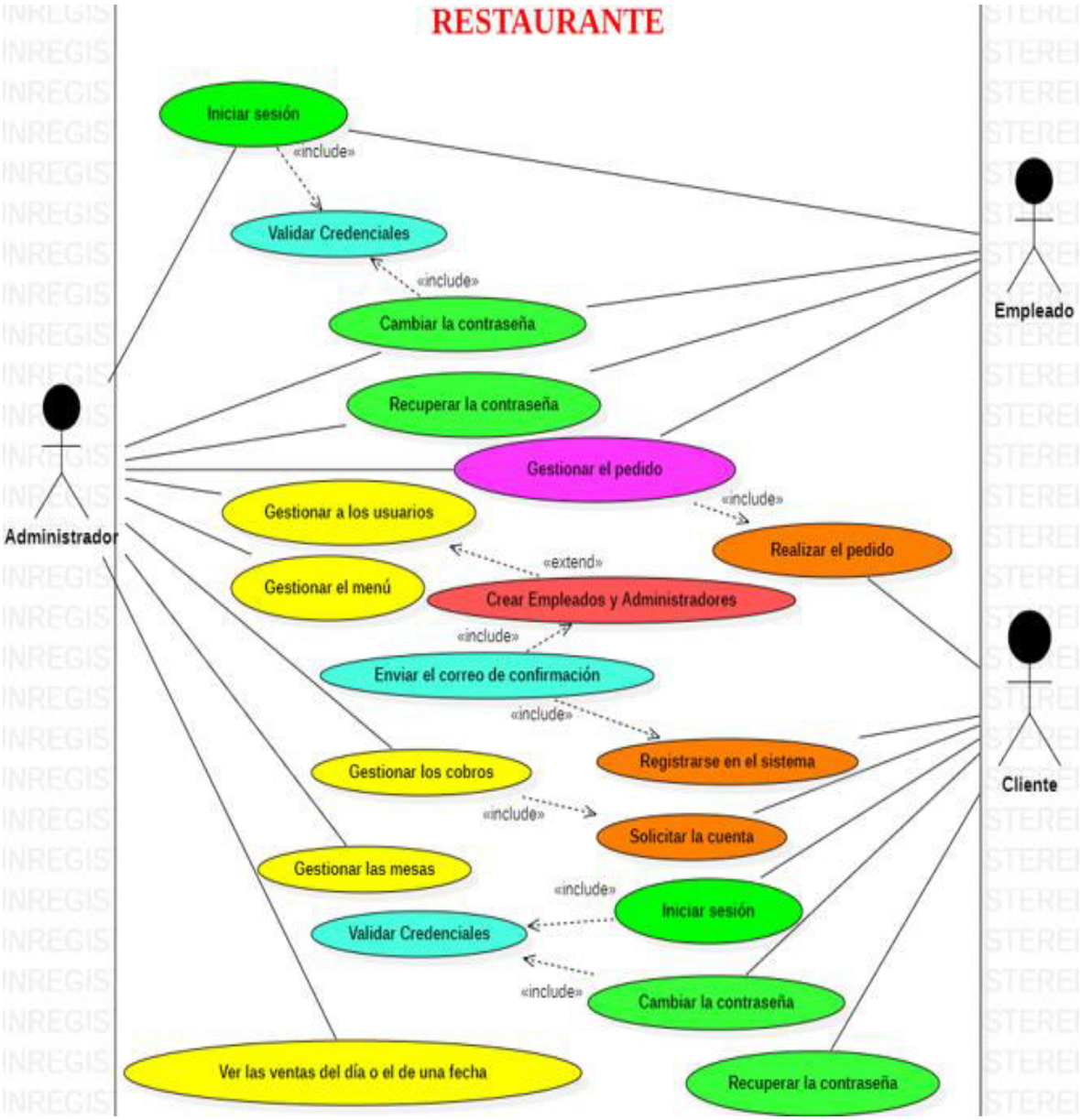


Figura 2.6. Diagrama de casos de uso

2.1.6. DISEÑO DE LA BASE DE DATOS

Para realizar el diseño de la base de datos se tomó como punto de partida toda la información recopilada en las entrevistas, las historias de usuario y los requerimientos del prototipo.

En la Figura 2.7 y Figura 2.8 se muestra el diagrama relacional de la base de datos con cada una de las tablas y sus respectivos atributos.

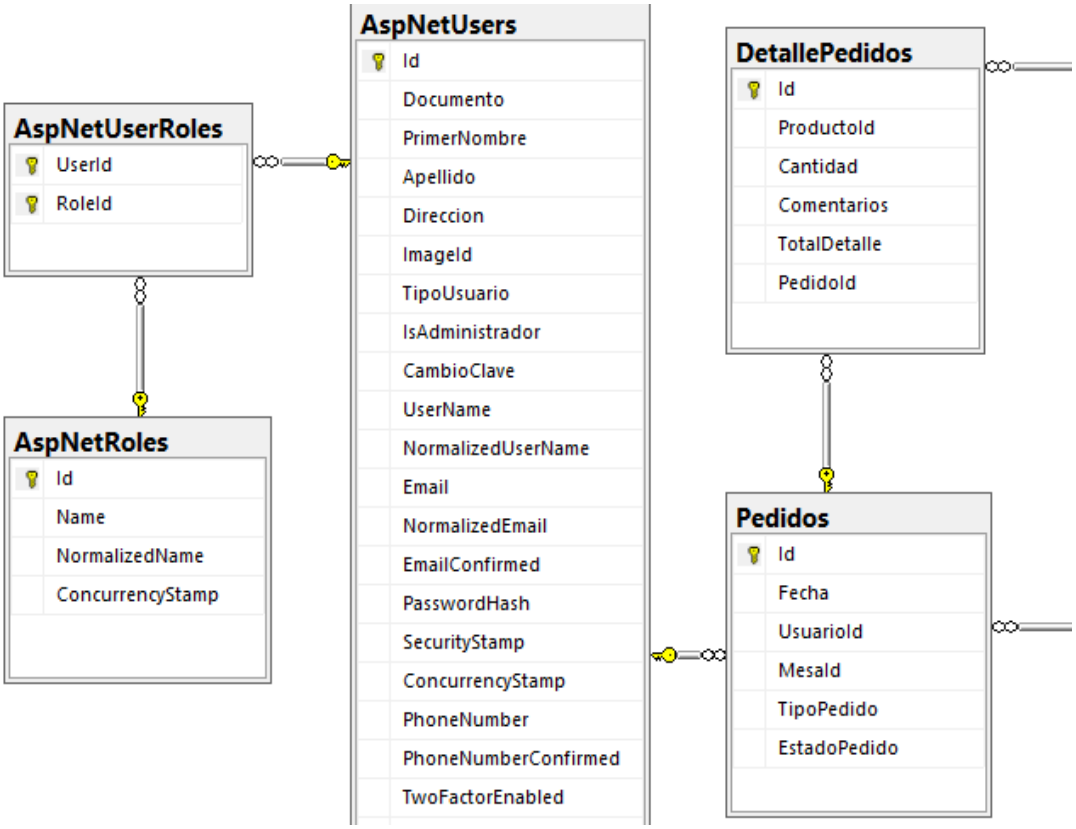


Figura 2.7. Diagrama relacional de la base de datos parte I

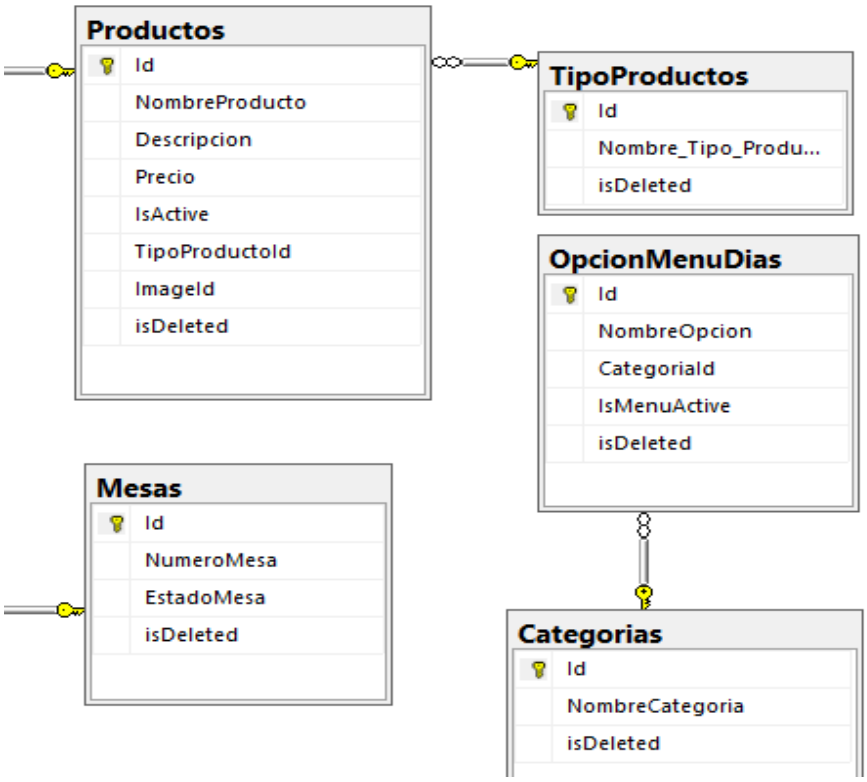


Figura 2.8. Diagrama relacional de la base de datos parte II

La tabla `Productos` contiene toda la información del producto como el nombre, descripción, precio, identificador de la imagen, entre otros y se encuentra relacionado la tabla `TipoProductos`.

La tabla `DetallePedidos` se relaciona con la tabla `Productos` y con la tabla `Pedidos` y se la utiliza para poder obtener la información del detalle de cada producto que se ingresa en un pedido. La tabla `Pedidos` a su vez, se relaciona con la tabla `AspNetUsers` y con la tabla `Mesas`, ya que cada pedido debe tener asociado un usuario y una mesa.

La tabla `AspNetUsers` se relaciona con la tabla `AspNetUserRoles` y la tabla `AspNetRoles`, las mismas que se generan automáticamente al utilizar la librería ASP.NET Identity, la tabla `AspNetUsers` facilita la implementación de procesos de autenticación, administración de usuarios, manejo de contraseñas, roles de usuario, datos de usuario, confirmación por correo electrónico, entre otras, para el desarrollo de aplicaciones basadas en .NET [21].

La tabla `OpcionMenuDias` contiene información acerca de las opciones que el restaurante maneja para ofrecer el menú del día (almuerzos y meriendas) y se relaciona con la tabla `Categorias`.

Al utilizar una base de datos relacional algunas tablas poseen el atributo `IsDeleted` del tipo `bool` que permite dar un manejo adecuado a la eliminación de datos y evitar un daño en la integridad referencial de la base de datos.

2.1.7. DIAGRAMA DE CLASES

Para el diseño del diagrama de clases, fue necesario dividirlo en dos partes: la primera parte corresponde a las clases asociadas a las tablas de la base de datos y la segunda parte corresponden a las clases ViewModel que facilitan el manejo de la información que se encuentra almacenada en la base de datos y visualización de la misma en cada una de las vistas.

De acuerdo con [22], las clases denominadas ViewModel contienen atributos que se utilizan para poder renderizar la información en las vistas de una manera más sencilla. Las clases ViewModel no interactúan directamente con la base de datos, pero se combinan con el modelo de para poder visualizar, gestionar y manejar la información. Además, permiten combinar uno o más modelos en una sola clase ViewModel, tal como se lo muestra en la Figura 2.9.

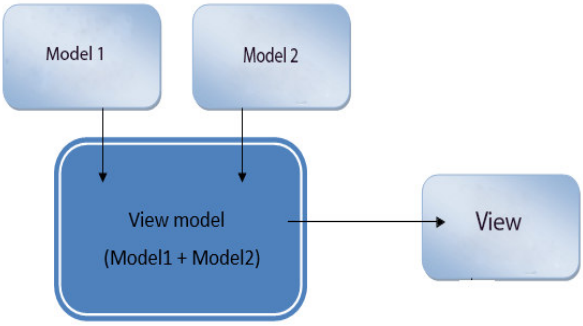


Figura 2.9. Relación del Modelo-ViewModel-Vista *[22]*

En la Figura 2.10 y la Figura 2.11 se encuentra el diagrama de clases asociados a las tablas de la base de datos para el cuál se utilizó Entity Framework con el enfoque Designer, y como se puede visualizar el diagrama posee clases y se relaciona de manera similar al diagrama de la base de datos de la Figura 2.7 y la Figura 2.8 con la diferencia de que incorpora algunas enumeraciones, como `TipoPedido` y `EstadoPedido` que permiten representar valores constantes de una manera sencilla y ambas clases se emplean en la clase `Pedido`.

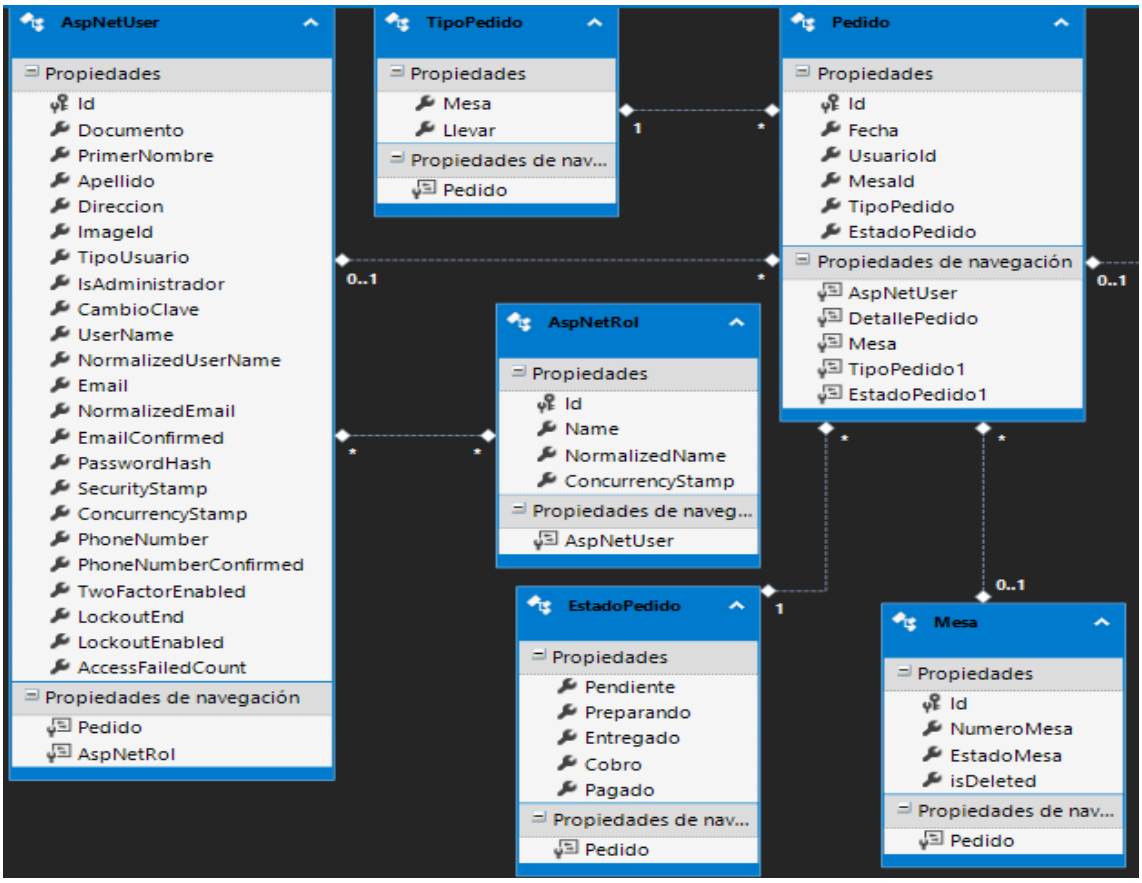


Figura 2.10. Diagrama de clases generado por Entity Framework con el enfoque Designer parte I

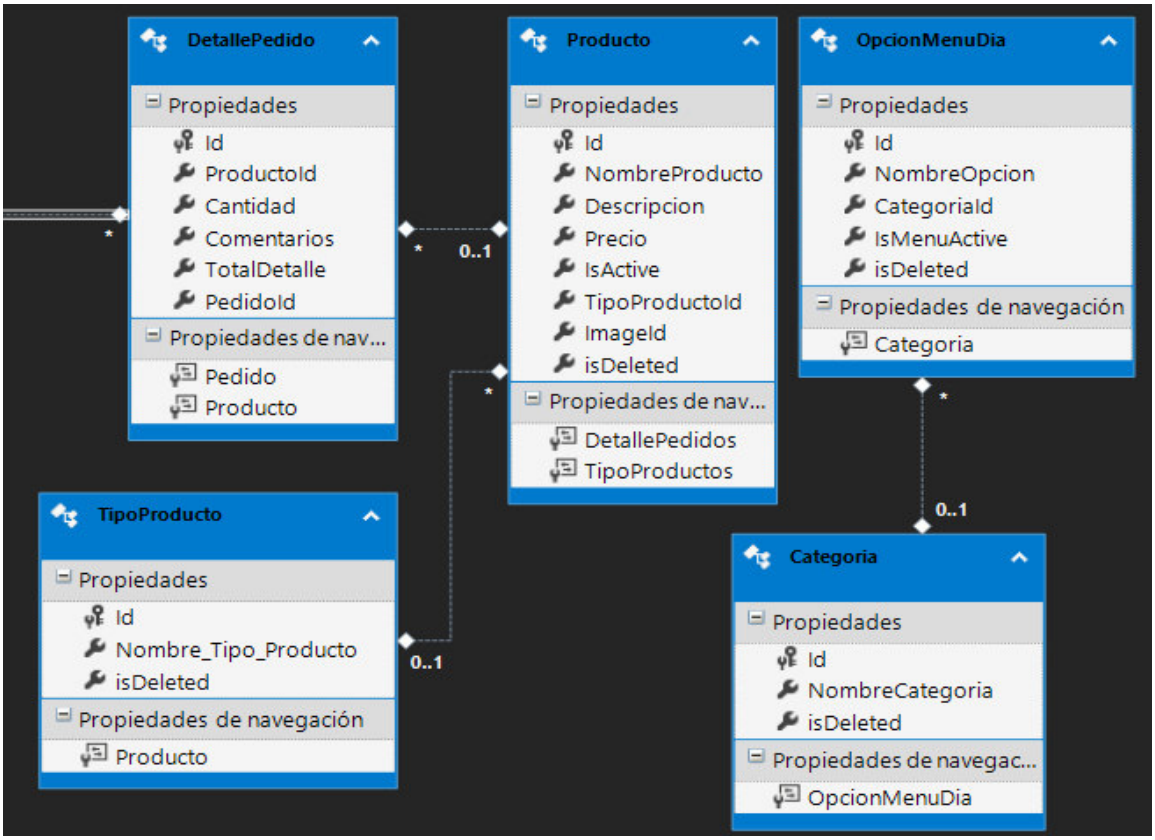


Figura 2.11. Diagrama de clases generado por Entity Framework con el enfoque Designer parte II

En la Figura 2.12 y la Figura 2.13 se muestra el diagrama de clases ViewModel, para poder diferenciarlas todas las clases utilizan el sufijo ViewModel. Además, se utilizan algunas clases asociadas a las tablas de la base de datos para poder entender de mejor manera como se encuentran relacionadas con las clases ViewModel.

La clase `EditUserViewModel` se utiliza para gestionar la información de los usuarios y poder realizar el CRUD de los mismos, la clase `AddUserViewModel` hereda de la clase `EditUserViewModel` y se la utiliza para la creación de las credenciales de usuario que permitan su autenticación en el prototipo. La clase `RecoverPasswordViewModel` en conjunto con la clase `ResetPasswordViewModel` se utilizan para poder recuperar la contraseña de usuario en caso de olvido o pérdida de esta. La clase `LoginViewModel` se utiliza para poder realizar la autenticación de usuario y se relaciona con la clase `ResetPasswordViewModel` y con la clase `ChangePasswordViewModel` que se utiliza para modificar la contraseña cuando el usuario lo requiera.

La clase `ProductoViewModel` hereda de la clase `Producto` y se la utiliza para generar una lista desplegable con los datos de la clase `TipoProducto`, así como para cargar la imagen del producto y para el manejo adecuado del precio. Las clases

`PedidoViewModel`, `DetalleViewModel` y `ProductoPedidoViewModel` permiten recopilar la información de los pedidos de los clientes. La clase `VentasViewModel` se utiliza para poder mostrar la información de las ventas del día o el de una fecha específica y se relaciona con la clase `Producto` y con la clase `Pedido`.

La clase `OpcionMenuDiaViewModel` se utiliza para poder generar una lista desplegable de las categorías. La clase `ErrorViewModel` se utiliza para dar manejo de errores HTTP como el error 404 o de acceso no autorizado.

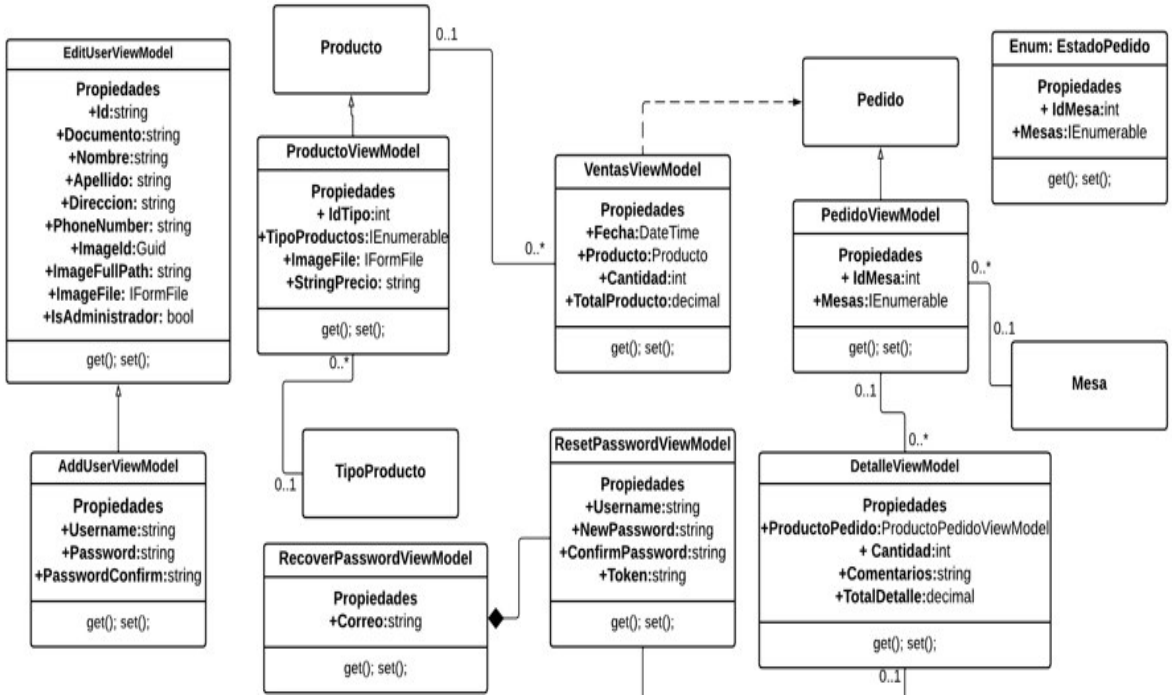


Figura 2.12. Diagrama de clases ViewModel parte I

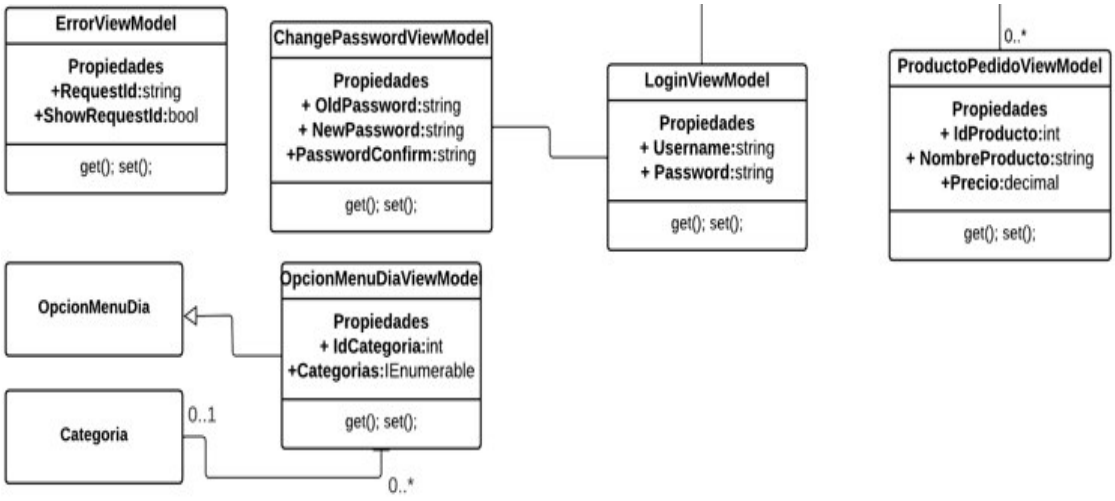


Figura 2.13. Diagrama de clases ViewModel parte II

2.1.8. SKETCHES DE LAS VISTAS

Los *sketches* son bocetos no funcionales que muestran una idea general del diseño de las principales interfaces de usuario con las que contará el prototipo. Los *sketches* tendrán elementos como botones, listas, cuadros de búsqueda, textos, panel de navegación, entre otros elementos que ayudarán a mostrar la estructura de la parte visual del prototipo.

Todas las vistas se encuentran en el Anexo B. En esta sección se realizará una breve descripción de algunos *sketches*.

En la Figura 2.14 se muestra el *sketch* que representa la vista de autenticación del sistema, contiene campos para ingresar un usuario y una contraseña, información requerida para poder acceder al sistema. Además, contiene botones para ingresar al sistema o para registrarse en el mismo en caso de no ser cliente.

Figura 2.14. *Sketch* de la vista inicio de sesión

En la Figura 2.15 se muestra el *sketch* para agregar un nuevo producto; dispone de campos para ingresar la información y con botones para guardar o regresar.

Figura 2.15. *Sketch* de la vista para agregar un producto

En la Figura 2.16 se presenta el *sketch* de la interfaz para gestión del menú, el mismo que tiene información del menú, además cuenta con botones para realizar las acciones CRUD, así como con un buscador para poder hacer el filtrado de información.

Figura 2.16. *Sketch* de la vista del menú del lado administrador

En la Figura 2.17 se presenta el *sketch* de la vista con el detalle del pedido, se puede apreciar que presentará toda la información del pedido, y contará con botones para poder realizar la gestión del pedido.

Figura 2.17. *Sketch* de la vista con el detalle del pedido

2.2. IMPLEMENTACIÓN DEL PROTOTIPO

2.2.1. CONTROL DE VERSIONES

Para realizar el control de versiones se creó un repositorio online en GitHub. En la Figura 2.18 se muestra el repositorio creado en GitHub para el presente Trabajo de Titulación. Una vez creado el repositorio fue necesario clonar el mismo por medio de Visual Studio en un repositorio Git local en el equipo que se está desarrollando el prototipo.

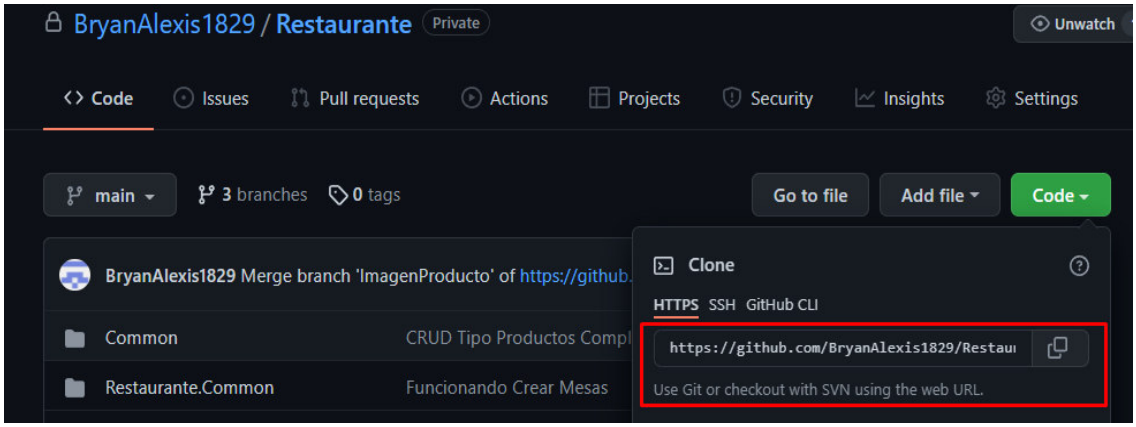


Figura 2.18. Ejemplo de repositorio en GitHub

Para realizar la clonación se debe vincular la URL del repositorio de GitHub con el repositorio Git local que se va a crear, tal como se lo muestra en la Figura 2.19.

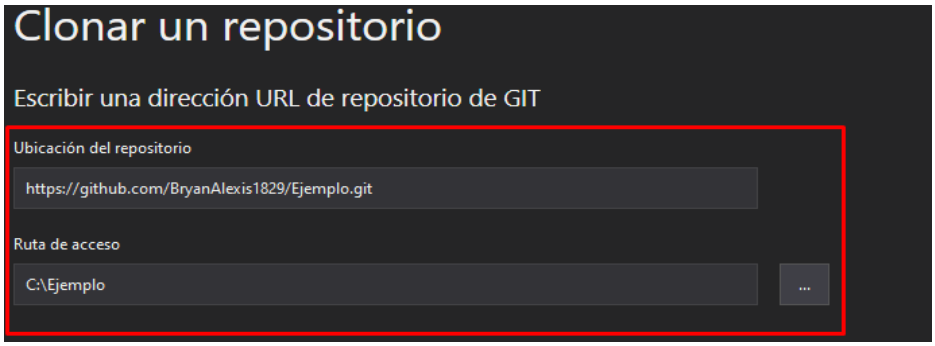


Figura 2.19. Clonar un repositorio en Visual Studio

La utilización de un sistema de control de versiones permite hacer un control adecuado de los cambios implementados en el código a medida que se desarrolla el prototipo tanto en el repositorio local como el repositorio remoto.

Visual Studio permite realizar operaciones Git de una manera visual a medida que se van realizando cambios en el código y guardarlos en el historial de versiones, entre las principales operaciones Git que se utilizaron para el desarrollo del prototipo se encuentran:

- **`Git commit:`** Se lo utilizó cada vez que se llegó a una versión estable del prototipo para crear una versión nueva añadiendo la misma al historial del repositorio local con todos los cambios y modificaciones efectuados en el código fuente.
- **`Git push:`** Se lo utilizó para cargar la información del repositorio local en el repositorio remoto de GitHub.

- **`Git revert:`** Se lo utilizó para regresar a versiones anteriores del código cuando fue necesario.

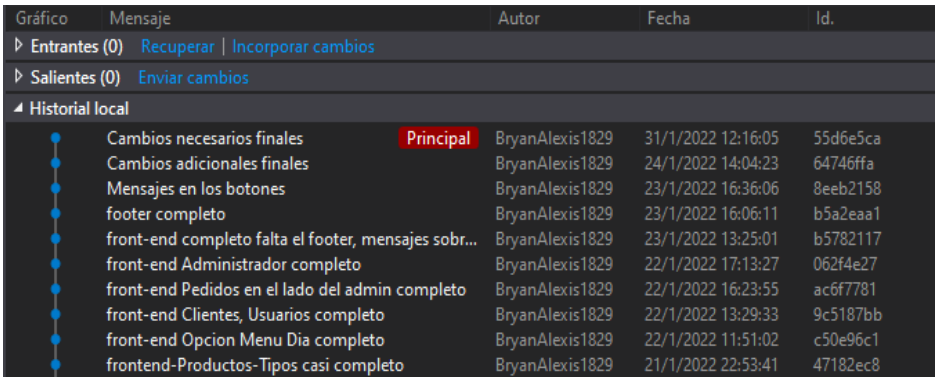


Figura 2.20. Parte del historial de versiones local del prototipo

2.2.2. ESTRUCTURA DEL CÓDIGO DEL PROYECTO

En Visual Studio se tienen soluciones que son contenedores de proyectos, y se tienen proyectos. Los proyectos incluyen los elementos necesarios para compilar la aplicación, como archivos de código fuente, mapas de bits, iconos y referencias de componentes y servicios. Para estructurar el proyecto en Visual Studio se creó una solución vacía denominada `Restaurante` que contiene dos proyectos, el primero es una biblioteca de clases .NET Standard denominada `Restaurante.Common` que contiene las siguientes carpetas:

`Dependencias`: Contiene los paquetes NuGet incorporados al proyecto.

`Entidades`: Contiene algunas clases de las entidades propuestas en el diagrama relacional de la Figura 2.7 y Figura 2.8.

`Enumeraciones`: Contiene enumeraciones que permiten representar valores constantes que se utilizan en el proyecto.

`Extensiones`: Contiene una clase para validar las extensiones de los archivos de las imágenes que se cargan en el prototipo.

`Respuestas`: Contiene una clase para el manejo del envío del correo de confirmación al registrar o crear un nuevo usuario.

El segundo es el proyecto web ASP .NET Core MVC que se denomina `Restaurante.Web` y contiene las siguientes carpetas y archivos:

`Dependencias`: Contiene los paquetes NuGet incorporados al proyecto.

`wwwroot`: Carpeta creada automáticamente por ASP .NET Core MVC y contiene subcarpetas denominadas CSS, JavaScript e imágenes con archivos de imágenes que se cargan automáticamente, además, contiene las librerías de Jquery y Bootstrap del proyecto.

`Controllers`: Carpeta creada automáticamente por ASP .NET Core MVC para almacenar todos los archivos de los controladores.

`Datos`: Carpeta que contiene la clase que permite la conexión con la base de datos, el *Seeder* para cargar la información por defecto y una subcarpeta denominada `Entidades` que contiene algunas entidades del modelo.

`Helpers`: Contiene interfaces y clases que permiten realizar la inyección de dependencias[23] en cada uno de los controladores en el prototipo.

`Migrations`: Creada automáticamente por Entity Framework y contiene el historial de migraciones del modelo.

`Models`: Carpeta creada automáticamente por ASP .NET Core MVC y se utiliza para guardar las clases ViewModel utilizadas en el proyecto.

`Views`: Es una carpeta creada automáticamente por ASP .NET Core MVC contiene todas las vistas del proyecto.

`Program.cs`: Es un archivo que contiene el código de inicio de la aplicación es creado automáticamente por ASP .NET Core MVC.

`Startup.cs`: Es un archivo que contiene los contenedores de servicios y de configuración de la aplicación.

`appsettings.json`: Es el archivo principal de configuración de la aplicación que se utiliza para la comunicación con la base de datos, el correo, entre otros servicios. Tiene formato JSON[24] y es creado automáticamente por ASP .NET Core MVC.

`libman.json`: Es un archivo en formato JSON y se utiliza para agregar los archivos que referencian a un librería, es creado automáticamente por ASP .NET Core MVC.

[23] Inyección de dependencias: Es una técnica orientada a disminuir el acoplamiento entre los componentes de una aplicación mediante el uso de interfaces.

[24] JSON: Notación de objetos de JavaScript, es un formato de texto sencillo utilizado para el intercambio de datos.

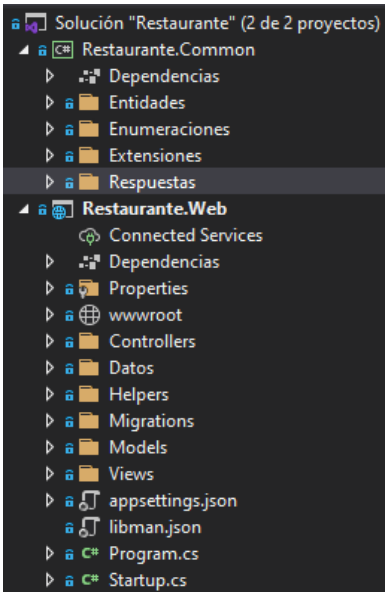


Figura 2.21. Estructura de la solución de Visual Studio

2.2.3. MODELO

En esta sección se realiza una explicación sobre las *Data Annotations*, la codificación del modelo utilizando Entity Framework mediante el enfoque Code First y la migración del modelo hacia la base de datos SQL Server.

2.2.3.1. DATA ANNOTATIONS

Es una librería que permite realizar validaciones de un modelo de datos de una manera más simple y sencilla ajustándola a las necesidades del programador. Entre las principales *Data Annotations* que se utilizaron se encuentran:

`DisplayName`: Permite definir como se muestra el nombre de cada atributo en la UI[25].

`MaxLenght`: Define el tamaño de longitud máximo de un valor ingresado del formulario.

`Required`: Permite validar los valores obligatorios de un formulario.

`Compare`: Permite comparar dos valores.

`DisplayFormat`: Define el formato específico de un valor ingresado.

`Range`: Especifica un valor máximo y un mínimo de un valor numérico ingresado.

[25] UI: Interfaz de Usuario, hace referencia a la interfaz de usuario con la que interactúa directamente el usuario final en la aplicación.

`EmailAddress`: Valida que el valor ingresado sea un correo electrónico válido.

`DataType`: Permite definir un valor de una manera más específica.

`StringLength`: Especifica un valor máximo y un mínimo de una cadena de texto.

2.2.3.2. CODIFICACIÓN DEL MODELO

En esta sección, se explica la codificación del modelo mediante clases escritas en el lenguaje de programación C#.

En la carpeta `Entidades` del proyecto `Restaurante.Common` se crearon las clases `TipoPedido`, `Producto`, `OpcionMenuDia`, `Mesa`, `DetallePedido`, y `Categoria` que hacen referencia a las tablas de la base de datos.

En la Figura 2.22 se muestra la clase correspondiente a la tabla `TipoProductos` con cada uno de sus atributos, en el recuadro en rojo se encuentran las *Data Annotarions* correspondiente al atributo `Nombre_Tipo_Producto`. Como se puede apreciar se pueden utilizar una o varias *Data Annotations* lo que permite realizar varias validaciones por cada atributo, la propiedad del tipo `ICollection<>` permite relacionar la tabla `TipoProductos` con la tabla `Productos` según el diseño realizado.

```
12  public class TipoProducto
13  {
14      public int Id { get; set; }
15      //Data Annotations
16      [DisplayName("Tipo")]
17      [MaxLength(50)]
18      [Required(ErrorMessage ="Debe Ingresar un nombre")]
19      public string Nombre_Tipo_Producto { get; set; }
20
21      public ICollection<Producto> Productos { get; set; }
22
23      public bool isDeleted { get; set; }
24  }
```

Figura 2.22. Clase correspondiente a la tabla `TipoProductos`

En la Figura 2.23 y Figura 2.24 se muestra la clase que hace referencia a la tabla `Productos`, el atributo `ImageId` (línea 33) del tipo `GUID`[26] es el identificador de la imagen del producto, el atributo `ImageFullPath` no se carga en la base de datos, pero se utiliza

[26] GUID: Identificador global único, es utilizado para identificar recursos utilizando algoritmos pseudoaleatorios.

para poder mostrar la imagen en las vistas. Las imágenes que se cargan a cada producto se almacenan en un contenedor de *Blob Storage*[27] de Azure, en caso de no cargar ninguna imagen al momento de crear el producto se muestra una imagen predeterminada cargada en la carpeta `imagenes` (líneas 35-38).

```
11      public class Producto
12      {
13          public int Id { get; set; }
14
15          [DisplayName("Nombre")]
16          [MaxLength(50)]
17          [Required(ErrorMessage = "Debe ingresar un nombre")]
18          public string NombreProducto { get; set; }
19
20          [DisplayName("Descripción")]
21          [DataType(DataType.MultilineText)]
22          public string Descripcion { get; set; }
23
24          [DisplayFormat(DataFormatString = "{0:C2}")]
25          public decimal Precio { get; set; }
26
27          [DisplayName("Estado(Activo/Inactivo)")]
28          public bool IsActive { get; set; }
29
30          public TipoProducto TipoProducto { get; set; }
```

Figura 2.23. Clase correspondiente a la tabla `Productos` parte I

```
32          [Display(Name = "Image")]
33          public Guid ImageId { get; set; }
34
35          [Display(Name = "Imagen")]
36          public string ImageFullPath => ImageId == Guid.Empty
37              ? $"https://localhost:8081/imagenes/noimage.png"//44323
38              : $"https://restaurantedonruiz.blob.core.windows.net/productos/{ImageId}";
39
40          public bool isDeleted { get; set; }
41      }
```

Figura 2.24. Clase correspondiente a la tabla `Productos` parte II

En la Figura 2.25 se muestra la clase que hace referencia a la tabla `OpcionMenuDias` utilizando el atributo `Categoria (línea 20)` para relacionarla con la tabla `Categorías`.

[27] Blob Storage: Almacén de datos no estructurados en la nube.

```
11  public class OpcionMenuDia
12  {
13      public int Id { get; set; }
14
15      [DisplayName("Nombre")]
16      [MaxLength(50)]
17      [Required(ErrorMessage = "Debe Ingresar un nombre")]
18      public string NombreOpcion { get; set; }
19
20      public Categoria Categoria { get; set; }
21
22      [DisplayName("Menu del día")]
23      public bool IsMenuActive { get; set; }
24
25      public bool isDeleted { get; set; }
26  }
```

Figura 2.25. Clase correspondiente a la tabla `OpciónMenuDias`

En la Figura 2.26 se muestra la clase que hace referencia a la tabla `Mesas`, el atributo denominado `EstadoMesa` (línea 20) se utiliza para saber cuándo la mesa se encuentra ocupada o libre. El atributo `isDeleted` es un `bool` que se utiliza para poder eliminar la mesa sin dañar la integridad referencial de la base de datos.

```
12  public class Mesa
13  {
14      public int Id { get; set; }
15
16      [DisplayName("Numero de Mesa")]
17      public int NumeroMesa { get; set; }
18
19      [DisplayName("Estado de la Mesa")]
20      public bool EstadoMesa { get; set; }
21
22      public bool isDeleted { get; set; }
23  }
```

Figura 2.26. Clase correspondiente a la tabla `Mesas`

En la Figura 2.27 se muestra la clase que hace referencia a la tabla `DetallePedidos`, que se utiliza para especificar los detalles de cada uno de los pedidos, el atributo `Producto` permite relacionar con la tabla `Productos` (línea 15), el atributo `TotalDetalle` (línea 24) se utiliza para calcular el total del detalle multiplicando el precio del producto por la cantidad, dicha operación se la realiza el controlador respectivo.

```
11      public class DetallePedido
12      {
13          public int Id { get; set; }
14
15          public Producto Producto { get; set; }
16
17          [DisplayName("Cantidad")]
18          public int Cantidad { get; set; }
19
20          [DataType(DataType.MultilineText)]
21          public string Comentarios { get; set; }
22
23          [DisplayName("Total")]
24          public decimal TotalDetalle { get; set; }
25      }
```

Figura 2.27. Clase correspondiente a la tabla `DetallePedidos`

La codificación de la tabla de Pedidos y las tablas generadas por librería Identity que permiten el manejo de autenticación, usuarios y roles de usuario se presenta en la sección 2.2.3.4.

2.2.3.3. ENUMERACIONES

Las enumeraciones permiten representar un grupo de valores de constantes con un nombre, es decir, asocia un número con un nombre específico.

En el prototipo las enumeraciones que se generaron son:

- `TipoUsuario`: Se utiliza para definir el nombre de los 3 tipos de usuario Administrador, Empleado y Cliente.
- `TipoPedido`: Se utiliza para especificar el tipo de pedido, puede ser para la mesa o para llevar.
- `EstadoPedido`: Se utiliza para realizar la gestión del pedido, se manejan 5 estados: pendiente, preparando, entregado, cobro, pagado.

En la Figura 2.28 se muestra un ejemplo de la enumeración `EstadoPedido` y como se puede apreciar la codificación es muy sencilla, cada uno de los elementos de la enumeración (líneas 11-15) se asocian un número como lo muestran los comentarios, lo que permite dar un manejo más sencillo de la gestión del pedido.

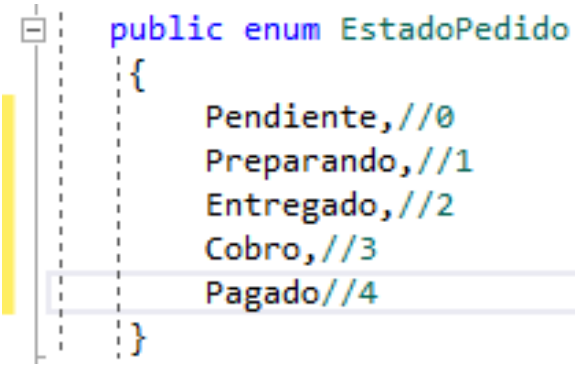


Figura 2.28. Ejemplo de enumeración

2.2.3.4. IDENTITY Y TABLA PEDIDOS

Durante el desarrollo del prototipo se utilizó ASP .NET Core Identity para gestionar los procesos de autenticación, manejo de usuarios, manejo de roles de usuario, utilización de contraseñas seguras, confirmación por correo y recuperación de contraseñas, se lo realizó mediante la implementación de clases y métodos propios de Identity.

Para el proyecto se utilizaron las siguientes clases de Identity:

- `UserManager`: Es una clase que implementa métodos para realizar la gestión de usuarios.
- `RoleManager`: Es una clase que implementa métodos que permiten realizar gestión de roles de usuario.
- `SignInManager`: Es una clase que implementa métodos que facilitan el proceso de autenticación.
- `SignInResult`: Es una clase que muestra el resultado de una operación `SignInManager`.
- `IdentityResult`: Es una clase que muestra el resultado de una operación Identity.

En la Figura 2.29 se define una clase denominada `Usuario` en el proyecto `Restaurante.Web` misma que hereda de la clase `IdentityUser`, lo cual se ha marcado con un recuadro rojo. El hecho de que la clase herede de `IdentityUser` permite que los atributos definidos en la clase se combinen con los atributos que viene por defecto con `IdentityUser` en una sola tabla al momento de realizar la migración del modelo a la base de datos. El atributo `ImageId` (línea 32) es del tipo `GUID`, que se utiliza como identificador de imagen ya que cada usuario puede incluir una imagen de perfil si así lo desea, el atributo `IsAdministrador` (línea 35) se utiliza para gestionar roles de usuario de administrador

o de empleado, el atributo `CambioClave` (línea 37) se utiliza para dar el manejo de la clave cuando se registra un nuevo administrador o empleado. En el recuadro en verde se especifica el rol del usuario con la enumeración `TipoUsuario` y al ser un tipo `enum` equivale a un `int` en la tabla `AspNetUsers` de la base de datos.

```
12  public class Usuario:IdentityUser
13  {
14      [MaxLength(10)]
15      [Required(ErrorMessage ="Es necesario ingresar el número de cédul
16      public string Documento { get; set; }
17
18      [Display(Name = "Nombre")]
19      [MaxLength(50)]
20      [Required(ErrorMessage = "Es necesario ingresar un nombre")]
21      public string PrimerNombre { get; set; }
22
23      [MaxLength(50)]
24      [Required(ErrorMessage = "Es necesario ingresar un apellido")]
25      public string Apellido { get; set; }
26
27      [MaxLength(120)]
28      [Required(ErrorMessage = "Es necesario ingresar una dirección")]
29      public string Direccion { get; set; }
30
31      [Display(Name = "Imagen")]
32      public Guid ImageId { get; set; }
33
34      [Display(Name = "Rol de Usuario")]
35      public bool IsAdministrador { get; set; }
36
37      public bool CambioClave { get; set; }
38
39      [Display(Name = "Rol de Usuario")]
40      public TipoUsuario TipoUsuario { get; set; }
```

Figura 2.29. Clase `Usuario`

La tabla `Pedidos` también se define en el proyecto web, ya que se relaciona directamente con un usuario que realiza el pedido. En la Figura 2.30 se muestra la implementación de la clase que hace referencia a la tabla `Pedidos`, contiene información como la fecha y la misma se relaciona con las tablas de `AspNetUsers` (línea 24), `Mesas` (línea 25) y `DetallePedidos` (línea 33), además, emplea la enumeración `TipoPedido` (línea 28) y `EstadoPedido` (línea 31). El atributo `Total` (línea 35) no se almacena en la base de datos, solo se lo utiliza para poder calcular el valor total del pedido en las vistas correspondientes.

```
15      public class Pedido
16      {
17          public int Id { get; set; }
18
19          [DisplayFormat(DataFormatString = "{0:dd/MM/yyyy HH:mm}")]
20          [Display(Name = "Fecha")]
21          public DateTime Fecha { get; set;  }
22
23          [Display(Name = "Cliente")]
24          public Usuario Usuario { get; set; }
25          public Mesa Mesa { get; set; }
26
27          [Display(Name = "Tipo de Pedido")]
28          public TipoPedido TipoPedido { get; set; }
29
30          [Display(Name = "Estado del Pedido")]
31          public EstadoPedido EstadoPedido { get; set; }
32
33          public ICollection<DetallePedido> DetallePedidos { get; set; }
34
35          public decimal Total => DetallePedidos == null ? 0
36              : DetallePedidos.Sum(dp => dp.TotalDetalle);
37      }
```

Figura 2.30. Clase correspondiente a la tabla `Pedidos`

2.2.3.5. MIGRACIÓN DEL MODELO

Una vez codificadas las clases que hacen referencia al modelo, con la ayuda de Entity Framework, se realiza la conexión con la base de datos y la migración del código en SQL Server, por lo cual primero fue necesario descargar e instalar en la solución los siguientes paquetes NuGet:

- `Microsoft.EntityFrameworkCore`[28]
- `Microsoft.EntityFrameworkCore.SqlServer`[29]
- `Microsoft.EntityFrameworkCore.Tools`[30]

Para realizar la conexión con la base de datos se incluyó un `string` de conexión de la base de datos en el archivo de configuración `appsettings.json`. También se creó una clase denominada `DbContext` en el proyecto web que contiene el nombre de las tablas que se migran a la base de datos y permite la comunicación con la misma, posteriormente, se realiza la inyección del servicio en el contenedor de servicios mediante el archivo `Startup.cs`.

[28] Paquete NuGet que permite utilizar Entity Framework Core en el proyecto.

[29] Paquete NuGet que permite utilizar Entity Framework Core en conjunto con SQL Server.

[30] Paquete NuGet que permite realizar migraciones de un modelo basado en Entity Framework hacia un motor de base de datos.

```
//para migrar y mapear las tablas en la DB
public DbSet<TipoProducto> TipoProductos { get; set; } //Tabla TipoProductos
public DbSet<Producto> Productos { get; set; }//Tabla Productos
public DbSet<Categoria> Categorias { get; set; }//Tabla Categorias
public DbSet<OpcionMenuDia> OpcionMenuDias { get; set; } //Tabla OpcionMenuDias
public DbSet<Mesa> Mesas { get; set; } //Tabla Mesas
public DbSet<DetallePedido> DetallePedidos { get; set; }// Tabla DetallePedidos
public DbSet<Pedido> Pedidos { get; set; }// Tabla Pedidos
```

Figura 2.31. Código para generar las tablas usando la clase `DbContext`

```
services.AddDbContext<DataContext>(cfg =>
{
    cfg.UseSqlServer(Configuration.GetConnectionString("DefaultConnection"));
});
```

Figura 2.32. Inyección del servicio de base de datos en el archivo `Startup.cs`

Finalmente, se realizó la migración del modelo a la base de datos mediante la utilización de la consola de administración de paquetes del proyecto insertando los siguientes comandos

- `add-migration`: Crea una nueva migración con un nombre específico definido por el programador.
- `update-database`: Ejecuta la última migración realizada para actualizar la base de datos.

Es importante tomar en cuenta las siguientes consideraciones una vez que se realiza la migración:

- El utilizar la clase `IdentityUser` genera automáticamente varias tablas en la base de datos, pero en el prototipo solo utilizaron las siguientes tablas: `AspNetUsers`, `AspNetRoles` y `AspNetUserRoles`.
- Las tablas `AspNetUsers`, `AspNetRoles` y `AspNetUserRoles` contienen todos los atributos necesarios que permiten realizar los procesos de autenticación, gestión de usuarios, manejo de roles de usuario y confirmación del registro por correo.
- Las enumeraciones no se almacenan directamente en la base de datos del prototipo, son atributos del tipo `int` en las tablas en las que se emplean. Por ejemplo, la tabla `Pedidos` cuenta con un atributo de tipo `int` denominado

`TipoPedido` y con otro atributo de tipo `int` denominado `EstadoPedido`, como se aprecia en las columnas del recuadro rojo de la Figura 2.33.

Id	Fecha	UsuarioId	MesaId	TipoPedido	EstadoPedido
10	2022-01-24 12:28:32.2186369	a2019bed-9874-4911-8d0e-186b4b279992	1	0	4
11	2022-01-24 12:29:10.5971027	f1f8fa63-2b57-4be0-af28-b6b2fc0edc06	1	0	4
12	2022-01-30 13:27:48.4397313	a2019bed-9874-4911-8d0e-186b4b279992	3	0	4
13	2022-01-31 10:25:34.5895771	a2019bed-9874-4911-8d0e-186b4b279992	3	0	4
14	2022-01-31 10:28:21.8683785	a2019bed-9874-4911-8d0e-186b4b279992	4	0	4
15	2022-02-06 13:20:28.2963895	60ecb04c-eb99-49ba-bad1-f3bffe77d95d	4	0	4

Figura 2.33. Contenido de la tabla `Pedidos`

2.2.4. INYECCIÓN DE DEPENDENCIAS

Es una técnica que consiste en dotar de un acoplamiento flexible a los componentes de una aplicación separando las responsabilidades de cada uno de estos, con el objetivo de obtener una aplicación mucho más modular, fácil de probar y de mantener en el tiempo [23].

ASP .NET Core MVC incorpora la inyección de dependencias de forma explícita en los constructores que se encuentran en los controladores, en la Figura 2.34 se muestra un ejemplo de una inyección de dependencias, las dependencias se definen como abstracciones (interfaces) de tipo privado y de solo lectura (marcado con un recuadro verde) que se inicializan en el constructor (marcado con un recuadro rojo) de la clase del controlador de `Productos` pudiendo utilizar los métodos que implementan cada una de las interfaces.

```
public class ProductosController : Controller
{
    private readonly DataContext _context;
    private readonly ICombosHelper _combosHelper;
    private readonly IConverterHelper _converterHelper;
    private readonly IBlobHelper _blobHelper;

    public ProductosController(DataContext context, IBlobHelper blobHelper,
        IConverterHelper converterHelper, ICombosHelper combosHelper)
    {
        _context = context;
        _blobHelper = blobHelper;
        _converterHelper = converterHelper;
        _combosHelper = combosHelper;
    }
```

Figura 2.34. Ejemplo de inyección de dependencias

La carpeta `Helpers` del proyecto web contiene las interfaces y las clases que implementan dichas interfaces, estas se utilizan para realizar la inyección de dependencias mediante constructores. Las interfaces que se implementaron en el prototipo son las siguientes:

- `ICombosHelper`: Define las firmas de métodos que se utilizan para realizar listas desplegables con la información de las tablas de la base de datos, y se implementa mediante la clase `CombosHelper`.
- `IConverterHelper`: Define las firmas de métodos encargados de convertir los modelos de datos Entity Framework en modelos ViewModel y viceversa, y se implementa mediante la clase `ConverterHelper`.
- `IBlobHelper`: Define la firma de un método encargado de enviar las imágenes del prototipo al contenedor del *Blob Storage* de Azure y de generar el identificador del tipo GUID de cada imagen, y se implementa mediante la clase `BlobHelper`.
- `IUserHelper`: Define la firma de los métodos encargados de realizar procesos de gestión de usuarios, roles de usuario y autenticación. Se implementa mediante la clase `UserHelper`.
- `IMailHelper`: Define la firma de un método para realizar el envío del correo de confirmación cuando se registra un nuevo usuario o cuando un usuario registrado en el sistema está intentando recuperar la contraseña. Se implementa mediante la clase `MailHelper`.
- `IVentasHelper`: Define la firma de un método para realizar los cálculos necesarios para mostrar las ventas del día o de una fecha, y se implementa mediante la clase `VentasHelper`.

Además, es importante incorporar las dependencias en el contenedor de servicios mediante el archivo `Startup.cs`, tal como lo muestra la Figura 2.35.

```
services.AddScoped<IBlobHelper, BlobHelper>();//Inyección del servicio BlobHelper
services.AddScoped<ICombosHelper, CombosHelper>();//Inyección del servicio CombosHelper
services.AddScoped<IConverterHelper, ConverterHelper>();//Inyección del servicio ConverterHelper
services.AddScoped<IUserHelper, UserHelper>();//Inyección del servicio UserHelper
services.AddScoped<IMailHelper, MailHelper>();//Inyección del servicio MailHelper
services.AddScoped<IVentasHelper, VentasHelper>();//Inyección del servicio VentasHelper
```

Figura 2.35. Dependencias el contenedor de servicios del `Startup.cs`

2.2.5. IMPLEMENTACIÓN DE UN SEEDER

La implementación de un *Seeder* tiene como objetivo cargar datos en algunas tablas de la base de datos del prototipo. Es importante tomar en cuenta que el *Seeder* funcionará

únicamente cuando la base de datos se encuentre vacía o cuando la base de datos aún no esté creada. En el proyecto web se definió una clase denominada `SeedDB` que define métodos que se encargan de realizar las siguientes tareas:

- Verificar si la base de datos existe y si no existe crea una nueva con el esquema de tablas definido por Entity Framework.
- Cargar datos en la tabla `TipoProductos`, `Mesas` y `Categorias` cuando las mismas estén vacías.
- Cargar roles de usuario en la tabla `AspNetRoles` y cargar un usuario administrador en caso de no exista ninguno en la base de datos.

2.2.6. CONTROLADORES Y VISTAS

A continuación, se explica los controladores y las vistas que se implementaron en el prototipo.

2.2.6.1. TIPOS DE VISTAS UTILIZADOS EN EL PROYECTO

ASP .NET Core permite utilizar 3 tipos de vistas, las cuales se describen a continuación:

- ***Layout:*** El *Layout* se implementa en archivo `_Layout.cshtml` y es la vista que contiene el diseño base del prototipo, el *header*[31], el *footer*[32], el menú de navegación principal, además, contiene las referencias comunes de las librerías y a los archivos que se utilizan en el prototipo.
- **Vistas:** Son las vistas que se utilizaron mayormente y están asociadas con acciones implementadas en los controladores.
- **Vistas Parciales:** Son vistas que se utilizaron para reutilizar contenido en común en las vistas y que no necesitan formar parte del *Layout*.

2.2.6.2. VISTA DE INICIO

En la Figura 2.36 se muestra la vista de inicio para la cual se utiliza una vista de la carpeta `Home` denominada `Index`. La vista `Index` es una vista general para todos los usuarios que acceden al sistema, incluye una imagen con el logotipo principal del restaurante, así como un mensaje de bienvenida, y una opción para redirigirse a la vista de autenticación del sistema. Y al igual que todas las vistas posee el *footer* definido en el archivo

[31] *Header*: Hace referencia a la etiqueta de la cabecera HTML que contienen todas las vistas del prototipo.

[32] *Footer*: Hace referencia a la etiqueta del pie de página HTML que contienen todas las vistas del prototipo.

`_Layout.cshtml` del proyecto, donde se encuentra información general del restaurante como los horarios de atención, dirección, redes sociales, teléfonos y dirección de correo electrónico.

Figura 2.36. Vista de inicio

2.2.6.3. AUTENTICACIÓN DEL PROTOTIPO

Para realizar el proceso de autenticación en el prototipo se le implementa por medio de un controlador denominado `Account` utilizando dos acciones denominadas: `Login` y `Logout`, que permiten realizar el inicio sesión o cierre de sesión en el prototipo respectivamente.

En la Figura 2.37 se muestra la definición de una clase ViewModel denominada `LoginViewModel` necesaria para definir un modelo que permita realizar la autenticación del sistema, la clase define 2 propiedades: `Username`, que debe ser un correo con un formato válido (líneas 11-13), y `Password`, que debe estar oculta y que debe tener una longitud mínima y máxima (líneas 15-20).

```
9   public class LoginViewModel
10  {
11      [Required(ErrorMessage = "El campo es obligatorio")]
12      [EmailAddress(ErrorMessage = "Se debe ingresar un correo válido")]
13      public string Username { get; set; }
14
15      [Required(ErrorMessage = "El campo es obligatorio")]
16      [DataType(DataType.Password)]
17      //La contraseña debe tener al menos 8 caracteres
18      [StringLength(100, MinimumLength = 8,
19      ErrorMessage = "El campo Contraseña debe contener al menos {2} caracteres")]
20      public string Password { get; set; }
21  }
```

Figura 2.37. Clase `LoginViewModel`

En la Figura 2.38 se muestra la vista de inicio de sesión del prototipo, la cual solicita el ingreso de un usuario y una contraseña, además posee un enlace con un mensaje que indica `¿Has olvidado tu contraseña?` para redirigir a otra vista que permite recuperar la contraseña en caso de ser necesario, además, la vista posee un botón para iniciar sesión una vez que se han ingresado las credenciales y otro botón para registrarse, en caso de que no sean clientes y no posean una cuenta en el prototipo. Las credenciales se envían en el modelo `LoginViewModel` por medio del método `POST`, y en la aplicación web corresponden a la acción `Login` del controlador, en dicha acción se procede a validar la información del modelo y dar acceso al prototipo en caso de que sean válidas, sino se muestra un mensaje de error.

Figura 2.38. Vista Login

En la Figura 2.39 se muestra el método de acción `Login` del controlador `Account`, el cual está relacionado con la acción *login*, este método es asíncrono, es decir que la operación se puede ejecutar al mismo tiempo que otras operaciones, y devuelve una operación del tipo `Task`[33]`<IActionResult`[34]`>`. El método de acción utiliza el método `POST` (línea 66), recibe un modelo `LoginViewModel` (línea 68), y se encarga de verificar que el modelo sea válido, después mediante el método `LoginAsync` el cual espera por una señal mediante el objeto `SignInResult` denominado `result`, si el objeto `result` es exitoso se obtiene el usuario mediante el método `ObtenerUserAsync` y se valida el valor del atributo `CambioClave` del usuario, esta validación se realiza con el objetivo de cambiar la clave en el primer inicio de sesión de un usuario administrador o empleado nuevo. Dependiendo del resultado de la validación se redirige a la vista correspondiente (líneas 70-90).

```
66      [HttpPost]
67      [ValidateAntiForgeryToken]
68      public async Task<IActionResult> Login(LoginViewModel modelo)
69      {
70          if (ModelState.IsValid)
71          {
72              Microsoft.AspNetCore.Identity.SignInResult result = await _userHelper
73                  .LoginAsync(modelo);
74              //Si se pudo logear
75              if (result.Succeeded)
76              {
77                  Usuario usuario = await _userHelper.ObtenerUserAsync(modelo.Username);
78                  if (usuario.CambioClave == false)
79                  {
80                      return RedirectToAction("IndexLogin", "Home");
81                  }
82                  else
83                  {
84                      return RedirectToAction("ChangePassword");
85                  }
86              }
87              ModelState.AddModelError(string.Empty, "Email o contraseña incorrecta.");
88          }
89          return View(modelo);
90      }
```

Figura 2.39. Método de acción `Login`

Una vez autenticado un usuario y dependiendo del rol que cumple dentro del prototipo, el mismo se encarga de presentar al usuario el menú de navegación principal con las diferentes opciones a las que puede acceder.

[33] `Task`: es una tarea asincrónica que no requiere recursos adicionales de memoria o CPU.

[34] `IActionResult`: Interfaz que se implementa en una clase `ActionResult` que se encarga de presentar el resultado de una acción.

En la Figura 2.40 se muestran las opciones del menú de navegación para el rol administrador, ahí se puede apreciar que cuenta con opciones para gestionar los productos, el menú del día, usuarios y clientes, pedidos y las mesas. En tanto que en la Figura 2.41 se muestra las opciones del menú de navegación para el rol cliente en donde se puede apreciar que tiene opciones para visualizar el menú, un historial de los pedidos que ha realizado y las mesas.

La pestaña denominada `Cuenta` muestra en todos los usuarios y se utiliza para editar los datos personales o cambiar la clave del usuario cuando este lo requiera. El logotipo de la esquina superior izquierda permite redireccionar al usuario a la vista de inicio del sistema.

Figura 2.40. Opciones del menú de navegación para el rol administrador

Figura 2.41. Opciones del menú de navegación para el rol cliente

2.2.6.4. CARGAR IMÁGENES

Las vistas para agregar o editar los `productos` y `usuarios` permiten cargar una imagen, si no se carga ninguna imagen, el prototipo procede a cargar una imagen predeterminada denominada `noimage.png`, que se encuentra en la carpeta `imagenes` del proyecto web. Como primer paso para cargar imágenes se creó un *Blob Storage* denominado `restaurantedonruiz` en Azure que a su vez contiene 2 contenedores para almacenar las imágenes de `productos` y `usuarios`, tal como lo muestra Figura 2.42.

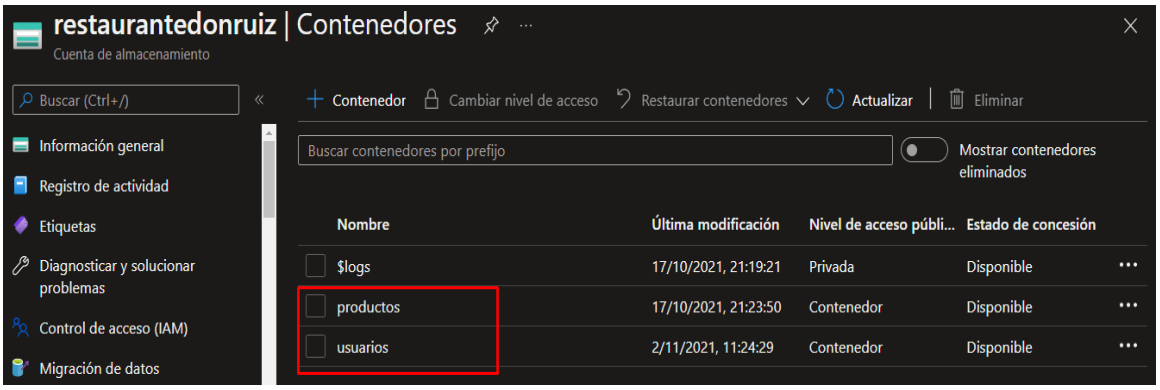


Figura 2.42. Contenedores del *Blob Storage*

Al igual que con la base de datos, para sincronizar el prototipo con el *Blob Storage* de Azure es necesario configurar un `string` de conexión del blob en el archivo de configuración `appsettings.json`. Un ejemplo se presenta en la Figura 2.43.

```
14    "Blob": {
15      "ConnectionString":
16        "DefaultEndpointsProtocol=https;AccountName=restaurantedonruiz;AccountKey=Bz4g9ARb
17    },
```

Figura 2.43. `String` de conexión del *Blob Storage*

Posteriormente, se descargó e instaló en el proyecto web un paquete NuGet denominado `Windows.Azure.Storage`[35], después se procedió a crear una interfaz `IBlobHelper` para utilizar el principio de inyección de dependencias, la interfaz mencionada define un método asíncrono denominado `UploadBlobAsync` que recibe un archivo `IFormFile`[36] y el nombre del contenedor, y devuelve el `GUID` con el identificador de la imagen que se almacena en la base de datos (ver Figura 2.44).

[35] Paquete NuGet que permite almacenar datos binarios y de texto en el servicio de *Blob Storage* de Azure.

[36] Es una propiedad de un objeto que permite enviar un archivo a través de una solicitud HTTP.

```
public interface IBlobHelper
{
    Task<Guid> UploadBlobAsync(IFormFile file, string containerName);
}
```

Figura 2.44. Interfaz `IBlobHelper`

En la Figura 2.45 y Figura 2.46 se muestra la clase `BlobHelper` que implementa la interfaz `IBlobHelper`. Primero se define en el constructor de la clase `BlobHelper` la conexión con el *blob* mediante las credenciales especificadas en el archivo de configuración (líneas 16-25). Después, se sobrescribe el método `UploadBlobAsync`, para lo cual primero se obtiene el `stream`[37] de la imagen (línea 30), luego se obtiene la referencia del contenedor dentro del *blob* y finalmente se carga el `stream` de la imagen en el *blob* y se devuelve el `GUID` de la imagen para almacenarlo en la base de datos (líneas 34-38).

```
13      public class BlobHelper : IBlobHelper
14      {
15          private readonly CloudBlobClient _blobClient;
16          public BlobHelper(IConfiguration configuration)
17          {
18              //Obtener las credenciales del string de conexion del blob
19              //storage en el appsettings.json
20              string keys = configuration["Blob:ConnectionString"];
21              //Se vincula con el blob storage account con las credenciales
22              CloudStorageAccount storageAccount = CloudStorageAccount.Parse(keys);
23              //Se crea el blob
24              _blobClient = storageAccount.CreateCloudBlobClient();
25          }
```

Figura 2.45. Codificación de la clase `BlobHelper` parte I

```
26      public async Task<Guid> UploadBlobAsync(IFormFile file, string containerName)
27      {
28          //stream para manejar la entrada y salida de bytes
29          //obtengo un stream a partir del file(Imagen)
30          Stream stream = file.OpenReadStream();
31          //El identificador que se le da a la imagen
32          Guid name = Guid.NewGuid();
33          //Obtener la referencia del contenedor a la que hace el containerName
34          CloudBlobContainer container = _blobClient.GetContainerReference(containerName);
35          CloudBlockBlob blockBlob = container.GetBlockBlobReference($"{name}");
36          //El método que sube la imagen al blob
37          await blockBlob.UploadFromStreamAsync(stream);
38          return name;
39      }
```

Figura 2.46. Codificación de la clase `BlobHelper` parte II

[37] *Stream*: Es una secuencia de bytes utilizados para la transferencia de datos.

Al utilizar la propiedad `IFormFile` para cargar imágenes, no existe una *data annotation* que permita validar la extensión del archivo que se carga en el campo imagen de los formularios, por lo cual fue necesario crear una clase denominada `AllowedExtensions` que permita realizar dicha validación. En la Figura 2.47 se muestra lo explicado anteriormente, el atributo `ImageFile` posee una línea `AllowedExtensions` para realizar la validación en base al arreglo de *strings* que contiene los formatos válidos de imagen.

```
[Display(Name = "Imagen (Formatos '.jpg', '.jpeg' o '.png')")]
[AllowedExtensions(new string[] { ".jpg", ".jpeg", ".png" })]
public IFormFile ImageFile { get; set; }
```

Figura 2.47. Atributo utilizado para cargar una imagen

En la Figura 2.48 se presenta el método `IsValid` de `AllowedExtensions` que recibe el archivo que se carga en el formulario respectivo, en caso de que el mismo exista, obtiene la extensión y valida que contenga una extensión de las especificadas en la Figura 2.47, en caso de que no tenga una extensión válida muestra un mensaje de error caso contrario devuelve un resultado exitoso.

```
21  protected override ValidationResult IsValid(object value,
22      ValidationContext validationContext)
23  {
24      var file = value as IFormFile;
25      if (file != null)
26      {
27          var extension = Path.GetExtension(file.FileName);
28          //Si no contiene las extensiones ingresadas(jpg,png,jpeg).
29          if (!_extensions.Contains(extension.ToLower()))
30          {
31              return new ValidationResult($"El archivo no corresponde a una imagen " +
32                  $"en el formato correcto. Seleccione un archivo con una extensión " +
33                  $"jpg,jpeg o png");
34          }
35      }
36      return ValidationResult.Success;
37  }
```

Figura 2.48. Método `IsValid` para validar extensiones de archivos

2.2.6.5. GESTIÓN DE USUARIOS

Tomando en cuenta los requerimientos establecidos en la sección 2.1.2 la gestión de usuarios está a cargo del usuario administrador, ya que este puede realizar operaciones CRUD de empleados y administradores.

Lista de usuarios

Un usuario administrador es capaz de visualizar las listas con todos los usuarios administradores y empleados registrados en el sistema. En la Figura 2.49 se muestra la vista correspondiente a la pestaña `Usuarios` (recuadro verde) que presenta la lista de usuarios administradores y empleados registrados y que se encuentran activos en el sistema. La tabla muestra información relevante como el nombre, el rol de usuario, correo electrónico, dirección, número telefónico, una imagen y un par de botones. El botón amarillo se utiliza para visualizar con más detalle la información y editar la misma en caso de ser necesaria y el botón rojo permite eliminar el usuario.

La vista de usuarios además cuenta con un botón azul para agregar un nuevo usuario administrador o empleado y otro botón de color rojo para visualizar los usuarios que no se encuentran activos en el sistema. Un usuario no está activo cuando el mismo no ha confirmado el correo electrónico o ha sido eliminado por un administrador.

Figura 2.49. Vista con la lista usuarios administradores y empleados

En la Figura 2.50 se muestra el método de acción `Index` del controlador `Account` que permite obtener por medio del método `GET` los datos de la lista de usuarios para poder mostrarlos en la vista, por defecto si no se especifica el método de HTTP, se asume que es `GET`. `Authorize` (línea 40) es una operación que se utiliza para especificar que la acción solamente la puede realizar un usuario administrador, la acción devuelve una vista con el mismo nombre `Index` con un objeto que contiene los datos solicitados en forma de lista. La operación `Where` (línea 44) se utiliza para filtrar los datos y que se envíen

únicamente los usuarios administradores y clientes que están activos en el sistema. El atributo `EmailConfirmed` (línea 44) es un booleano de la tabla de usuarios creado automáticamente por Identity y se lo utiliza para activar usuarios en el sistema al momento que se confirma el correo.

```
40      [Authorize(Roles = "Administrador")]
41      public async Task<IActionResult> Index()
42      {
43          return View(await _context.Users
44              .Where(u => u.TipoUsuario != TipoUsuario.Cliente && u.EmailConfirmed != false)
45              .ToListAsync());
46      }
```

Figura 2.50. Código de la lista de usuarios

En la Figura 2.51 de igual manera se muestra la lista de clientes registrados, la vista es similar a la de la Figura 2.49 y tiene funcionalidades similares con la diferencia de que no existe un botón para agregar clientes, ya que el registro de clientes lo hacen directamente los clientes con el botón registrarse en la vista de *Login*.

Figura 2.51. Vista con la lista clientes

Agregar nuevos usuarios

Solo un usuario administrador puede agregar un usuario administrador o empleado. Para realizar el proceso de creación de usuarios fue necesario agregar dos clases ViewModel, la primera denominada `EditUserViewModel` que contiene todos los campos necesarios

para el ingreso de información de usuario como el nombre, apellido, imagen, etc. En la Figura 2.52 se muestra la segunda clase ViewModel denominada `AddUserViewModel` que hereda de `EditUserViewModel` (línea 9) y se utiliza para la creación, validación y edición de las credenciales de usuario ingresadas para autenticarse en el sistema.

```
9   public class AddUserViewModel:EditUserViewModel
10  {
11      [Display(Name ="Email")]
12      [Required(ErrorMessage ="El campo {0} es obligatorio")]
13      [EmailAddress(ErrorMessage = "Se debe ingresar un correo válido")]
14      public string Username { get; set; }
15
16      [Display(Name = "Contraseña")]
17      [Required(ErrorMessage = "El campo Contraseña es obligatorio")]
18      [DataType(DataType.Password)]
19      [StringLength(100,MinimumLength =8,
20          ErrorMessage ="El campo Contraseña debe contener al menos {2} caracteres")]
21      public string Password { get; set; }
22
23      [Display(Name = "Confirmar Contraseña")]
24      [Required(ErrorMessage = "El campo Contraseña es obligatorio")]
25      [DataType(DataType.Password)]
26      [StringLength(100,MinimumLength = 8,
27          ErrorMessage = "El campo Contraseña debe contener al menos {2} caracteres")]
28      [Compare("Password",ErrorMessage ="Las contraseñas no coinciden")]
29      public string PasswordConfirm { get; set; }
30  }
```

Figura 2.52. Clase ViewModel `AddUserViewModel`

En la Figura 2.53 se muestra parte del formulario utilizado para crear un nuevo usuario empleado o administrador, el mismo contiene los campos y las validaciones necesarias para el ingreso de información.

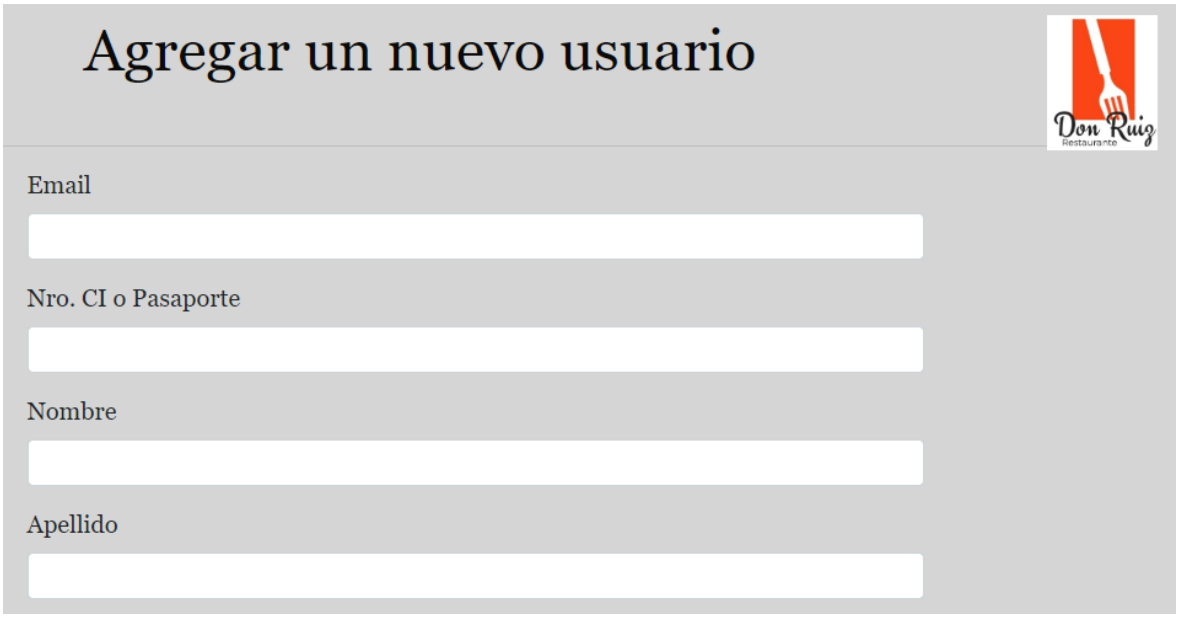


Figura 2.53. Formulario para agregar un nuevo usuario

Para la construcción del formulario se utiliza una vista parcial denominada `_Usuario`, la vista parcial permite el reutilizar su código en las vistas que se encargan de crear, editar y registrar los datos del usuario. En la Figura 2.54 se muestra la vista, la cual emplea el modelo `EditUserViewModel` y cuenta con las etiquetas HTML necesarias para ingresar los datos de usuario. Para agregar la vista parcial `_Usuario` en otras vistas se utiliza la etiqueta `<partial name>`.

```
1  @model Restaurante.Web.Models.EditUserViewModel
2  <div class="form-group">
3      <label asp-for="Documento" class="control-label"></label>
4      <input asp-for="Documento" class="form-control" />
5      <span asp-validation-for="Documento" class="text-danger"></span>
6  </div>
7  <div class="form-group">
8      <label asp-for="Nombre" class="control-label"></label>
9      <input asp-for="Nombre" class="form-control" />
10     <span asp-validation-for="Nombre" class="text-danger"></span>
11 </div>
12 <div class="form-group">
13     <label asp-for="Apellido" class="control-label"></label>
14     <input asp-for="Apellido" class="form-control" />
15     <span asp-validation-for="Apellido" class="text-danger"></span>
16 </div>
17 <div class="form-group">
18     <label asp-for="Direccion" class="control-label"></label>
19     <input asp-for="Direccion" class="form-control" />
20     <span asp-validation-for="Direccion" class="text-danger"></span>
21 </div>
```

Figura 2.54. Código de la vista parcial `_Usuario`

Para realizar el registro de clientes se maneja un formulario similar al presentado en la Figura 2.53, la diferencia radica en la lógica del negocio. En particular se emplea el controlador `Account` y el método de acción denominado `Register`, en tanto que para crear administradores o empleados se utiliza el método de acción `Create`, el cual remite las credenciales de acceso por correo y una vez que confirme y realice el primer inicio de sesión se solicita el cambio de contraseña.

Manejo de contraseñas

Una de las bondades de Identity es que permite el manejo de contraseñas fuertes[38] y cifradas en la base de datos de manera automática. La Figura 2.55 muestra el código requerido para la configuración de contraseñas fuertes, lo cual se realiza al inyectar el servicio de autenticación en el contenedor de servicios en el archivo `Startup.cs`.

[38] Contraseñas que posean al menos 8 caracteres de longitud una mayúscula, una minúscula, un número y un carácter especial.

```
cfg.Password.RequireDigit = true;//password con numeros
cfg.Password.RequiredUniqueChars = 0;//password con caracteres unicos
cfg.Password.RequireLowercase = true;//password con minusculas
cfg.Password.RequireNonAlphanumeric = true;//password con caracteres especiales
cfg.Password.RequireUppercase = true;//password con mayusculas
```

Figura 2.55. Configuración de contraseñas fuertes

Correo de confirmación

El correo de confirmación se lo utiliza con el objetivo de validar que los correos registrados en el prototipo existan cuando se registra un nuevo usuario, por lo cual es necesario realizar la configuración de una cuenta de correo en el archivo `appsettings.json` que será el encargado de enviar los correos de confirmación, tal como lo muestra la Figura 2.56. Por conveniencia se utilizó una cuenta de correo de Gmail[39] (línea 19), además se configuró el servidor SMTP, se especificó el número de puerto SMTP y la contraseña de la cuenta de correo (líneas 20-23).

```
18    "Mail": {
19      "From": "restaurantedonruiz@gmail.com",
20      "Smtp": "smtp.gmail.com",
21      "Port": 587,
22      "Password": [redacted]
23    }
```

Figura 2.56. Configuración de la cuenta de correo en el archivo de configuración

Además, fue necesario descargar e instalar el paquete NuGet `MailKit`[40], y crear la interfaz `IMailHelper` para definir un método denominado `EnviarMail`, el cual se requiere para realizar el envío del correo al usuario. El código de la interfaz se presenta en la Figura 2.57. El método `EnviarMail` devuelve un objeto denominado `Respuesta` que se utiliza para indicar si el correo fue enviado exitosamente o no y se lo emplea al momento de crear o registrar nuevos usuarios en el sistema.

```
public interface IMailHelper
{
    Respuesta EnviarMail(string to, string subject, string body);
}
```

Figura 2.57. Interfaz IMailHelper

[39] Gmail: Servicio de correo electrónico de Google.

[40] MailKit: Paquete NuGet de cliente de correo.

En la Figura 2.58 se muestra un ejemplo del correo enviado cuando se registra un usuario nuevo, para el ejemplo se utiliza un usuario administrador y como se puede apreciar en el recuadro en rojo se envía un enlace que contiene el id de usuario y un *token*[41] generado por un método de la interfaz `IUserHelper`. Cuando el usuario realice un clic sobre dicho enlace, se confirma la recepción del correo y actualiza el valor `EmailConfirmed` en la tabla `usuarios`.

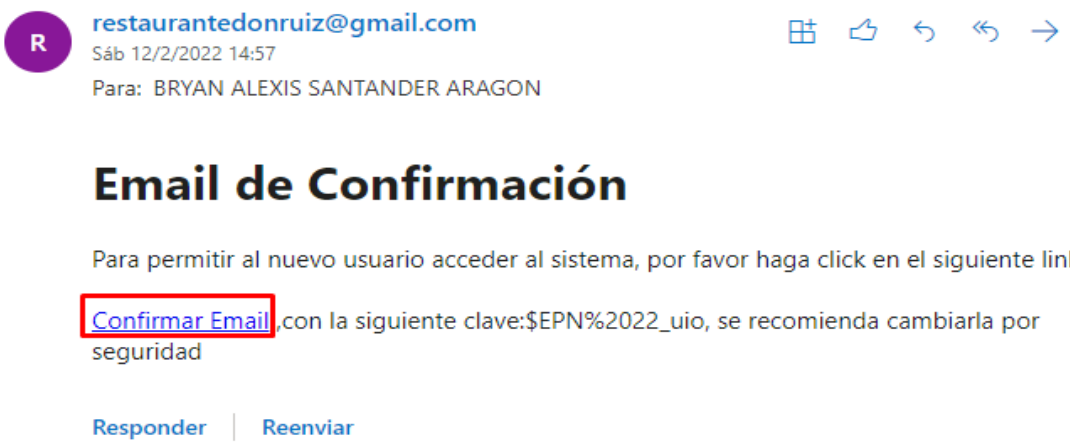


Figura 2.58. Correo electrónico de confirmación

Eliminar usuarios

Al utilizar una base de datos relacional es importante no eliminar datos que puedan dañar la integridad referencial de la misma, por ese motivo para manejar la eliminación de usuarios se utiliza un atributo de tipo `bool` denominado `EmailConfirmed` que cuando su valor es `true` indica que la cuenta de usuario se encuentra activa, caso contrario la cuenta estará inactiva y el usuario no podrá acceder al prototipo.

En la Figura 2.59 se muestra una alerta que solicita la confirmación para eliminar una cuenta de usuario en la cual se ha resaltado mediante un recuadro su nombre. Si se confirma la eliminación, el valor de `EmailConfirmed` pasa a `false` y no se muestra en la lista de usuarios, además, dicho usuario ya no podrá acceder al prototipo. Para el manejo de la eliminación de usuarios se utilizó AJAX que mediante el método `POST` envía la dirección del correo del usuario que se desea eliminar al método de acción correspondiente del controlador y para el manejo de alertas se utilizó *sweetalert*[42].

[41] *Token*: Es un conjunto de caracteres utilizados como identificador que permite convertir datos sensibles en datos seguros.

[42] *Sweetalert*: Es una librería de JavaScript utilizada para el manejo de alertas personalizadas.

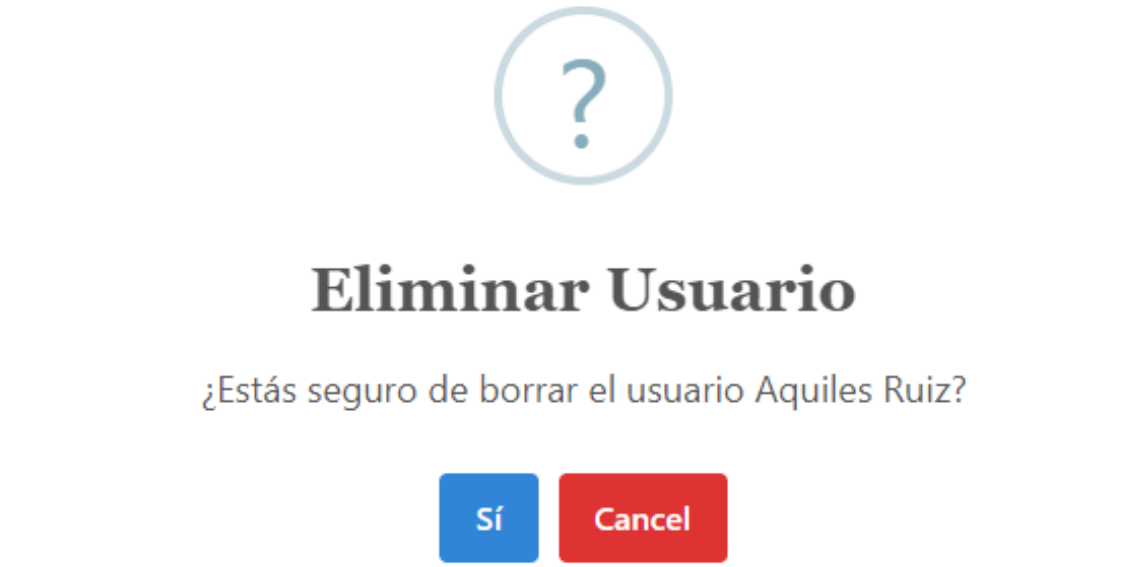


Figura 2.59. Confirmación para eliminar usuario.

Los usuarios que se encuentran eliminados o inactivos se muestran en una lista, como se puede visualizar en la Figura 2.60, la cual dispone de un botón que permite volver a activar al usuario en el prototipo. En caso de presionar dicho botón, el usuario se activa y pasa a la lista de usuarios o clientes registrados según corresponda y se le permitirá acceder al prototipo.

Figura 2.60. Lista de usuarios eliminados

Configuración de la cuenta

Todos los usuarios son capaces de cambiar sus datos personales o su contraseña cuando lo requieran, seleccionando la opción `Cuenta` en el menú de navegación principal. La Figura 2.61 muestra el formulario que permite editar la información del usuario.

Figura 2.61. Vista para editar los datos de usuario

Recuperar la contraseña

La vista de *login* cuenta con un enlace que permite recuperar la contraseña, al hacer clic sobre el enlace se muestra una vista como la de la Figura 2.62, la cual solicita el ingreso de un correo electrónico registrado para poder recuperar la contraseña. El modelo utilizado para realizar esta acción se denomina `RecoverPasswordViewModel` y cuenta únicamente con un atributo denominado `Email`.

Figura 2.62. Vista para recuperar contraseña

Una vez ingresado el correo electrónico y presionando el botón para recuperar la contraseña, se valida que el correo corresponda a uno registrado en el sistema, después se envía un correo electrónico con un enlace compuesto por la URL, el id de usuario y un *token* generado, y al hacer clic sobre el enlace se presenta la vista para cambiar la contraseña. En la Figura 2.63 se muestra un ejemplo del correo enviado y en la Figura 2.64 se muestra la vista que permite cambiar la contraseña.

Figura 2.63. Correo de recuperación de contraseña

Figura 2.64. Vista para resetear la contraseña

La Figura 2.65 muestra el método de acción `ResetPassword` del controlador `Account`, que se encarga de realizar el proceso de cambio de contraseña por medio del método de HTTP `POST`, la acción es asíncrona y acepta como argumento un modelo `ResetPasswordViewModel` (línea 307) que contiene atributos como el nombre de usuario, la contraseña nueva y un *token* que forma parte del enlace en el correo enviado previamente al usuario. Por medio del nombre de usuario se obtiene el usuario (líneas 309-

310) y se valida que este exista, si no existe se muestra el mensaje “usuario no encontrado”, caso contrario, si existe el usuario por medio de una operación `IdentityResult` y del método de la interfaz `IUserHelper` denominado `ResetPasswordAsync` se realiza el cambio de contraseña y se muestra un mensaje en caso de ser exitosa la operación (líneas 310-321).

```
303     [HttpPost]
304     public async Task<IActionResult> ResetPassword(ResetPasswordViewModel modelo)
305     {
306         Usuario usuario = await _userHelper.ObtenerUserAsync(modelo.UserName);
307         if (usuario != null)
308         {
309             IdentityResult resultado = await _userHelper.ResetPasswordAsync(usuario,
310                 modelo.Token, modelo.NewPassword);
311             if (resultado.Succeeded)
312             {
313                 ViewBag.Message = "Clave reseteada exitosamente";
314                 return View();
315             }
316             else
317             {
318                 ViewBag.Message = "Error la clave no cuenta con los parámetros de seguridad";
319                 return View(modelo);
320             }
321         }
```

Figura 2.65. Método de acción `ResetPassword`

2.2.6.6. GESTIÓN DE PRODUCTOS

Lista de Productos

En la Figura 2.66 se muestra la vista de la lista de productos que ofrece el restaurante que se presenta al usuario administrador, como se puede apreciar la misma cuenta con los botones y opciones necesarias para realizar operaciones CRUD de los productos; además, la lista cuenta con la información más relevante de cada producto como el nombre, el precio, la descripción, el tipo, la imagen, entre otros. El botón `Tipos de Productos` se utiliza para realizar operaciones CRUD de los distintos tipos de productos que ofrece el restaurante. Para la eliminación de un producto se utiliza un atributo del tipo `bool` denominado `IsDeleted` de manera que el producto no se elimine de la base de datos, en su lugar se emplea dicho atributo para que no se muestre al usuario en el menú principal. Los productos que son marcados con este atributo pueden ser presentados mediante el botón `Productos Eliminados`. La interfaz de usuario que presenta el listado de productos eliminados también dispone de una opción para su recuperación como se indicó en el caso de usuarios eliminados.

Figura 2.66. Vista de la lista de productos.

Agregar Productos

En la Figura 2.67 se muestra la clase ViewModel denominada `ProductoViewModel` que hereda de la clase `Producto` y que se utiliza para escoger un tipo de producto de una lista (líneas 17-23), para cargar la imagen (líneas 25-27) y para dar el manejo adecuado de los decimales del precio del producto (líneas 29-34).

```
14  public class ProductoViewModel:Producto
15  {
16      //Seleccioar un tipo
17      [Display(Name = "Tipo")]
18      [Range(1, int.MaxValue, ErrorMessage = "Debe seleccionar un tipo.")]
19      [Required]
20      public int IdTipo { get; set; }
21
22      //La lista desplegable con los tipos de productos
23      public IEnumerable<SelectListItem> TipoProductos { get; set; }
24
25      [Display(Name = "Imagen (Formatos '.jpg', '.jpeg' o '.png')")]
26      [AllowedExtensions(new string[] { ".jpg", ".jpeg", ".png" })]
27      public IFormFile ImagenFile { get; set; }
28
29      [Display(Name ="Precio")]
30      [MaxLength(12)]
31      [RegularExpression(@"^\d+([\.\,]?\d+)?$",
32          ErrorMessage = "Solo poner números y (.) o (,) para poner decimales")]
33      [Required(ErrorMessage ="Debe ingresar un precio")]
34      public string StringPrecio { get; set; }
35  }
```

Figura 2.67. Clase `ProductoViewModel`

En la Figura 2.68 se muestra el formulario ocupado que permite agregar un nuevo producto, el mismo cuenta con los campos necesarios para llenar su información, para escoger el tipo de producto se utiliza una lista desplegable la cual se obtiene utilizando uno de los métodos de la interfaz `ICombosHelper`, el campo de precio admite valores decimales con coma o con punto, y permite ingresar una imagen en el formato correcto. Finalmente, el formulario cuenta con dos botones uno para volver al menú principal y otro para guardar los cambios.

Figura 2.68. Formulario para agregar un nuevo producto

En el segmento de código que se encuentra en la Figura 2.69 se muestra la vista del formulario para agregar el producto, la etiqueta `<form>` que está entre las líneas 14 a la 23 hace referencia a los campos que deben ser llenados para crear el nuevo producto. Como se aprecia en la línea 16 se utiliza una vista parcial denominada `_Producto` que posee los campos para agregar el producto, esto se realizó con la finalidad de poder reutilizar la misma vista parcial en la vista que se encarga de editar la información del producto, ya que ambos formularios son similares. Las etiquetas `<div>` se utilizan para estructurar de una mejor manera el formulario mediante la utilización de clases de Bootstrap.

```
11  <div class="row">
12      <div class="col-md-8 offset-md-2">
13          <form asp-action="Create" enctype="multipart/form-data">
14              <div asp-validation-summary="ModelOnly" class="text-danger"></div>
15
16              <partial name="_Producto" />
17
18              <div class="form-group offset-4">
19                  <a asp-action="Index" class="btn btn-success">
20                  <i class="fas fa-arrow-left"></i>   Volver</a>
21                  <button type="submit" class="btn btn-primary">
22                  <i class="fas fa-save"></i>   Guardar</button>
23              </div>
24          </form>
25      </div>
26  </div>
```

Figura 2.69. Segmento de código de la vista para agregar un nuevo producto

Información del producto

En la Figura 2.70 se muestra un ejemplo de la vista que contiene la información detallada de un producto, además posee un botón para redirigir al usuario hacia la vista que permite editar la información del producto.

Figura 2.70. Vista de la información de producto.

Para mostrar la información del producto, en el controlador respectivo se creó un método de acción asíncrono-denominada `Details` que se visualiza en la Figura 2.71. Este método de acción recibe el `id` del producto que se obtiene de la lista de productos (línea 39), se valida si el `id` es `null`, si no lo es se realiza el proceso de búsqueda en la base de datos

(líneas 45-48), es necesario validar que el producto exista y de existir se lo envía a la vista respectiva (líneas 48-52).

```
39      public async Task<IActionResult> Details(int? id)
40      {
41          if (id == null)
42          {
43              return NotFound();
44          }
45          Producto producto = await _context.Productos
46              .Include(p=>p.TipoProducto)
47              .FirstOrDefaultAsync(m => m.Id == id);
48          if (producto == null)
49          {
50              return NotFound();
51          }
52          return View(producto);
53      }
```

Figura 2.71. Método de acción `Details`

Gestión del menú del día

La gestión del menú del día se encuentra a cargo del usuario administrador, el menú del día hace referencia al menú del almuerzo o de la merienda que ofrece el restaurante, el mismo se encuentra conformado por una sopa, un segundo y un jugo, y una vez definido será presentado a los clientes.

En la Figura 2.72 se muestra la lista de las opciones que se pueden agregar al menú del día, en la imagen se visualiza la lista con todas las opciones con las que se puede armar el menú, cada opción tiene un nombre y una categoría. Cada opción cuenta con tres botones, uno para añadir la opción al menú del día, otro para editar la información y otro botón para eliminar la opción de la lista. En la parte superior existe un botón para agregar una nueva opción a la lista y otro para ver el menú del día.

Figura 2.72. Vista con las opciones para agregar al menú del día

Para realizar la gestión del menú del día se utilizó funciones de JavaScript que hacen uso de AJAX y mediante el método `POST` envía información al servidor para que este realice la acción respectiva y se ejecute el método de acción correspondiente del controlador.

2.2.6.7. GESTIÓN DE LAS MESAS

La gestión de las mesas está a cargo del usuario administrador y consiste en visualizar la lista de las mesas con su respectivo número y su estado para saber que mesas se encuentran libres u ocupadas, además, se debe permitir agregar o eliminar una mesa en caso de ser necesario.

En la Figura 2.73 se muestra la vista con la lista de mesas, cada una de las mesas posee una columna con el número respectivo que permite identificarlas, además, se indica si se encuentran libres u ocupadas. En la parte superior se encuentran los botones que permiten agregar o eliminar una mesa, al presionar el botón agregar se muestra una ventana como la de la Figura 2.74, en la cual se pregunta si se desea agregar una nueva mesa con el último número de la lista incrementado en uno; caso contrario, si se presiona el botón eliminar, se muestra una ventana como la de la Figura 2.75, en la cual se pregunta si se desea eliminar la última mesa de la lista.

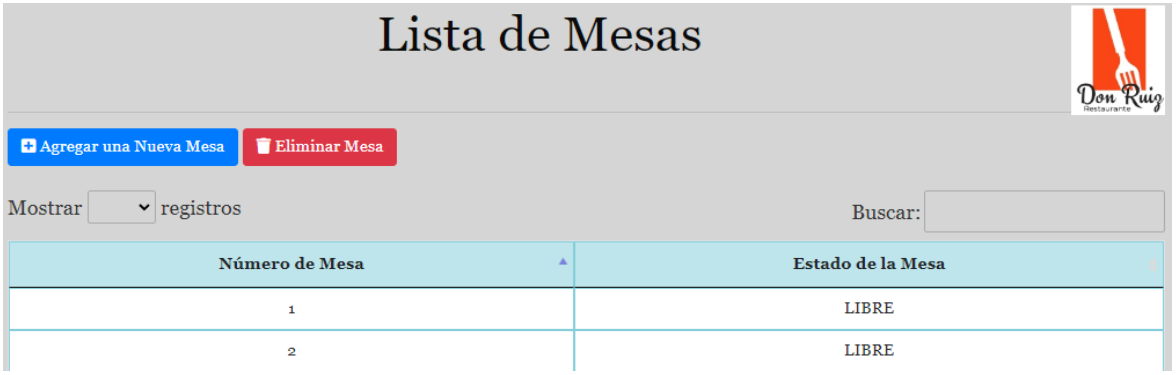


Figura 2.73. Vista principal de las mesas

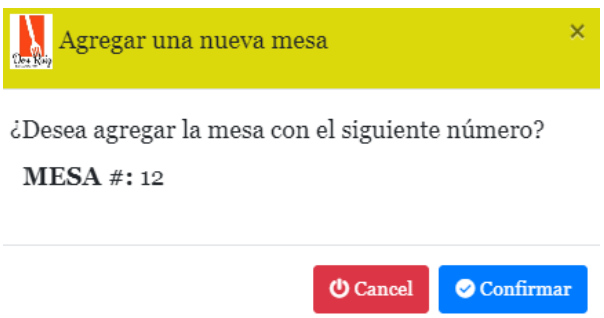


Figura 2.74. Ventana para agregar una nueva mesa

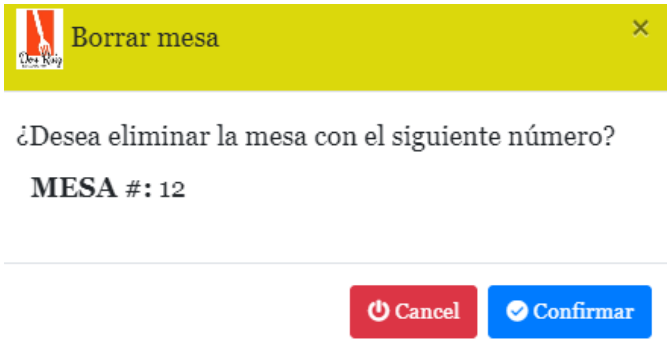


Figura 2.75. Ventana para eliminar una mesa

2.2.6.8. INGRESO Y GESTIÓN DE PEDIDOS

Ingreso del pedido

Los clientes inician sesión en el prototipo y en la pestaña menú pueden visualizar el menú principal del restaurante con los productos disponibles. En la Figura 2.76 se presenta la vista del menú principal que se le muestra a los clientes, cada producto posee un botón verde que se utiliza para mostrar una ventana con la información del producto e ingresar la cantidad requerida para agregarlo al pedido. El botón `Ver Pedido` se utiliza para mostrar los productos agregados al pedido antes finalizar el pedido y el botón amarillo se utiliza para presentar el menú del día.

Figura 2.76. Menú principal

En la Figura 2.77 se muestra la ventana para agregar un producto al pedido, donde se aprecia la información del producto que se carga automáticamente como el Id, el nombre, la descripción y el precio, en tanto que el cliente debe ingresar la cantidad de productos que desea agregar al pedido, también, opcionalmente se puede agregar un comentario acerca del producto. Además, la ventana posee los botones necesarios para confirmar el ingreso del producto en el pedido o cancelarlo.

Figura 2.77. Ventana para ingresar el producto al pedido

Al presionar el botón `Ver Pedido` del menú principal se muestra una vista como la de la Figura 2.78, donde se visualiza el pedido con todos los ítems agregados al pedido con los respectivos botones para editar o eliminar del pedido, además se muestra el valor total del pedido y unos botones para ir a la vista de finalización del pedido o para regresar al menú y seguir realizando el pedido.

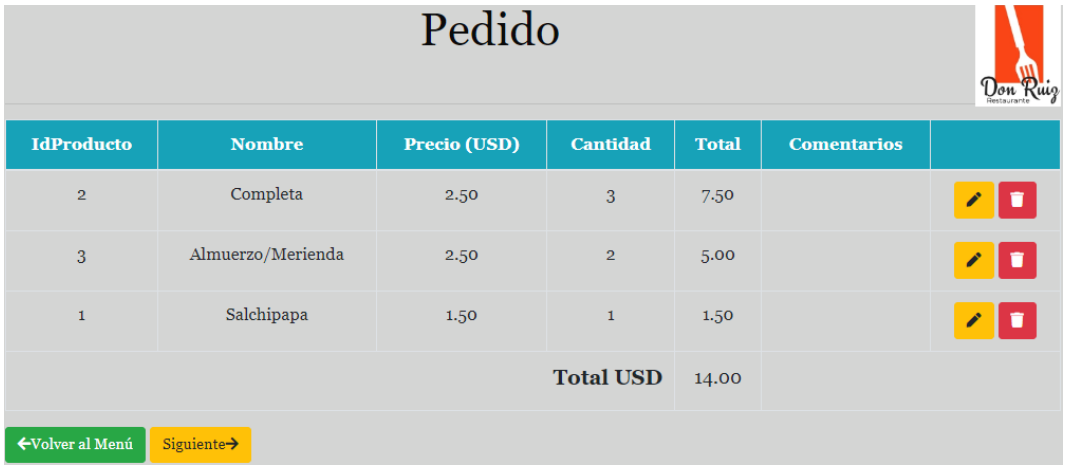


Figura 2.78. Vista del pedido

En la Figura 2.79 se muestra la vista que contiene el resumen del pedido que cuenta con el detalle del mismo, además permite indicar si el pedido es para llevar o para la mesa, y permite indicar el número de la mesa. Al final se encuentra el botón para finalizar el pedido.

Figura 2.79. Resumen del pedido

Al ser HTTP un protocolo sin estado, es decir que maneja las peticiones como mensajes independientes que no guardan ningún valor o información, para la codificación del ingreso de los pedidos se optó por utilizar sesiones HTTP y almacenar el pedido del usuario en la memoria caché del cliente mientras el mismo continúa navegando por la aplicación web. Una vez que finaliza el pedido, se envía al servidor, el cual se encarga con la lógica implementada de procesar el mismo.

Para poder utilizar sesiones HTTP en ASP .NET Core es necesario especificar su uso en el contenedor de servicios mediante el archivo `Startup.cs` como lo muestra la Figura 2.80. Se puede configurar el tiempo de duración de la sesión, para este prototipo se optó por dejarlo en el valor que viene por defecto que es de 20 minutos.

```
//Se define el uso de sesiones HTTP
services.AddSession();
//Se define un almacén de memoria caché para almacenar información
services.AddDistributedMemoryCache();
```

Figura 2.80. Definición del uso de sesiones en el contenedor de servicios

En la Figura 2.81 se muestra los dos de los métodos definidos en el controlador `Pedidos` para realizar el ingreso de pedidos, uno denominado `GetSession` utilizado para obtener el nombre de la sesión y su contenido, y otro método denominado `SetSession` que se utiliza para actualizar la información de la sesión.

```
//Obtener la sesión
public string GetSession()
{
    return HttpContext.Session.GetString("sesion");
}
//Editar valores de la sesión
public void SetSession(object obj)
{
    HttpContext.Session.SetString("sesion", JsonConvert.SerializeObject(obj));
}
```

Figura 2.81. Métodos para el manejo de sesiones

Gestión del pedido

Una vez confirmado el pedido por parte del cliente, el mismo se envía al servidor para que tanto los empleados como administradores pueden visualizarlo y comiencen a gestionarlo. En la Figura 2.82 se puede visualizar la lista de los pedidos del día a medida que van ingresando al sistema, cada pedido contiene la información general como el estado del pedido, tipo de pedido, cliente, total, etc. Además, cada pedido cuenta con un botón para visualizar el detalle de este en una vista adecuada. En la parte superior existen dos botones uno denominado `Historial de pedidos` que se utiliza para visualizar el historial de todos los pedidos ingresados y el otro botón se denomina `Ventas` y permite visualizar las ventas del día o el de una fecha específica.

Figura 2.82. Vista de los pedidos del día

Para presentar los pedidos que se generan en el día, en el método de acción correspondiente del controlador `Pedidos` se emplean filtros, como se indica en la Figura 2.83, en la cual se aprecia que se devolverá una vista que incluye el detalle de los pedidos, los productos, el usuario y la mesa (líneas 278-282), además solo se incluirán los pedidos que se hayan generado en la fecha actual (línea 283) y con esta información se genera una lista (línea 284).

```
276     public async Task<IActionResult> PedidosGeneral()
277     {
278         return View(await _context.Pedidos
279                 .Include(p => p.DetallePedidos)
280                 .ThenInclude(dp=>dp.Producto)
281                 .Include(p => p.Usuario)
282                 .Include(p => p.Mesa)
283                 .Where(p => p.Fecha.Date == DateTime.Now.Date)
284                 .ToListAsync());
285     }
```

Figura 2.83. Método de acción `PedidosGeneral` utilizado para la vista de pedidos

La vista de pedidos del día y la del historial se actualizan automáticamente cada 60 segundos, de forma de poder visualizar los nuevos pedidos que ingresan, para lo cual se implementaron en las vistas mencionadas la función de JavaScript que se muestra en la Figura 2.84. La función `actualizar` permite cargar de nuevo la URL actual (línea 112) y la sentencia `setInterval` (línea 114) se utiliza para llamar la función `actualizar` cada 60 segundos.

```
112     function actualizar() { location.reload(true); }
113     //Función para actualizar cada 60 segundos(60000 milisegundos)
114     setInterval("actualizar()", 60000);
```

Figura 2.84. Función para recargar automáticamente la vista

Para realizar la gestión de los pedidos, en cada pedido se incluyen botones para los distintos tipos de usuario que al presionarlos van a realizar el cambio de estado. Por ejemplo, en la Figura 2.85 se muestra un detalle de un pedido donde entre otras cosas se puede apreciar que el estado se encuentra `pendiente` y que posee un botón denominado `preparando` que al presionar cambia de estado como lo muestra en la Figura 2.86 en la cual se presenta el cambió a `preparando` y aparece otro botón para cambiar al estado a `entregado`.

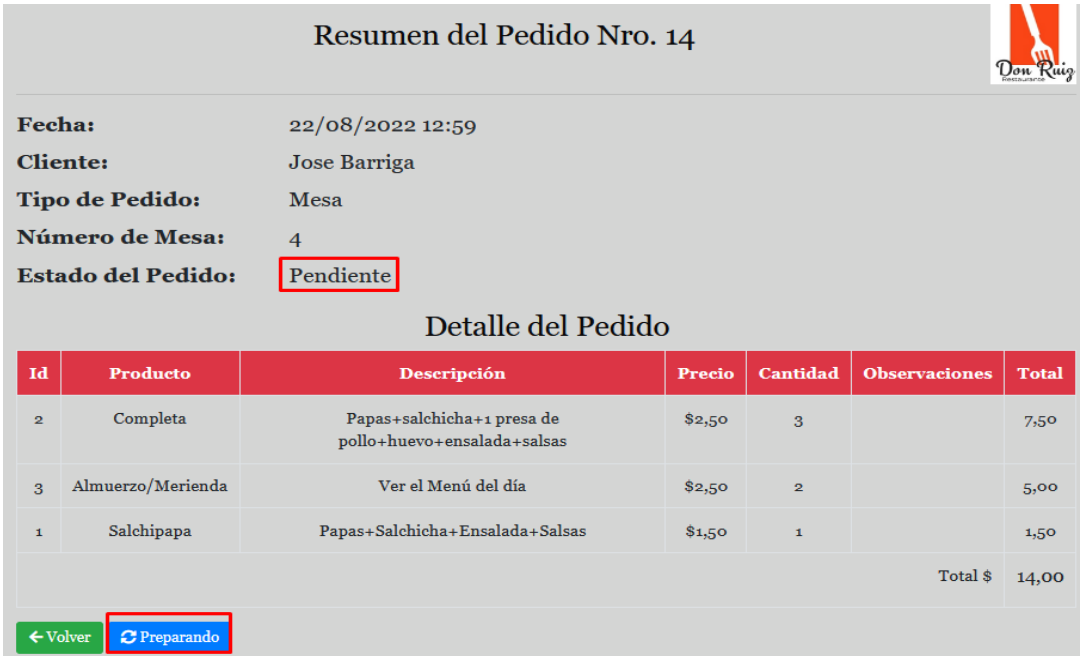


Figura 2.85. Ejemplo 1 del cambio de estado de un pedido

Figura 2.86. Ejemplo 2 del cambio de estado de un pedido

Cuando el pedido se encuentra en el estado `entregado`, el cliente por medio de un botón como se muestra en la Figura 2.87, puede solicitar el cobro de este cambiando el estado del pedido de `entregado` a `cobro`.

Figura 2.87. Ejemplo 3 del cambio de estado de un pedido

Al pasar el pedido a estado de `cobro` solo un administrador puede confirmar que se ha realizado su pago, por lo cual en la cuenta del administrador se visualiza una vista como la de la Figura 2.88, que posee un botón denominado `Confirmar el pago` que permite cambiar el estado de `cobro` a `pagado`, agregando el valor del total a la venta, liberando la mesa, en caso de que no haya más pedidos activos asociados a la misma y añadiendo el pedido al historial, finalizando así la gestión de este pedido.

Figura 2.88. Ejemplo 4 del cambio de estado de un pedido

Para el cambio de estados de los pedidos se utiliza AJAX, mediante el cual a medida que se van presionando los botones de estado se comunica con el servidor y mediante un método de acción que acepta como argumento el ID del pedido y utilizando sentencias `if` va cambiando de estado al pedido según corresponda, como lo muestra la Figura 2.89.

```
304             if (pedido.EstadoPedido == EstadoPedido.Pendiente)
305             {
306                 pedido.EstadoPedido = EstadoPedido.Preparando;
307             }
308             else if (pedido.EstadoPedido == EstadoPedido.Preparando)
309             {
310                 pedido.EstadoPedido = EstadoPedido.Entregado;
311             }
312             else if (pedido.EstadoPedido == EstadoPedido.Entregado)
313             {
314                 pedido.EstadoPedido = EstadoPedido.Cobro;
315             }
```

Figura 2.89. Segmento de código utilizado para el cambio de estados del pedido

En la Figura 2.90 se muestra el segmento de código utilizado para mostrar los botones de estado, en el ejemplo se valida el rol de usuario que está autenticado en el sistema (líneas 107-108) y dependiendo del estado del pedido se muestra el botón preparando o el botón entregado (líneas 110-122). Para los botones de solicitar el cobro y confirmar el pago se utiliza la misma lógica.

```
107     @if (User.Identity.IsAuthenticated && User.IsInRole("Administrador") ||
108         User.IsInRole("Empleado"))
109     {
110         <a asp-action="PedidosGeneral" class="btn btn-success">
111            <i class="fas fa-arrow-left"></i>   Volver</a>
112         if (Model.EstadoPedido == Restaurante
113             .Common.Enumeraciones.EstadoPedido.Pendiente)
114         {
115             <button id="btnGestionarPedido" class="btn btn-primary">
116            <i class="fas fa-sync-alt"></i>   Preparando</button>
117         }
118         if (Model.EstadoPedido == Restaurante
119             .Common.Enumeraciones.EstadoPedido.Preparando)
120         {
121             <button id="btnEntregarPedido" class="btn btn-primary">
122            <i class="fas fa-check-circle"></i>   Entregado</button>
123         }
124     }
```

Figura 2.90. Segmento de código utilizado para mostrar los botones de estado en la vista

2.2.6.9. RESUMEN DE LAS VENTAS

Los pedidos almacenados en la base de datos contienen toda la información necesaria para poder desplegar un resumen de las ventas generadas por el restaurante de una manera adecuada, por lo cual para poder realizar este proceso se optó por crear una clase ViewModel denominada `VentasViewModel`, que se muestra en la Figura 2.91. Esta clase contiene la información del producto, la fecha, la cantidad de productos vendidos y el valor total que ha generado ese producto en esa fecha.

```
10    public class VentasViewModel
11    {
12        [DisplayFormat(DataFormatString = "{0:dd/MM/yyyy}")]
13        [Display(Name = "Fecha")]
14        public DateTime Fecha { get; set; }
15
16        public Producto Producto { get; set; }
17
18        public int Cantidad { get; set; }
19
20        public decimal TotalProducto { get; set; }
21    }
```

Figura 2.91. Clase ViewModel `VentasViewModel`

Posteriormente, se procedió a crear una interfaz denominada `IVentasHelper`, que se muestra en la Figura 2.92, la cual implementa un método asíncrono llamado `GetVentasAsync` que recibe una fecha y recupera la lista con toda la información de las ventas realizadas en esa fecha específica.

```
public interface IVentasHelper
{
    Task<IList<VentasViewModel>> GetVentasAsync(DateTime fecha);
}
```

Figura 2.92. Interfaz `IVentasHelper`

En la Figura 2.93 se visualiza el método de acción `Ventas` del controlador, el cual acepta una fecha como argumento (línea 344), si no se ha incluido una fecha se emplea la fecha actual (línea 349), caso contrario se recupera la fecha sin la hora (línea 353), después, se procede a obtener la lista con la información de las ventas para retornar una la vista con esta información (líneas 355-356).

```
344     public async Task<IActionResult> Ventas(string fecha)
345     {
346         DateTime fecha2;
347         if (fecha == null)
348         {
349             fecha2 = DateTime.Now.Date;
350         }
351         else
352         {
353             fecha2 = Convert.ToDateTime(fecha);
354         }
355         IList<VentasViewModel> ventas = await _ventasHelper.GetVentasAsync(fecha2);
356         return View(ventas);
```

Figura 2.93. Método de acción `Ventas`

En la Figura 2.94 se muestra la vista que se encarga de desplegar la información de las ventas con base a la fecha ingresada, la lista muestra únicamente los productos que se han vendido con su respectivo ID, nombre, cantidad y total que se ha generado para cada producto. En la parte final de la tabla se muestra el valor total de ventas generadas en la fecha indicada.

Id	Nombre de Producto	Cantidad	Total (USD)
5	Apanado	1	4,00
4	Choripapa	1	2,25
9	Colas Personales	6	3,00
2	Completa	2	5,00
7	Papipollo	5	8,75
8	Pollo Entero	3	45,00
1	Salchipapa	6	9,00
	Total de Ventas USD		77,00

Figura 2.94. Vista de las ventas del día

3. PRUEBAS Y RESULTADOS OBTENIDOS

En el presente capítulo se muestran los resultados de las pruebas realizadas en cada una de las vistas para verificar el correcto funcionamiento del prototipo conforme a los requerimientos.

3.1. PRUEBAS DE FUNCIONAMIENTO

3.1.1. AUTENTICACIÓN EN EL SISTEMA

A continuación, se muestran las distintas validaciones necesarias para realizar el proceso de autenticación de una manera correcta, en la Figura 3.1 se muestran las validaciones en caso de que uno o varios campos se encuentren vacíos. En la Figura 3.2, se valida que las credenciales de usuario sean correctas, caso contrario se muestra un mensaje de credenciales incorrectas como el del recuadro rojo.

Figura 3.1. Validación de cambios vacíos

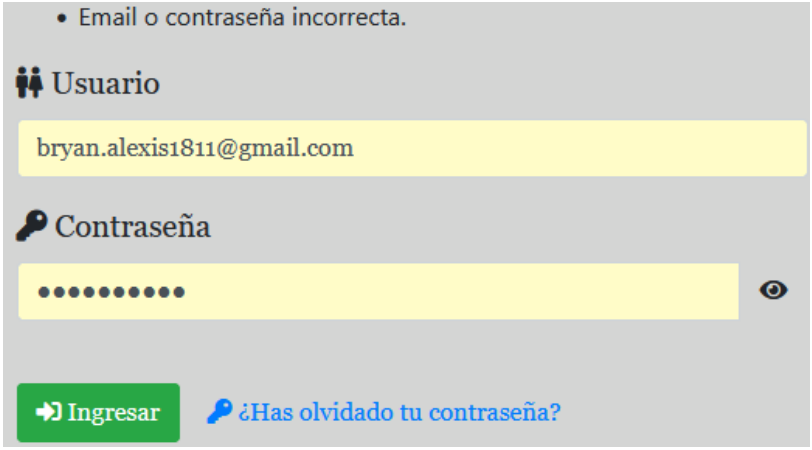


Figura 3.2. Validación de las credenciales de usuario

3.1.2. RECUPERACIÓN DE CONTRASEÑA

Para poder recuperar la contraseña en la vista de inicio de sesión se encuentra una opción para poder recuperar la misma. La vista cuenta con las validaciones necesarias para garantizar que la cuenta de correo ingresada sea válida y esté registrada en el sistema, caso contrario se muestra un mensaje de error como se aprecia en la Figura 3.3.

Figura 3.3. Validación necesaria para recuperar la contraseña

Si se ingresa una dirección de correo válida y que se encuentre registrada se envía un correo electrónico con un enlace de verificación para poder ingresar y cambiar la contraseña de acceso.

Figura 3.4. Correo de recuperación de contraseña

La vista encargada de cambiar la clave cuenta con las validaciones necesarias para un correcto funcionamiento del prototipo, en la Figura 3.5 se visualiza un ejemplo de lo que

sucede cuando el campo de la clave no coincide con el campo de confirmación de clave al momento de realizar el cambio.

Figura 3.5. Validaciones para cambiar la clave de acceso

3.1.3. AGREGAR PRODUCTO

En la Figura 3.6 se muestra el formulario para agregar los productos al sistema, el mismo cuenta con los campos y validaciones necesarias.

Figura 3.6. Funcionamiento del formulario para agregar un producto

Una vez que se presiona el botón para guardar la información del producto del cual se ha ingresado la información en los campos correspondientes y que la misma cumple con todas las validaciones establecidas, se agrega el nuevo producto a la lista de productos, como se muestra en la Figura 3.7.

Id	Nombre	Descripción	Estado	Precio (USD)	Tipo	Imagen
12	Papas de Balde	Papas+Salchicha+Salsas+Ensalada	Activo	$2,25	A la Carta	
10	Encebollado	Encebollado+chifles+canguil	Activo	$4,50	A la Carta	

Figura 3.7. Producto agregado correctamente al sistema

3.1.4. EDITAR PRODUCTO

Si se requiere editar la información de un producto se utiliza el formulario presentado en la Figura 3.8, que es similar al de agregar el producto. A nivel de se maneja el mismo tipo de información y validaciones, la diferencia que existe entre ambos es que en el formulario de editar producto se carga la información de cada producto en los respectivos campos y se visualiza la imagen de este.

Figura 3.8. Funcionamiento del formulario para editar un producto

Una vez hechos los cambios que el usuario considere necesarios los mismos se guardan y se presentarán en la lista como se muestra en la Figura 3.9.

8	Pollo Entero Don Ruiz	1 Pollo+5 porciones de papas+5 porciones de arroz+5 consomes+ensalada fresca	Activo	$20,00	A la Carta	

Figura 3.9. Producto editado correctamente en el sistema

3.1.5. ELIMINAR PRODUCTO

Para la realización de la operación que permite eliminar los productos se utilizan alertas como la presentada en la Figura 3.10, mediante la cual se muestra un mensaje para confirmar que se desea eliminar el registro seleccionado. En caso de confirmar la eliminación se muestra una alerta como la de la Figura 3.11.

Figura 3.10. Alerta para confirmar la eliminación del producto

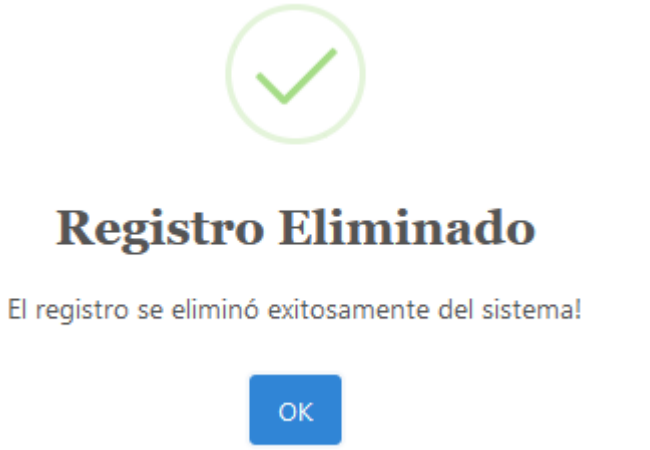


Figura 3.11. Alerta de registro eliminado

3.1.6. AGREGAR USUARIOS Y REGISTRAR CLIENTES

Un usuario administrador puede crear un usuario con rol empleado o administrador mediante el formulario presentado en la Figura 3.12, el cual cuenta con los campos para ingresar la información requerida, así como las validaciones respectivas. El administrador ingresa la clave que será asignada al usuario, y que será remitida por medio del correo de confirmación, como se visualiza en la Figura 3.13. Cuando el nuevo usuario inicie sesión el prototipo solicita el cambio de contraseña por seguridad.

Figura 3.12. Formulario para agregar un administrador o empleado

Figura 3.13. Correo de confirmación para un usuario administrador o empleado

En la Figura 3.14 se muestra la solicitud de cambio de contraseña una vez que el usuario nuevo se ha autenticado en el prototipo por primera vez. En la Figura 3.15 se muestra la lista de usuarios en la cual se observa al usuario que fue recientemente creado por lo que se concluye que el proceso de creación fue exitoso.

Figura 3.14. Cambio de clave en el primer inicio de sesión

Figura 3.15. Lista de Usuarios

El proceso para el registro de clientes nuevos es muy parecido al de usuarios, la diferencia radica en que cada cliente puede registrarse por su cuenta, para lo cual en la vista de inicio de sesión al presionar el botón de registro se presenta un formulario similar al de la Figura 3.12, una vez completo el formulario y enviada la información al servidor, el prototipo envía el correo de confirmación y el cliente puede acceder al sistema.

3.1.7. ELIMINAR UN USUARIO

Un administrador puede eliminar un usuario sin importar el rol de usuario que este posea al presionar el botón correspondiente. En la Figura 3.16 se muestra la alerta que solicita la confirmación para eliminar un usuario; en caso de confirmar se mostrará una alerta como la de la Figura 3.17 y el usuario se eliminará de la lista de usuarios registrados en el sistema.

Figura 3.16. Alerta para confirmar la eliminación del usuario

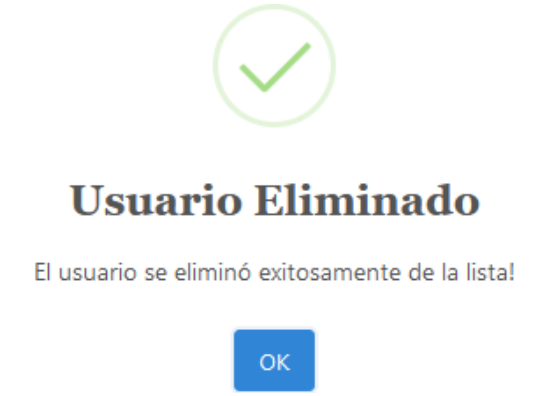


Figura 3.17. Alerta de usuario eliminado

3.1.8. PEDIDOS

Los clientes pueden ingresar pedidos en el apartado de menú respectivo, luego de lo cual pueden seleccionar los productos que desean agregar a su pedido. Cada vez que se selecciona un producto se muestra una ventana modal que cuenta con una validación que garantiza que en el campo de cantidad se ingrese un valor correcto, el resto de información como el nombre o el precio se carga automáticamente y el campo de comentarios no es un campo requerido, pero el cliente lo puede emplear para agregar algún comentario respecto al producto. En la Figura 3.18 se muestra un ejemplo en el caso de que se intente

ingresar un valor inválido en el campo `cantidad`. Si el valor ingresado en el campo `cantidad` es correcto se muestra una alerta como la de la Figura 3.19 en la que indica que el producto se ha agregado exitosamente al pedido.

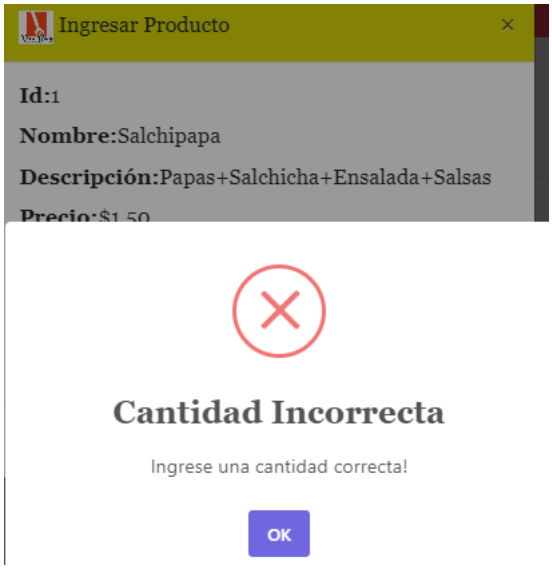


Figura 3.18. Alerta en caso de ingresar una cantidad incorrecta en el formulario de ingreso de productos

Figura 3.19. Alerta que se muestra al ingresar el pedido correctamente

Una vez que el producto se ha agregado exitosamente al pedido se presentará una vista como la de la Figura 3.20, donde se puede apreciar los productos ingresados al pedido con la cantidad, el precio, el valor total del pedido, etc. Cada fila de la lista cuenta con botones para editar o eliminar el producto del pedido, además, es importante mencionar que la vista de pedidos cuenta con las validaciones necesarias para evitar que en el sistema ingrese pedidos sin productos, si el sistema detecta que no existen productos bloquea el botón siguiente y muestra un mensaje que indica que el pedido no tiene productos.

Figura 3.20. Vista del pedido

Una vez que se finaliza el pedido, desde el cliente se envía el pedido para mostrarlo a todos los usuarios y que pueda ser gestionado adecuadamente; además se muestra la vista del historial de pedidos. Es importante mencionar que los clientes únicamente podrán ver el historial de sus pedidos y no los de otros clientes, en cambio los demás usuarios como el usuario administrador y empleado pueden ver los pedidos de todos los clientes.

En la Figura 3.21 se muestra el historial que se presenta en el lado del cliente donde se puede apreciar el último pedido que ingresó junto con todos los pedidos que ha realizado dicho cliente, cada pedido cuenta con toda la información necesaria para gestionar el pedido por parte del personal del restaurante. El botón amarillo permite visualizar el detalle del pedido desplegando una vista como la de la Figura 3.22 donde se puede apreciar todo el detalle del pedido como la información del cliente, información del pedido y el costo total del mismo.

Figura 3.21. Historial de pedidos visto como cliente

Resumen del Pedido Nro. 16

Fecha: 24/08/2022 09:24
Cliente: Jose Barriga
Tipo de Pedido: Mesa
Número de Mesa: 5
Estado del Pedido: Pendiente

Detalle del Pedido

Id	Producto	Descripción	Precio	Cantidad	Observaciones	Total
1	Salchipapa	Papas+Salchicha+Ensalada+Salsas	$1,50	3		4,50
2	Completa	Papas+salchicha+1 presa de pollo+huevo+ensalada+salsas	$2,50	2		5,00
3	Almuerzo/Merienda	Ver el Menú del día	$2,50	1		2,50
					Total $	12,00

Figura 3.22. Detalle del pedido

En la Figura 3.23 se muestra el historial de pedidos que se presenta al usuario administrador, como se puede visualizar es muy similar al de la Figura 3.21, sin embargo, posee información adicional como el nombre del cliente. El estado de los pedidos va cambiando conforme se va gestionando cada uno de los mismos. El botón amarillo se utiliza para poder ver el detalle de los productos solicitados en el pedido.

Pedidos del Día

Historial de pedidos | Ventas

Mostrar 25 registros | Buscar:

Id	Fecha	Número de Mesa	Tipo de Pedido	Estado del Pedido	Cliente	Total	
16	24/08/2022 09:24	5	Mesa	Pendiente	Jose Barriga	12,00	

Figura 3.23. Historial de pedidos visto como administrador

3.1.9. CUADROS DE BÚSQUEDA

La mayoría de las vistas que despliegan información en una lista poseen cuadros de búsqueda que permiten filtrar la información de una manera más sencilla. En la Figura 3.24 se muestra un ejemplo donde se encuentra la lista de los productos y en la esquina superior

derecha se encuentra el cuadro de búsqueda que permite al usuario ingresar algún parámetro para la búsqueda que desea realizar.

Figura 3.24. Ejemplo de búsqueda

3.1.10. INFORMACIÓN DE LAS VENTAS

En la Figura 3.25 se muestra la información de ventas del día, en tanto que en la Figura 3.26 se muestra la información de las ventas que se generaron en una fecha específica, para mostrar que el funcionamiento es adecuado en la esquina inferior de la imagen se muestra la fecha en la que se realizó la captura de la imagen.

Figura 3.25. Información de las ventas del día

Figura 3.26. Información de las ventas de una fecha específica

3.1.11. ACCESOS NO AUTORIZADOS Y PÁGINA DE ERROR

En la Figura 3.27 se muestra la vista que se presenta a los usuarios cuando no se encuentran autorizados para realizar una acción, en el ejemplo como tal un usuario autenticado como cliente está tratando de ingresar a la vista que despliega la información de las ventas a la cuál únicamente puede acceder un administrador por lo cual el prototipo indica un error 401.

Figura 3.27. Vista de acceso no autorizado

En la Figura 3.28 se muestra la vista que se presenta a los usuarios cuando intentan acceder a un recurso que no existe en la aplicación, por lo cual el prototipo genera un error 404.

Figura 3.28. Vista de error

3.1.12. PRUEBAS EN DISTINTOS NAVEGADORES

Los navegadores Mozilla Firefox, Google Chrome, Microsoft Edge y Brave se utilizaron para realizar las distintas pruebas de funcionamiento y en todos se comprobó un funcionamiento adecuado del prototipo. En la Figura 3.29 se muestra el prototipo en el navegador Google Chrome y en la Figura 3.30 se muestra el prototipo en el navegador Microsoft Edge.

Figura 3.29. Prototipo en Google Chrome

Figura 3.30. Prototipo en Microsoft Edge

3.2. RESULTADOS DE LAS ENCUESTAS

En la presente sección se muestran los resultados obtenidos de la encuesta que se aplicaron a 6 usuarios del prototipo. El modelo de encuesta que se aplicó se encuentra en el ANEXO C.

La pregunta 1 tiene como objetivo conocer si los usuarios pudieron acceder correctamente al prototipo mediante el método de autenticación implementado. Se observa en la Figura 3.31 que el 100% encuestados pudo acceder correctamente al sistema.

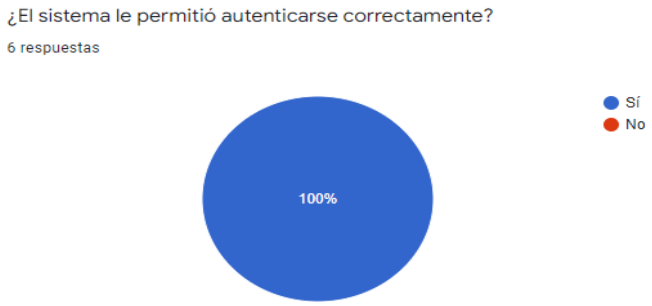


Figura 3.31. Resultados de la pregunta 1

La pregunta 2 está orientada a conocer si el mecanismo que permite recuperar la contraseña. Se puede apreciar que de acuerdo con las respuestas (Ver Figura 3.32) no existió ningún inconveniente en la recuperación de contraseñas por parte de los usuarios.

Figura 3.32. Resultados de la pregunta 2

La pregunta 3 permite determinar si el prototipo realiza una gestión correcta de todos los usuarios y de acuerdo con los resultados (ver Figura 3.33) el 100% de los encuestados indican que pudieron realizar una gestión de usuarios.

Figura 3.33. Resultados de la pregunta 3

La pregunta 4 permite conocer si pudieron realizar de una manera adecuada las distintas operaciones CRUD de productos. El resultado de esta pregunta se presenta en la Figura 3.34.

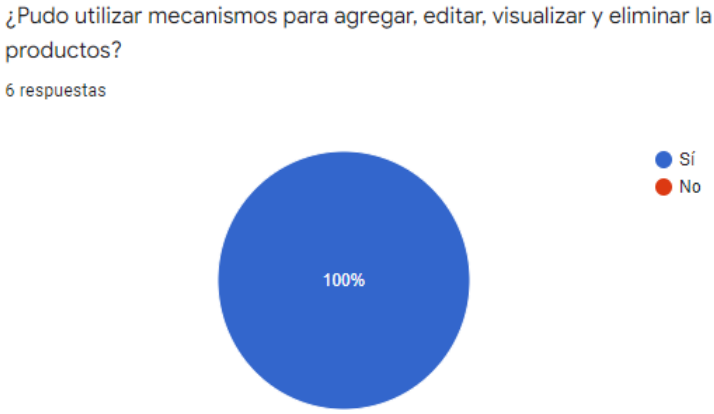


Figura 3.34. Resultados de la pregunta 4

La pregunta 5 indica si los encuestados pudieron realizar el proceso de búsqueda, utilizando un filtro de información en las respectivas vistas del prototipo. El resultado se presenta en la Figura 3.35.

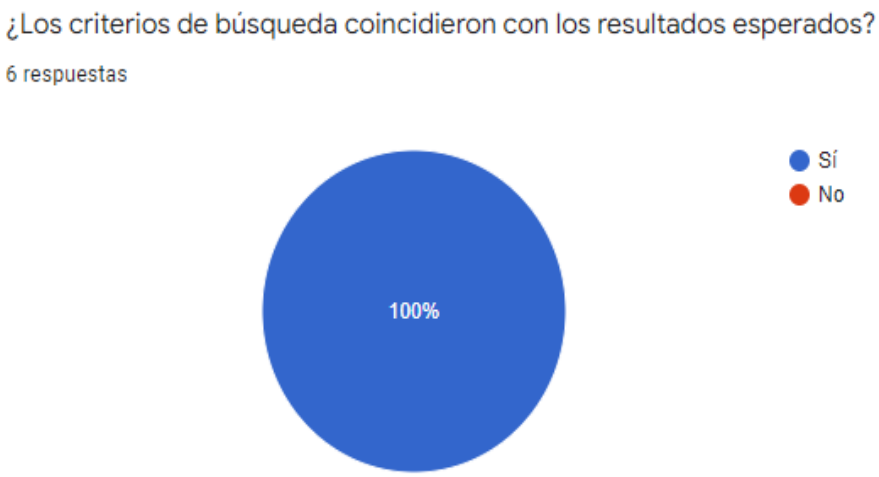


Figura 3.35. Resultados de la pregunta 5

La pregunta 6 está orientada a conocer si la funcionalidad implementada en el prototipo para el manejo del menú principal y si el menú del día del restaurante es adecuado de acuerdo con la función que cumple cada persona en el restaurante.

Conforme a los resultados obtenidos los cuales se presentan en la Figura 3.36, el 100% de los encuestados indican que pudieron ver el menú.

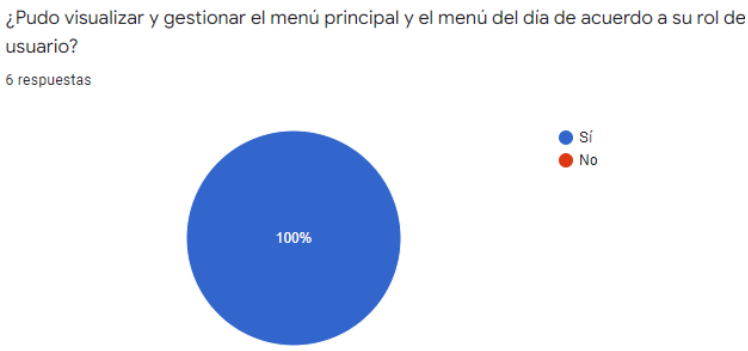


Figura 3.36. Resultados de la pregunta 6

La pregunta 7 permite conocer si el prototipo permite gestionar los pedidos de una manera correcta y adecuada acorde con los requerimientos levantados. Los resultados indican que ningún usuario tuvo problemas gestionando pedidos (ver Figura 3.37).

Figura 3.37. Resultados de la pregunta 7

La pregunta 8 está orientada a conocer si el prototipo de aplicación es amigable para el usuario o no, y de acuerdo con los resultados, un 66,7% de los encuestados considera que el manejo es muy fácil, en tanto, que el 33,3% indica que el manejo es fácil por lo que se puede concluir que la aplicación es amigable al usuario. El resultado se presenta en la Figura 3.38.

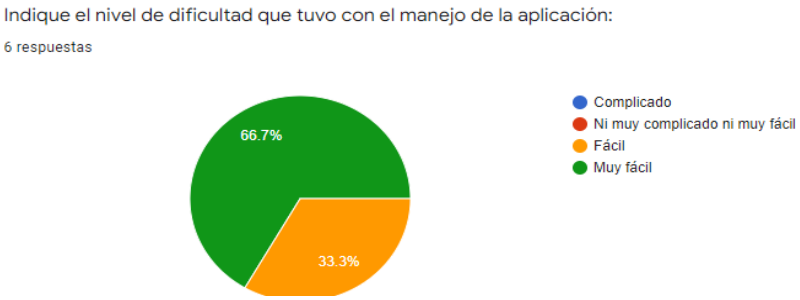


Figura 3.38. Resultados de la pregunta 8

La pregunta 9 permite conocer si la parte visual del prototipo es del agrado de los usuarios, conforme a los resultados que se presentan en la Figura 3.39 se indica que al 66,7% les agrada mucho, en tanto que un 33,3% indica que ni les agrada ni les desagrada.

Figura 3.39. Resultados de la pregunta 9

La pregunta 10 permite conocer si el prototipo ha cumplido con las expectativas de los usuarios, los resultados presentados en la Figura 3.40 indican que efectivamente el prototipo ha cumplido con las expectativas de los usuarios.

Figura 3.40. Resultados de la pregunta 10

La pregunta 11 es una pregunta abierta que permite conocer si el prototipo tuvo algún inconveniente al momento de ser usado por los encuestados con el objetivo de poder detectar errores en la funcionalidad, los resultados de esta pregunta se muestran en la Figura 3.41.

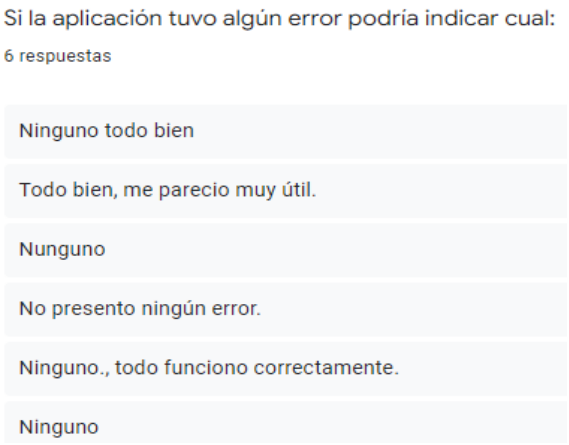


Figura 3.41. Resultados de la pregunta 11

3.3. CAMBIOS Y CORRECCIONES FINALES

Una vez procesadas las encuestas se optó por realizar pequeños cambios principalmente en la parte visual del prototipo con el objetivo de que sea más interactiva y que permita mejorar su funcionalidad; entre los principales cambios que se realizaron destacan:

- Agregar descripción en los botones de las listas, este cambio se lo realizó principalmente en los botones de las listas ya que poseen únicamente íconos, así fue necesario implementar una descripción cuando el cursor se encuentra sobre el botón lo que permite un mejor manejo del prototipo al usuario.
- Permitir ver el historial de pedidos a los usuarios empleados ya que únicamente pueden ver los pedidos que se generan en el día.
- Agregar efectos *hover*[43] a las listas.

En la Figura 3.42 se muestra un ejemplo de la descripción de los botones que poseen íconos únicamente, como se puede apreciar en la imagen cuando el cursor se sitúa sobre el botón se muestra un mensaje indicando la operación que permite realizar dicho botón.

Figura 3.42. Mostrar la descripción del botón

En la Figura 3.43 se muestra un ejemplo del efecto *hover* en una de las listas del prototipo, como se puede visualizar el efecto se aplica al momento en que el cursor se encuentra sobre una de las líneas, en este caso el cursor se encuentra sobre el producto con ID 2 y su aspecto cambia ya que cambia de color toda la fila de la tabla.

Figura 3.43. Ejemplo de efecto hover

[43] *Hover*: Alterar el aspecto de un elemento cuando se sitúa el puntero sobre el mismo

4. CONCLUSIONES Y RECOMENDACIONES

En el presente capitulo se exponen las conclusiones y recomendaciones del Trabajo de Titulación desarrollado.

4.1. CONCLUSIONES

- Al finalizar el presente Trabajo de Titulación se dispone de un prototipo de aplicación web que permite realizar la gestión de pedidos de un restaurante utilizando una arquitectura cliente-servidor, el patrón arquitectónico Modelo-Vista-Controlador y el el *framework* ASP .NET Core.
- La codificación del modelo se la realizó mediante el enfoque *Code First* de Entity Framework el cual fue realizado mediante el lenguaje de programación C#.
- Se utilizaron *Data Annotations* para realizar las validaciones necesarias en cada uno de los formularios del prototipo.
- Se utilizaron clases ViewModel para facilitar para la manipulación de la información que se renderizan en las vistas.
- Se utilizó la librería Identity propia de .NET para el manejo de los procesos de autenticación, manejo de credenciales de usuario, administración de roles de usuario y el manejo del correo de confirmación en el prototipo.
- El manejo de roles de usuario permitió limitar el acceso al prototipo a cada uno de los usuarios evitando que los mismos accedan a vistas para las que no están autorizados.
- La utilización de sesiones HTTP permitió manejar el ingreso de los pedidos al almacenar los productos seleccionados de manera temporal en un carrito de compras.
- Se utilizó el *framework* Bootstrap y tecnologías de *front-end* como HTML, CSS y JavaScript para renderizar la información en las respectivas interfaces de usuario, además, se utilizó Razor para combinar sentencias escritas en el lenguaje de programación C# con sentencias escritas en el lenguaje de marcado HTML, en el proceso de codificación de las vistas.

4.2. RECOMENDACIONES

- Se recomienda a futuro usar Xamarin para implementar la parte móvil del prototipo puesto que el mismo cuenta con una biblioteca de clases .NET Standard lo que permite compartir código entre plataformas.
- Se recomienda implementar un mecanismo que genere contraseñas automáticamente cuando se cree un nuevo usuario administrador o empleado ya que actualmente este proceso se lo realiza manualmente.
- Al ser ASP .NET Core multiplataforma se recomienda a futuro desarrollar y ejecutar aplicaciones web en otras plataformas basadas en Linux y MacOs.
- Se recomienda a futuro hospedar las aplicaciones desarrolladas en ASP .NET Core en servidores web como Apache o Nginx, ya que son libres de licenciamiento y no siempre se puede disponer de un Windows Server.
- Se recomienda en el futuro utilizar el *framework* Blazor que forma parte del proyecto de ASP .NET y permite desarrollar aplicaciones web dinámicas mediante el uso de lenguajes como C#, HTML, CSS y JavaScript.
- Se recomienda para futuros proyectos utilizar el protocolo HTTPS mediante el uso de certificados SSL/TLS otorgados por una entidad certificadora autorizada ya que dicho procedimiento no forma parte del alcance del presente Trabajo de Titulación.

5. REFERENCIAS BIBLIOGRÁFICAS

[1] K. Jacksi y S. Abass, «Development History of The World Wide Web,» INTERNATIONAL JOURNAL OF SCIENTIFIC & TECHNOLOGY RESEARCH VOLUME 8, pp. 75-78, 2019 Septiembre 2019.

[2] M. Contributors, «MDN Web Docs moz://a,» 29 Julio 2021. [En línea]. Available: https://developer.mozilla.org/es/docs/Web/HTTP/Overview. [Último acceso: 31 Julio 2021].

[3] C. Mateu, «Conceptos básicos del servidor web,» de Desarrollo de aplicaciones web, Barcelona, Eureca Media, 2004, p. 23.

[4] NETCRAFT, «netcraft.com,» 21 Diciembre 2016. [En línea]. Available: https://news.netcraft.com/archives/2016/12/21/december-2016-web-server-survey.html. [Último acceso: 01 Agosto 2021].

[5] S. Al-Fedaghui, «Developing Web Applications,» International Journal of Software Engineering and Its Applications, vol. 5, nº 2, pp. 57-58, 2011.

[6] SOFTWARE WEB & APPS, «SOFTWARE WEB & APPS,» 14 Agosto 2018. [En línea]. Available: https://www.desarrollodepaginasweb.com.mx/patrones-de-arquitectura-de-software/. [Último acceso: 09 Agosto 2021].

[7] CEC-EPN, Ciclo de vida de una solicitud HTTP con ASP.NET MVC, Quito, 2019.

[8] L. Gilibets, «IEBS,» 11 Noviembre 2020. [En línea]. Available: https://www.iebschool.com/blog/metodologia-kanban-agile-scrum/. [Último acceso: 17 Agosto 2021].

[9] M. Liang, «altexsoft,» 11 Marzo 2018. [En línea]. Available: https://www.altexsoft.com/blog/object-relational-mapping/.[Último acceso:14 Agosto 2018].

[10] Atlassian.com, «Atlassian BitBucket,» 7 Enero 2020. [En línea]. Available: https://www.atlassian.com/es/git/tutorials/what-is-git. [Último acceso: 16 Agosto 2021].

[11] Microsoft, «Microsoft Docs,» 24 Mayo 2019. [En línea]. Available: https://docs.microsoft.com/en-us/nuget/what-is-nuget. [Último acceso: 14 Agosto 2021].

[12] Microsoft, «Microsoft Docs,» 11 Septiembre 2019. [En línea]. Available: https://docs.microsoft.com/en-us/sql/ssms/sql-server-management-studio-ssms?view=sql-server-ver15. [Último acceso: 14 Agosto 2021].

[13] M. Docs, «Microsoft Docs,» 18 Junio 2021. [En línea]. Available: https://docs.microsoft.com/en-us/dotnet/standard/net-standard?tabs=net-standard-1-0. [Último acceso: 1 Octubre 2021].

[14] CEC-EPN, Entity Framework, Quito, 2019.

[15] desarrolloweb.com, «desarrolloweb.com,» 10 Noviembre 2020. [En línea]. Available: https://desarrolloweb.com/home/html. [Último acceso: 16 Agosto 2021].

[16] MDN Contributors, «MDN Web Docs moz://a,» 16 Agosto 2021. [En línea]. Available: https://developer.mozilla.org/es/docs/Learn/JavaScript/First_steps/What_is_JavaScript. [Último acceso: 17 Agosto 2021].

[17] javaTpoint, «javaTpoint,» 18 Agosto 2018. [En línea]. Available: https://www.javatpoint.com/what-is-bootstrap. [Último acceso: 17 Agosto 2021].

[18] Expertos en Negocios Online, «Expertos en Negocios Online,» 29 Noviembre 2019. [En línea]. Available:https://www.expertosnegociosonline.com/que-es-trello-para-que-sirve/.[Último acceso: 23 Agosto 2021].

[19] M. REHKOPF, «Atlassian Agile Coach,» [En línea]. Available: https://www.atlassian.com/es/agile/project-management/user-stories. [Último acceso: 18 Octubre 2021].

[20] a. .com, «asesorias .com,» 2020. [En línea]. Available: https://asesorias.com/empresas/modelos-plantillas/historias-usuario-scrum/. [Último acceso: 18 Octubre 2021].

[21] M. Docs, «Microsoft Docs,»27 Enero 2022.[En línea].Available: https://docs.microsoft.com/eses/aspnet/core/security/authentication/identity?view=aspnetcore-6.0&tabs=visual-studio. [Último acceso: 29 Enero 2022].

[22] Z. U. Hasan, «C#Corner,»2 Diciembre 2020.[En línea].Available:https://www.c-sharpcorner.com/article/managing-data-with-viewmodel-in-asp-net-mvc/.[Último acceso: 31 Enero 2022].

[23] M. Docs, «Microsoft Docs,» 13 Enero 2022. [En línea]. Available: https://docs.microsoft.com/es-es/dotnet/architecture/modern-web-apps-azure/architectural-principles#dependency-inversion. [Último acceso: 6 Febrero 2022].

6. ANEXOS

ANEXO A: Historias de usuario

ANEXO B: Sketches de las vistas

ANEXO C: Encuesta sobre el uso del prototipo

ANEXO D: Código Fuente del prototipo desarrollado

ANEXO E: Manual de usuario del prototipo

ANEXO F: Manual de instalación del prototipo

ANEXO A

En este anexo se muestran las historias de usuario generadas a partir de la entrevista realizada.

Id: UH-001-USR	
Nombre: Inicio de Sesión	
Cómo	Usuario
Puedo	Iniciar sesión utilizando un usuario y una contraseña.
Para	Ingresar al sistema.
Detalles	
La contraseña debe ser fuerte. Debe mostrar un mensaje de error si el usuario o la contraseña es incorrecta.	
Debe permitir modificar la contraseña.	

Anexo A.1. Historia de usuario inicio de sesión (usuarios)

Id: UH-002-USR	
Nombre: Configuración de la cuenta	
Cómo	Usuario
Puedo	Modificar los datos personales cuando lo requiera.
Para	Actualizar la información.
Detalles	
Debe existir una opción que permita modificar la información personal del usuario.	

Anexo A.2. Historia de usuario configuración de la cuenta (usuarios)

Id: UH-003-USR	
Nombre: Ver estado de las mesas	
Cómo	Usuario
Puedo	Ver la lista de las mesas
Para	Conocer el estado de las mesas.
Detalles	
La mesa puede estar libre u ocupada.	

Anexo A.3. Historia de usuario ver estado de las mesas (usuarios)

Id: UH001-ADM	
Nombre: Gestión de los productos del menú del restaurante.	
Cómo	Administrador
Puedo	Realizar el CRUD de los productos del menú del restaurante.
Para	realizar una gestión del menú.
Detalles	
Cada producto debe contener una imagen de este.	
Cada producto debe tener información como el nombre, el precio, descripción, imagen, tipo de producto, entre otros.	
Solo un administrador puede gestionar el menú.	

Anexo A.4. Historia de usuario para gestionar productos (usuario administrador)

Id: UH002-ADM	
Nombre: Gestión del menú del día	
Cómo	Administrador
Puedo	Realizar el CRUD del menú del día (almuerzos/meriendas).
Para	Realizar una gestión adecuada del menú del día
Detalles	
El almuerzo o merienda debe contener el nombre sopa, el nombre del segundo y el sabor del jugo.	
Solo el administrador puede gestionar el menú del día.	

Anexo A.5. Historia de usuario gestionar el menú del día (usuario administrador)

Id: UH003-ADM	
Nombre: Gestión de empleados y administradores.	
Cómo	Administrador
Puedo	Realizar el CRUD de los empleados y administradores del restaurante.
Para	Realizar una gestión adecuada de los empleados y administradores.
Detalles	
Las credenciales de acceso de los empleados y administradores se enviarán al correo del usuario cuando se los registra.	
Cada usuario debe tener información como el nombre, apellido, cédula, teléfono, dirección, entre otros.	
Solo un usuario con rol administrador puede crear otro administrador o empleado.	

Anexo A.6. Historia de usuario gestión de usuarios (usuario administrador)

Id: UH004-ADM	
Nombre: Gestionar los pedidos	
Cómo	Administrador
Puedo	Gestionar los pedidos de los clientes con el detalle de estos.
Para	Una gestión adecuada de los pedidos.
Detalles	
Se debe conocer el estado de los pedidos ingresados.	
Un administrador puede gestionar los pedidos mediante el uso de botones de gestión adecuados.	
Debe existir un historial de los pedidos.	

Anexo A.7. Historia de usuario visualizar los pedidos (usuario administrador)

Id: UH005-ADM	
Nombre: Gestión de cobros	
Cómo	Administrador
Puedo	Confirmar el pago cuando el cliente lo requiera
Para	Destinar las ventas adecuadamente.
Detalles	
El total de la cuenta se suma a las ganancias del día cuando se confirma el pago.	

Anexo A.8. Historia de usuario gestión de cobros (usuario administrador)

Id: UH006-ADM	
Nombre: Gestión de las ventas	
Cómo	Administrador
Puedo	Gestionar las ventas que se generan en el día o en alguna fecha específica.
Para	Administrar de mejor manera las ventas que genera el restaurante
Detalles	
El resumen indicará cuantos productos y el dinero que generó cada uno de los mismos con un valor total de ventas del día o de una fecha específica.	

Anexo A.9. Historias de usuario de gestión de las ventas (usuario administrador)

Id: UH007-ADM	
Nombre: Gestión de las mesas	
Cómo	Administrador
Puedo	Crear y eliminar una mesa.
Para	Gestionar las mesas que existen en el restaurante.
Detalles	
Cada mesa manejara un estado para indicar si la misma está disponible u ocupada.	
Los administradores pueden crear o eliminar una mesa mediante un botón.	

Anexo A.10. Historia de usuario gestión de las mesas (usuario administrador)

Id: UH001-EMP	
Nombre: Visualizar los pedidos.	
Cómo	Empleado
Puedo	Visualizar los pedidos del día
Para	Gestionar los pedidos a medida que ingresan.
Detalles	
Mostrar una lista de los pedidos y sus detalles a medida que ingresan al sistema.	
Visualizar el historial de los pedidos.	

Anexo A.11. Historia de usuario visualizar los pedidos (usuario empleado)

Id: UH002-EMP	
Nombre: Gestión de los pedidos.	
Cómo	Empleado
Puedo	Gestionar los pedidos de los clientes.
Para	Gestionar el pedido y llevarlo a la mesa correspondiente.
Detalles	
Indicar cuando el pedido se esté preparando o entregado.	
Puede gestionar dos tipos de pedidos: puede ser para servirse o para llevar.	
Cada pedido se asocia a un número de mesa.	

Anexo A.12. Historia de usuario gestión de pedidos (usuario empleado)

Id: UH001-CLI	
Nombre: Registrar Cliente	
Cómo	Cliente
Puedo	Registrarme en la aplicación.
Para	Poder ingresar al sistema.
Detalles	
Se debe ingresar un correo válido y existente.	
Cada usuario debe tener información como el nombre, apellido, cédula, teléfono, dirección, imagen, entre otros.	
Debe ingresar una contraseña fuerte.	

Anexo A.13. Historia de usuario registrar cliente (usuario cliente)

Id: UH002-CLI	
Nombre: Realizar el pedido	
Cómo	Cliente
Puedo	Escoger los productos disponibles del menú.
Para	Realizar el pedido.
Detalles	
Los clientes pueden visualizar todas las opciones del menú.	
Los clientes pueden ingresar el número de la mesa y elegir si desean el pedido para llevar o para servirse.	

Anexo A.14. Historia de usuario realizar el pedido (usuario cliente)

Id: UH003-CLI	
Nombre: Solicitar la cuenta	
Cómo	Cliente
Puedo	Solicitar la cuenta.
Para	Realizar el pago de la cuenta.
Detalles	
Puede ser a través de un botón.	

Anexo A.15. Historia de usuario solicitar la cuenta (usuario cliente)

Id: UH004-CLI	
Nombre: Historial de los pedidos de cada cliente	
Cómo	Cliente
Puedo	Visualizar todo el historial de pedidos realizados.
Para	Tener el detalle y la información de los pedidos
Detalles	
Se deben listar los pedidos realizados asociados únicamente a dicho cliente.	

Anexo A.16. Historia de usuario historial de pedidos (usuario cliente)

ANEXO B

En este anexo se visualizan los sketches de las vistas.

Anexo B.1. *Sketch* de inicio de sesión

Anexo B.2. *Sketch* del menú

Anexo B.3. *Sketch* para agregar producto

Anexo B.4. *Sketch* Editar Productos

Anexo B.5. *Sketch* de las opciones del menú del día

Anexo B.6. *Sketch* agregar nueva opción del menú del día

Anexo B.7. *Sketch* con el menú del día

Anexo B.8. *Sketch* para agregar un nuevo usuario

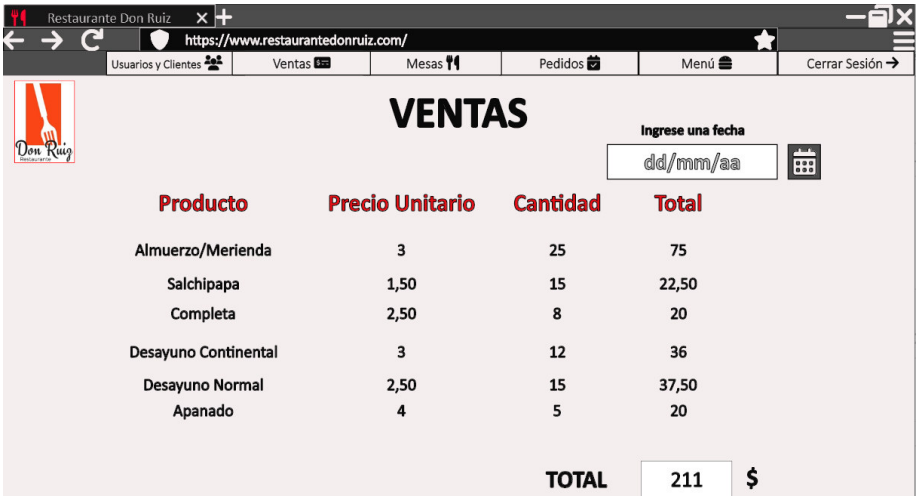
Anexo B.9. *Sketch* con la lista de usuarios

Anexo B.10. *Sketch* con la lista de usuarios

Anexo B.11. *Sketch* para seleccionar el pedido en el menú

Anexo B.12. *Sketch* resumen del pedido

Anexo B.13. *Sketch* con la lista de las mesas



Anexo B.14. *Sketch* de las ventas del día

ANEXO C

En este anexo se muestra la encuesta aplicada a los usuarios para validar el funcionamiento del prototipo.

1. ¿El sistema le permitió autenticarse correctamente?

☐ Sí
☐ No

2. ¿El sistema le permitió recuperar la contraseña mediante el correo registrado?

☐ Sí
☐ No

3. ¿El sistema le permitió realizar una gestión de usuarios de una manera adecuada y correcta?

☐ Sí
☐ No

4. ¿Pudo utilizar mecanismos para agregar, editar, visualizar y eliminar la información de los productos?

☐ Sí
☐ No

5. ¿Los criterios de búsqueda coincidieron con los resultados esperados?

☐ Sí
☐ No

6. ¿Pudo visualizar y gestionar el menú principal y el menú del día de acuerdo a su rol de usuario?

☐ Sí
☐ No

7. ¿Tuvo algún inconveniente gestionando el pedido?:

☐ Sí
☐ No

8. Indique el nivel de dificultad que tuvo con el manejo de la aplicación:

☐ Complicado
☐ Ni muy complicado ni muy fácil
☐ Fácil
☐ Muy Fácil

9. Indique que tanto es de su agrado la parte visual de la aplicación:

☐ Mucho
☐ Ni mucho ni poco
☐ Poco

10. Indique si el prototipo ha cumplido con sus expectativas:

☐ Sí
☐ No

11. Si la aplicación tuvo algún error podría indicar cual: _____

ANEXO D

Este anexo se adjunta de manera digital y contiene el código fuente generado.

ANEXO E

Este anexo se adjunta de manera digital y contiene el manual de usuario para el manejo del prototipo.

ANEXO F

Este anexo se adjunta de manera digital y contiene el manual de instalación del prototipo, así como el manual de usuario.