

My Reserve

Por: Adrián Iglesias Fernández

Índice

Índice	1
Introducción	2
Objetivos	2
Módulos docentes	3
Agradecimientos	4
Desarrollo	4
Análisis	4
Diseño	5
Implementación	15
Pruebas	20
Conclusiones	21
Líneas futuras	22
WebGrafía	23

Introducción

My Reserve es un proyecto diseñado para modernizar las peluquerías, reemplazando los métodos tradicionales de gestión de las peluquerías por un nuevo sistema totalmente digitalizado, pero al mismo tiempo sencillo e intuitivo, para que cualquiera sea capaz de utilizarlo. Los usuarios podrán reservar citas fácilmente y las peluquerías podrán optimizar la administración de sus servicios y horarios.

Objetivos

- **Objetivo General**

Modernizar el sistema de gestión de reservas de citas en peluquerías mediante una plataforma web, facilitando la organización interna, mejorando la experiencia del usuario y promoviendo el crecimiento de la peluquería.

- **Objetivos Específicos**

- Diseñar una estructura de datos relacional:
 - Crear entidades en la base de datos para representar a los usuarios, peluqueros, peluquerías, grupos de peluquerías, servicios, horarios, citas, categorías.
 - Definir las relaciones entre estas entidades, garantizando la integridad y las claves foráneas para vincular las tablas.
- Diseñar un sistema de gestión de roles:
 - Desarrollar funcionalidades de registro e inicio de sesión para las entidades de la base de datos con validaciones de credenciales.
 - Permitir que cada entidad gestione sus propios datos.
- Gestionar la administración de las peluquerías y grupos de peluquerías:
 - Implementar métodos para que las peluquerías puedan crear y editar sus peluqueros.
 - Implementar métodos para que los grupos de peluquería puedan crear y editar peluquerías.
- Permitir la personalización del servicio de la peluquería:
 - Independientemente de la plantilla ya hecha por la web, se da la libertad a la peluquería de crear sus propios servicios y asociarlos a ellos.

- Diseñar un sistema de reservas fácil de usar y accesible:
 - Se crea una interfaz intuitiva y accesible para todos los dispositivos electrónicos que permita a los usuarios reservar su cita sin problemas.
 - Permitir a los usuarios reservar cita seleccionando la peluquería, el peluquero, los servicios, la fecha y la hora.
- Desarrollar una interfaz de usuario intuitiva y responsive:
 - Crear un diseño web que facilite la navegación en todos los dispositivos.
 - Implementar un diseño claro y agradable, utilizando paletas de colores sencillas.
- Validación en tiempo real:
 - Implementar métodos de validación a los formularios utilizando JavaScript para mejorar la experiencia del usuario, mostrando mensajes de error.
- Optimizar la gestión interna de citas:
 - Implementar un sistema para evitar las dobles reservas y minimizar errores.
- Realizar pruebas de la aplicación
 - Comprobar la integridad de las consultas y relaciones de tablas en la base de datos, asegurando que las operaciones **CRUD** (Create, Read, Update, Delete) funcionen correctamente.
 - Validar la interfaz del usuario en dispositivos móviles.

Módulos docentes

Para la realización del proyecto he empleado varias tecnologías en las que he trabajado a lo largo de esta formación.

- Referente a la asignatura de Desarrollo de Aplicaciones Web Entorno Servidor he utilizado el lenguaje C# junto al framework .NET para realizar las conexiones pertinentes entre la aplicación web y la base de datos, esto lo conseguimos siguiendo el modelo MVC (Modelo-Vista-Controlador).

- Referente a la asignatura de Diseño de Interfaces Web he utilizado el lenguaje HTML5, CSS3 y la librería Bootstrap 5 para crear las interfaces necesarias para una mejor experiencia del usuario.
- Referente a la asignatura de Desarrollo de Aplicaciones Web Entorno Cliente he utilizado el lenguaje de JavaScript, pero para aumentar su rendimiento decidí utilizar la librería JQuery para realizar la parte lógica de la aplicación en el ámbito FrontEnd.
- Referente a la asignatura de Entornos de Desarrollo junto a la asignatura de Despliegue de Aplicaciones Web he utilizado GitHub para subir el proyecto a un repositorio y que se guarden los cambios realizados para poder volver atrás en caso de que me equivoque o poder trabajar desde otro dispositivo y tener la aplicación actualiza en todo momento.
- Referente a la asignatura de Bases de Datos he utilizado como IDE el Management Studio 19 de SQL Server Management Studio, gracias a lo aprendido en esta asignatura he podido crear la base de datos de la aplicación siendo robusta y con las restricciones necesarias para evitar errores.

Agradecimientos

Agradezco al Colegio Montecastelo y a todos los profesores que me han apoyado en este camino, a Ramón Ríos, Tacio Camba, Martín García, Daniel Sánchez y Andrés Luna por la información que me brindaron durante el curso y las facilidades que me dieron para hacer este proyecto.

Desarrollo

Análisis

- **Entidades de datos**
Se necesitará crear entidades de datos relacionadas entre ellas para que los usuarios puedan ver y concertar sus citas, las peluquerías crear sus peluqueros y los grupos sus respectivas peluquerías.
- **Tratado de datos**
Como se registran los datos es como deben de ser tratados, cada entidad de la base de datos puede crear, modificar, editar o eliminar su información según el rol que tenga en la aplicación.

- **Gestión de usuarios**

Un usuario necesitará una cuenta para poder reservar las citas, seleccionando la peluquería, el peluquero, la hora y el día. Él usuario puede editar y borrar sus datos.

- **Gestión de peluqueros**

Un peluquero necesitará su cuenta para que pueda ser seleccionado como el que le va a cortar el pelo al cliente y ver las citas que pueda tener, su información es editable.

- **Gestión de peluquerías**

Una peluquería puede crear sus propios peluqueros como a su vez eliminarlos, también puede crear horarios y servicios personalizados para que el usuario pueda utilizarlos.

- **Gestión de los grupo de peluquerías**

Un Grupo de peluquerías se encargará de gestionar todas las peluquerías que tenga en su franquicia, puede editar y eliminar su información más las de sus peluquerías.

- **Registro e inicio de Sesión**

Como contamos con usuarios, peluqueros, peluquerías y grupos debemos tener una vista de inicio de sesión y otra de registro. Por lo que cada una de las entidades debe de tener unas credenciales para poder iniciar sesión cuando más lo deseé

- **Diseño intuitivo**

La aplicación cuenta con un diseño que le facilita al usuario en todo momento a saber dónde está y que puede hacer, aparte, cuenta con un diseño responsive adaptado a teléfonos móviles y otros dispositivos. Tiene una paleta de colores sencilla agradable a la vista del usuario donde indica la información más importante.

- **Creación de citas**

Entrando como usuario a la aplicación, el usuario puede buscar la peluquería que más deseé y en ella podrá ver los peluqueros disponibles por la peluquería, los horarios disponibles y los servicios que le puede dar la peluquería, solo tiene que seleccionar lo que él desea para hacer la cita.

Diseño

La aplicación está hecha en un proyecto de .NET, en el cual separamos dos grandes grupos El **BackEnd** y el **FrontEnd**. Para el proyecto decidí utilizar el sistema de **Screaming Architecture**, es una arquitectura que promueve la claridad y flexibilidad en el desarrollo de software.

- **BackEnd**

Ahora toca enfocarnos en la Base de Datos de **MyReserve**. Su estructura está conformada por las siguientes tablas

Tabla Usuarios

ID	Nombre	Correo Electrónico	Contraseña
ID del usuario que se autoincrementa una vez que se crea un usuario	Nombre del usuario.	Correo Electrónico del usuario.	Contraseña del usuario.

La tabla de usuarios se encarga de la gestión de los propios usuarios del sistema.

- Usa el **ID** como clave primaria.
- El **nombre** es por como se le tratará al usuario y donde se verá reflejada su información en la página.
- El **correo electrónico** tiene una restricción donde comprueba que el usuario solo pueda tener un correo asociado y es necesario para iniciar sesión.
- La **contraseña** es necesaria a la vez que el correo para iniciar sesión.

Tabla Grupo Peluquerías

ID	Nombre	Correo Electrónico	Contraseña
ID del Grupo de Peluquerías que se autoincrementa una vez que se crea un Grupo.	Nombre del Grupo de Peluquerías.	Correo Electrónico del Grupo de Peluquerías necesario para iniciar sesión.	Contraseña del Grupo de Peluquerías necesaria para iniciar sesión.

La tabla de Grupos de Peluquerías se encarga de la creación como de la eliminación de las peluquerías encargadas a su nombre.

- Usa el **ID** como clave primaria.
- El **nombre** se utiliza en gran parte de los casos para guardar peluquerías o peluqueros y asociarlos al grupo para saber a qué grupo pertenecen.
- El **correo electrónico** tiene una restricción donde comprueba que el grupo solo pueda tener un correo asociado y es necesario para iniciar sesión.
- La **contraseña** es necesaria a la vez que el correo para iniciar sesión.

Tabla Peluquería

ID	Nombre	Correo Electrónico	Contraseña	País	Región	Ciudad	Dirección	Teléfono	Grupo ID
ID de las Peluquerías que se autoincrementa una vez que se crea una.	Nombre de la Peluquería	Correo Electrónico de la peluquería	Contraseña de la peluquería	País de la peluquería	Región de la peluquería	Ciudad de la Peluquería	Dirección de la peluquería	Teléfono de la peluquería	ID del grupo de la peluquería

La tabla de Peluquería se encarga de la creación de peluqueros y eliminación de peluqueros, aparte de ofrecer sus servicios

- Usa el **ID** como clave primaria.
- El **nombre** se usa para identificar la peluquería.
- El **correo electrónico** tiene una restricción donde comprueba que la peluquería solo pueda tener un correo asociado y es necesario para iniciar sesión.
- La **contraseña** es necesaria a la vez que el correo para iniciar sesión.
- El **país** se usa para filtrar por peluquerías.
- La **región** también se usa para el filtro de peluquerías.
- La **ciudad** también se usa para el filtro de peluquerías.
- La **dirección** para informar al usuario de donde puede encontrar la peluquería.
- El **teléfono** para informar al usuario de cuál es el número al que puede llamar en caso de duda o de lo que necesite.
- El **ID Grupo** se encarga de vincular la peluquería con su grupo de peluquerías.

Tabla Peluquero

ID	Nombre	Correo Electrónico	Contraseña	Descripción	Experiencia	Instagram	Peluquería ID	Grupo ID
ID del peluquero que se autoincrementa a una vez que se añade.	Nombre del peluquero	Correo Electrónico del peluquero	Contraseña del peluquero	Descripción del peluquero para dar información al cliente	Años de Experiencia del peluquero	Red social del peluquero	ID de la peluquería	ID del grupo de la peluquería

La tabla de Peluquería se encarga de la creación de peluqueros y eliminación de peluqueros, aparte de ofrecer sus servicios

- Usa el **ID** como clave primaria.
- El **nombre** se usa para informar al usuario del peluquero que le va a entender.
- El **correo electrónico** tiene una restricción donde comprueba que el peluquero solo pueda tener un correo asociado y es necesario para iniciar sesión.
- La **contraseña** es necesaria a la vez que el correo para iniciar sesión.
- La **descripción** se utiliza para dar más información al usuario para ver las especialidades del peluquero.
- La **experiencia** se utiliza para informar al usuario cuantos años tiene en el ámbito laboral.
- El **instagram** es la red social donde el peluquero en caso de que tenga puede exponer sus cortes de pelo y que el usuario puede verlos.
- El **ID Peluquería** se encarga de vincular el peluquero con su peluquería.
- El **ID Grupo** se encarga de vincular el peluquero con el grupo de peluquería al que pertenece.

Tabla Categoría

ID	Nombre
ID de la categoría.	Nombre de la categoría.

La tabla de Categoría se encarga de proporcionar información tanto a la peluquería como al usuario para ver qué secciones tiene la misma peluquería

- Usa el **ID** como clave primaria.
- El **nombre** de la categoría para indicar tantos a las peluquerías como usuarios de los servicios divididos por categorías.

Tabla Servicios

ID	Nombre	Precio	Categoría ID	Peluquería ID
ID del servicio.	Nombre del servicio.	Precio del servicio.	ID de la Categoría.	ID de la peluquería

La tabla de Servicios se encarga de proporcionar a la peluquería los servicios que puede tener.

- Usa el **ID** como clave primaria.
- El **nombre** para que la peluquería o el usuario la pueda seleccionar.
- El **precio** para cuando se haga la cita se compruebe el pago final.
- Usa el **ID** de la Categoría como clave foránea para poder agruparlas.
- Usa el **ID** de la Peluquería para saber si el servicio que hemos creado está asociado únicamente a esa peluquería.

Tabla PeluqueríaServicios

ID de la Peluquería	ID del servicio
ID del servicio.	Nombre del servicio.

La tabla de PeluqueríaServicios se encarga de proporcionar a la peluquería los distintos servicios que puede tener.

- Usa los 2 **IDs** como clave primaria.
- Usa los 2 **IDs** como clave foránea para que cada peluquería tenga tantos servicios como ellos quieran.

Tabla Horarios

ID	Fecha
ID del horario.	Hora de la cita.

La tabla de Horarios se encarga de proporcionar información a la peluquería para que pueda seleccionar los horarios para la misma.

- Usa el **ID** como clave primaria
- La **fecha** para que el usuario y la peluquería puedan seleccionar la que deseen.

Tabla PeluqueriaHorarios

ID de la Peluquería	ID del Horario	Hora Reservado
ID de la peluquería.	ID del horario.	Booleano que nos sirve para saber si la hora ya está reservada.

La tabla de PeluqueriaHorarios se encarga de proporcionar a la peluquería los distintos horarios que puede tener.

- Usa los 2 **IDs** como clave primaria.
- Usa los 2 **IDs** como clave foránea para que cada peluquería tenga tantos servicios como ellos quieran.
- La **hora reservada** se utiliza para que cuando un usuario haya guardado una cita con esa hora no vuelva a aparecer

Tabla Citas

ID	ID del usuario	ID del peluquero	ID de la peluquería	ID del horario	Fecha de la Cita
ID de la cita	ID del usuario	ID del peluquero	ID de la peluquería	ID del horario	Es la fecha que se guarda para concretar la cita

La tabla de **citas** se encarga de guardar las citas de los usuarios según los parámetros que le pasamos.

- Usa el **ID** como clave primaria.
- Usa el **ID** del Usuario como clave foránea, esto nos sirve para realizar queries donde podamos ver las citas del usuario.
- Usa el **ID** del Peluquero como clave foránea, esto nos sirve para realizar queries donde podamos ver las citas del peluquero.
- Usa el **ID** de la Peluquería como clave foránea, esto nos sirve para realizar queries donde podamos ver las citas de la peluquería.
- Usa el **ID** del Horario como clave foránea.
- La **fecha de la cita** se utiliza para informar al usuario que día cuadra la cita establecida.

Tabla CitasServicios

ID de la Cita	ID del servicio
ID de la Cita	ID del servicio

La tabla de **CitasServicios** se encarga de guardar los servicios seleccionados por los usuarios, gracias a una query la podemos juntar los servicios.

- Usa los 2 **IDs** como clave primaria.
- Usa los 2 **IDs** como clave foránea para que cada peluquería tenga tantos servicios como ellos quieran.

Tabla País

ID	Nombre
ID del país.	Nombre del país.

La tabla de País se encarga de proporcionar a la aplicación una lista de países para dar más detalles a la peluquería

- Usa el **ID** como clave primaria.
- El **nombre** del país se utiliza para informar que país se va a utilizar.

Tabla Región

ID	Nombre	ID País
ID de la región	Nombre de la Región	ID del País

La tabla de Región se encarga de proporcionar a la aplicación una lista de regiones conectada a la de país para mostrar las regiones de cada país.

- Usa el **ID** como clave primaria.
- El **nombre** de la región se utiliza para informar de la región que se va a utilizar.
- El **ID País** se encarga de vincular la región con el país correspondiente.

Para manejar las tablas de la base de datos programé una serie de archivos en las carpetas **Controllers** y **Models** y utilicé los siguientes paquetes **NuGet**: **System.Data.SqlClient**, el **MICRO-ORM** de **Dapper** y **Newtonsoft.Json**.

- **System.Data.SqlClient**: Se encarga de crear conexión con la base de datos que estemos utilizando.
- **Dapper**: Se encarga de mapear nuestra base de datos y darnos una respuesta. Está orientada a objetos.
- **Newtonsoft.Json**: Nos proporciona funcionalidad para serializar y deserializar desde JSON las entidades que inician sesión en nuestra App

En cuanto a las carpetas, están distribuidas de la siguiente manera:

- Carpeta **Controllers**: Se encargan procesar las solicitudes de **HTTP** de tipo **GET** como **POST**, se encargan de la recuperación de datos de la entidad y de la respuesta que le daremos al cliente, en este caso, la vista.
- Carpeta **Models**: Esta carpeta se divide en tres carpetas,
- la “**TablasBBDD**” “**HelpersTablasBBDD**”, “**Repository**” y dos archivos. Archivo **Conexion**: Contamos con un archivo llamado en este caso **conexion.cs** que nos permite conectarnos a la base de datos gracias al paquete NuGet **System.Data.SqlClient**.
 - Archivo **ErrorViewModel**: Tiene 2 propiedades que sirven para informar al usuario sobre el error de la aplicación.
 - Carpeta **TablasBBDD**: Tiene todas las propiedades que tienen la Base de Datos más añadidos para la funcionalidad de la App.
 - Carpeta **Repository**: En este proyecto, cada controlador tiene su propia carpeta Repository dentro de la misma. Esto beneficia a la organización y limpieza del código. En esta carpeta hay dos archivos, una **clase** y una **interfaz**, en la interfaz se declaran los métodos y en la clase con la

ayuda de **Dapper** y la conexión a la base de datos podemos realizar las *querys* necesarias para obtener la información.

- Carpeta **HelpersTablasBBDD**: Tienen propiedades de varias clases y están hechas para mejorar la vista final jugando varias propiedades de las tablas de la base de datos.
- Fuera de las carpetas tenemos el archivo **appsettings.json** que se encarga de realizar la conexión con la base de datos según la cadena que le pasemos y que luego en el **program.cs** la especifiquemos.

- **FrontEnd**

Ahora vamos a enfocarnos en cómo está estructurada la información de la web y cómo la tratamos. Se dividen en subcarpetas dentro de la carpeta **Views**, aunque también contamos con la carpeta **wwwroot** y las propiedades del **program.cs**:

- **Home:**
 - La página principal se trata de una presentación de la página web donde a parte de poner información sobre la App, contiene un *login* y registro de usuario, un *login* de Peluqueros, un *login* de grupo de peluqueros y uno de peluquerías.
- **Formularios:**
 - Los *logins* y registros se encuentran en una carpeta asociada a su mismo controlador para pasar los métodos y acciones necesarias.
 - Los registros de horarios y servicios también se encuentran disponibles en esta carpeta.
 - También podrás editar los servicios y horarios disponibles de la peluquería una vez esta misma los haya guardado en la base de datos.
- **Grupo Peluquerías:**
 - El grupo de peluquerías tiene un panel de administrador donde podrá ver las peluquerías que tiene asociadas al grupo.
 - Un grupo de peluquerías podrá editar y eliminar la peluquería asociada y modificar su información.
 - Al crear una peluquería, en el apartado en el que indicas a qué grupo pertenece la peluquería, se te rellena automáticamente con el grupo con el que inicias sesión.
 - El grupo de peluquerías podrá editar y eliminar el grupo.

- **Peluquerías:**

- Una peluquería tiene un panel de administrador donde ve los peluqueros asociados a la misma.
- Podrá modificar la información de un peluquero y eliminarlo si la peluquería lo desea.
- Contiene varios enlaces que llevan al registro de horarios y servicios para asociarlos a la peluquería.
- La peluquería podrá ver las citas pendientes que hay.
- La peluquería podrá editar y eliminar un peluquero.
- Un peluquero podrá acceder al portal, editar su información y ver las citas pendientes.

- **Info Usuarios:**

- Un usuario cuando acceda a su portal le aparecerán todas las peluquerías que seleccione según el país y la región deseada.
- Podrá editar su información y eliminar su cuenta
- El usuario podrá crear la cita seleccionando los servicios, horarios y peluqueros correspondientes a una peluquería, también puede eliminarlas.
- El usuario podrá ver sus citas pendientes

- **Error:**

- Se encarga de mostrar distintas vistas según el error correspondiente que da la solicitud.

- **Shared:**

- Contiene archivos generales que se comporten entre las vistas

- **Carpeta `wwwroot`:**

- Dentro de esta carpeta se encuentran los archivos CSS3 y JavaScript de la página.

- **Archivo `program.cs`:**

- Se encarga de mostrar la página principal de la aplicación.

Implementación

- **BackEnd**

En la parte de desarrollo de la aplicación hay varios archivos hechos para poder hacer la conexión a la base de datos.

- **Conexion.cs:** Es el archivo que se encarga de realizar la conexión a la base de datos.

```
15 referencias
public SqlConnection getConexion() {
    var conexion = new SqlConnection(_conexion);
    conexion.Open();
    return conexion;
}
```

Con este método podemos hacer la conexión a la base de datos, gracias al paquete NuGet **System.Data.SqlClient**.

- **appsettings.json:** Es el archivo donde le especificamos mediante un .json la base de datos que queremos utilizar.

```
{:} appsettigns.json +
{
  "ConnectionStrings": {
    "MyReserve": "Data source=DESKTOP-9DV3842\\MSSQLSERVER02;Initial Catalog=MyReserve;User ID=sa; " +
    "Password=root;Trusted_Connection=True;MultipleActiveResultSets=true;TrustServerCertificate=True;"
  }
}
```

- **Program.cs:** Es el archivo donde se realizan los servicios de la aplicación.

```
builder.Services.AddSingleton(new Conexion(builder.Configuration.GetConnectionString("MyReserve")));
builder.Services.AddScoped<IFormulario, FormularioRepository>();
builder.Services.AddScoped<IGrupoPeluquerias, GrupoPeluqueriasRepository>();
builder.Services.AddScoped<IPeluqueria, PeluqueriaRepository>();
builder.Services.AddScoped<IInfoUsuarios, IInfoUsuariosRepository>();
builder.Services.AddSession();

app.UseSession();
```

En estas líneas de código hacemos la conexión a la base de datos especificando el nombre de la cadena de conexión puesta en el archivo **appsettings.json**, utilizamos el modelo **singleton** debido a que es un patrón que solo deja crear una instancia. En las demás líneas indicamos los

repositorios que va a tener nuestra aplicación para realizar las queries necesarias y con el `AddSession()` guardamos los datos en el servidor.

- Carpeta **Repository**:

Aquí se encuentran todos los archivos destinados a hacer el mapeo a la base de datos utilizando **Dapper**. Un ejemplo sería el siguiente:

- Esto sería un login

```
public async Task<Usuarios> Login(string usu_correo_electronico, string usu_contrasenha) {
    var query = "SELECT * FROM Usuarios WHERE usu_correo_electronico = @usu_correo_electronico " +
        "AND usu_contrasenha = @usu_contrasenha";
    using(var connection = _conexion.getConexion()) {
        var usuario = await connection.QueryFirstOrDefaultAsync<Usuarios>(query,
            new { usu_correo_electronico, usu_contrasenha });
        return usuario;
    }
}
```

Declaramos una variable llamada query donde escribimos la query que vamos a ejecutar. Declaramos una variable para llamar a la conexión a la base de datos, al ser método asíncrono esperamos a la respuesta y le mandamos la información necesaria para que haga la query, al acabar nos devuelve el usuario.

- Esto sería un registro

```
public async Task RegistroUsuario(Usuarios usuarios) {
    var query = "INSERT INTO Usuarios (usu_nombre, usu_correo_electronico, usu_contrasenha) " +
        "VALUES (@usu_nombre, @usu_correo_electronico, @usu_contrasenha)";
    var parametros = new DynamicParameters();
    parametros.Add("usu_nombre", usuarios.usu_nombre, DbType.String);
    parametros.Add("usu_correo_electronico", usuarios.usu_correo_electronico, DbType.String);
    parametros.Add("usu_contrasenha", usuarios.usu_contrasenha, DbType.String);

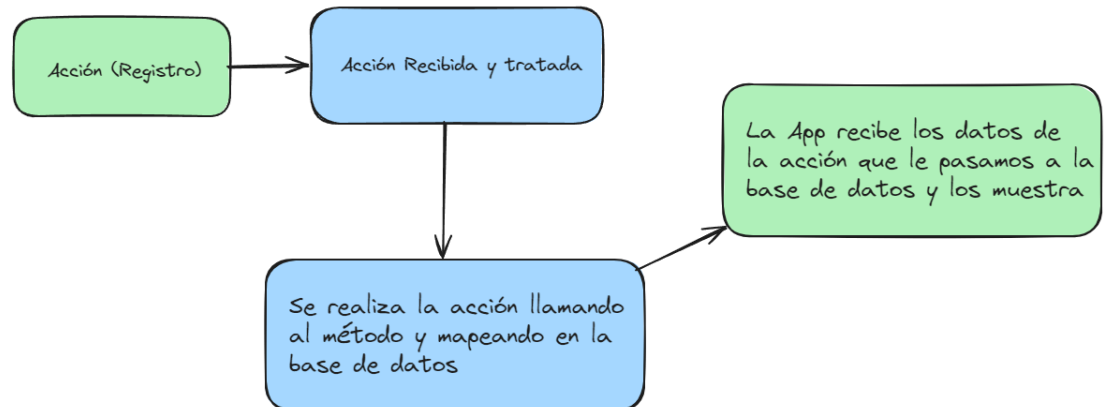
    using(var connection = _conexion.getConexion()) {
        await connection.ExecuteAsync(query, parametros);
    }
}
```

Hacemos lo mismo que hicimos en el ejemplo anterior pero con la diferencia de que esta vez estamos haciendo un **INSERT** en vez de un **SELECT**. La diferencia es que ahora tendremos que utilizar los **DynamicParameters()**. Estos parámetros se añaden de la siguiente manera:

- En primer lugar se pone el nombre del parámetro de la base de datos.
- En segundo lugar el nombre del usuario que le estamos pasando.
- En tercer lugar el tipo de datos que vamos a registrar.

Al pasar todos los datos, hacemos la conexión y gracias a **Dapper** añadimos el usuario.

El funcionamiento de la aplicación a la hora de hacer las peticiones es la siguiente:



- Carpeta **Models**:

Se encuentran todas las clases de las entidades de nuestra base de datos incluyendo sus parámetros, con esto conseguimos tratar los datos y acceder a ellos para poder mostrarlos y usarlos en la vista, también podemos pasarlos al controlador.

```

public class Region {
    0 referencias
    public int reg_id { get; set; }
    0 referencias
    public int reg_pai_id_fk { get; set; }
    0 referencias
    public string reg_nombre { get; set; }
}
  
```

- Carpeta **Controllers**:

Aquí se encuentran todas las peticiones que vamos a realizar para mostrar la información en la página. Según el código de estado que nosotros enviemos nos dará una vista u otra.

Un ejemplo para mostrar la página sería el siguiente

```

public IActionResult LoginPeluqueria() {
    return View();
}
  
```

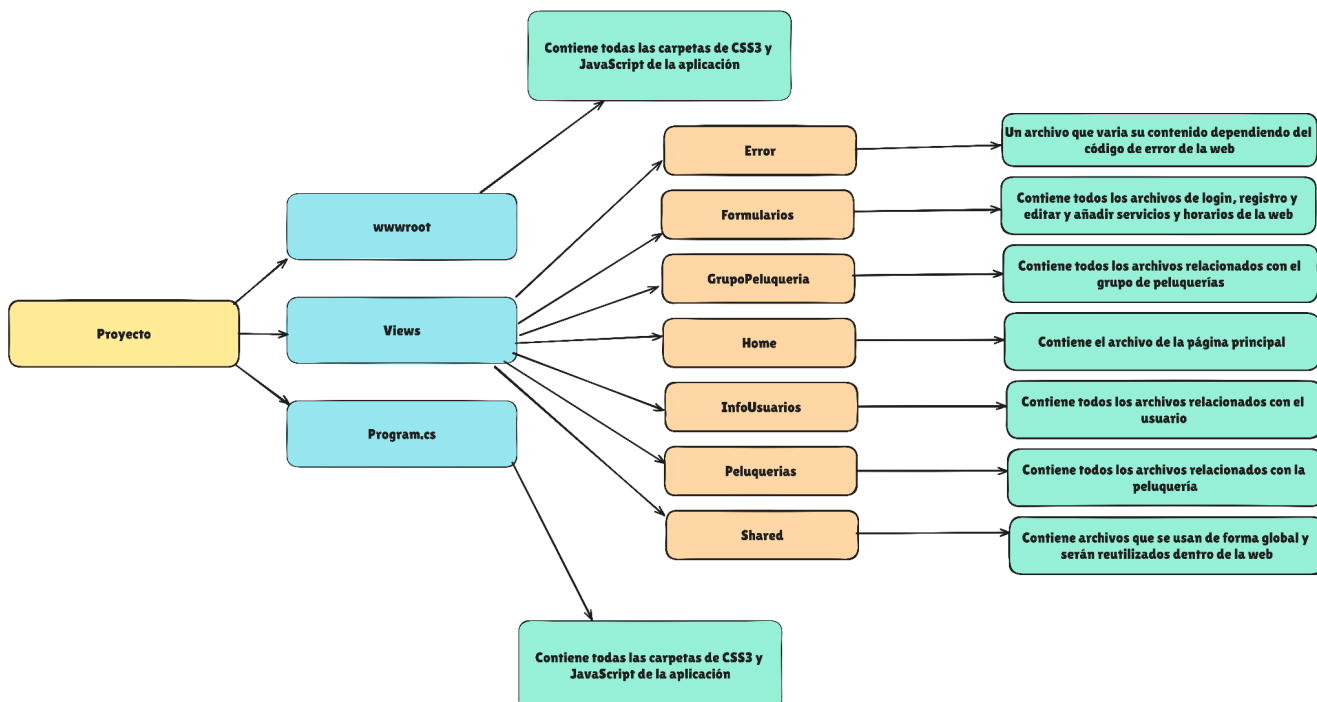
Un ejemplo de una petición POST sería la siguiente:

```
0 referencias
public async Task<IActionResult> LoginPeluquerias(string pelu_correo_electronico, string pelu_contrasenha) {
    var peluqueria = await _formularioRepository.LoginPeluqueria(pelu_correo_electronico, pelu_contrasenha);
    if(peluqueria == null) {
        return View("LoginPeluqueria");
    } else {
        serializarPeluqueria(peluqueria);
        return RedirectToAction("Portal", "Peluquerias", peluqueria);
    }
}
```

- Cada carpeta **controller** contiene métodos de serialización y deserialización según el tipo de controller que sea.
 - **Serialización:** Se utiliza para convertir un objeto en un formato JSON para mandarla a través de una URL y así almacenar los datos en la sesión. Contiene un método de seguridad para que no puedas serializarse si estás intentando entrar por URL
 - **Deserialización:** Es lo contrario a la serialización, se encarga de convertir de JSON a objeto y se utiliza para recuperar los datos mandados por una URL.

• FrontEnd

Ahora vamos a ver la implementación que llevamos desde el backend a mostrarla en el frontend, en una primera vista el diseño sería al siguiente.



Este proyecto está hecho con ASP .NET, es un framework de **Microsoft** que se utiliza para crear tanto aplicaciones de escritorio como web, utiliza C# por su lenguaje tipado que da una gran seguridad al código.

- Cada proyecto viene cubriendo grandes necesidades, tiene librerías instaladas dentro como JQuery, Bootstrap 5, etc...
- En la carpeta **Shared** se encuentran los archivos que usamos de forma global y reutilizamos.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <title>@ViewData["Title"] - MyReserve</title>
  <link href="/css/site.css" rel="stylesheet" />
  <link rel="stylesheet" href="/lib/bootstrap/dist/css/bootstrap.min.css" />
  @RenderSection("Styles", required: false) <!--Para que no rendire un CSS que no es de la página-->
</head>
<body>
  <main role="main" class="pb-3">
    @RenderBody()
  </main>
  <script src="/lib/jquery/dist/jquery.min.js"></script>
  <script src="/lib/bootstrap/dist/js/bootstrap.bundle.min.js"></script>
  <script src="/js/site.js" asp-append-version="true"></script>
  @await RenderSectionAsync("Scripts", required: false)
</body>
</html>
```

Lo que va dentro de la etiqueta main es lo que renderiza el cuerpo de la página y contiene el RenderSection para que en los archivos que estemos creando podamos poner el CSS complementario a esa página.

```
@section Styles {
  <!--El style que renderizamos-->
  <link rel="stylesheet" href="/css/home/home.css" asp-append-version="true" />
}
```

- Para esterilizar la página y darle un toque más profesional, utilicé la fuente de Coffe Spark, sacada de [DaFont](#), una página muy conocida para el uso de fuentes. Le da más estilo a la página y la hace más bonita a primera vista el usuario.
- Utilicé el archivo **normalize.css** ([Normalize.css](#)) aunque su contenido está copiado en el **site.css** para no cambiarle el nombre, este archivo se encarga de corregir los errores al mostrar la página en los distintos navegadores.
- Utilicé la librería de **Font Awesome** ([Font Awesome](#)) que está conectada a la página mediante un **CDN** para poder mostrar los iconos en la página y hacer para el usuario un diseño más intuitivo.
- Para las fechas utilicé la librería de **Flatpickr** ([Flatpickr](#)).

- Para un mejor diseño he utilizado código JavaScript de bootstrap para mejorar la interfaz y la experiencia de la identidad, cosa que con CSS3 puro no lograría cumplir.

Pruebas

- **BackEnd**

- En todas las funciones de los controladores de nuestro proyecto, he comprobado el error por si la entidad mete un tipo de dato incorrecto que le salga un aviso de que el dato es incorrecto con el tipo de la base de datos devolviendo la vista, también si el usuario hace directamente una petición post que le salga un pantalla de error o si el usuario intenta acceder a un recurso del cuál no tiene permisos, se le redirige a la página principal.
- En cuanto a eliminar una entidad que esté relacionada con otras (Si elimino una Peluquería, también tengo que eliminar las citas) he controlado mediante queries que sea posible la eliminación completa de las entidades que dependen con la que se está eliminando.
- Gracias a las queries que he programado en la carpeta **Repository** de cada entidad de la base de datos, estos me hacían más rápido el trabajo, teniendo que llamar simplemente a la función pasándole los datos necesarios.
- En casos concretos del proyecto, he tenido que pasar los datos que me devolvía la base de datos a un formato JSON para poder tratar los datos desde una petición AJAX y que puedan ser tratados correctamente y poderlos devolver a la vista.

- **FrontEnd**

- En todos los formularios del proyecto he añadido JavaScript utilizando la librería de JQuery para darle un mejor rendimiento al código y hacer las validaciones para que todos los datos que se vayan a mandar sean comprobados previamente gracias a esta librería, añadiendo mensajes de error informativos para el usuario.
- Todas las vistas de la aplicación tienen un diseño responsive hecho para que pueda ser accesible desde cualquier navegador y dispositivo electrónico.
- Para la validación de fechas se ha utilizado la librería **Flatpickr** para validar que los domingos no puedan ser seleccionados y que no puedas seleccionar una fecha anterior al día actual.

- A la hora de reservar una cita, hay una validación cuando el usuario escoge el día para realizar su cita, si ya tiene una cita el día seleccionado se lanza un mensaje avisando que el usuario ya tiene una cita y no dejándole añadir una cita más.

Conclusiones

- **Autenticación de entidades:**

Se han creado páginas necesarias para el inicio de sesión y registro de la entidad, con las validaciones necesarias para llevar a cabo el acto en la base de datos en caso de una nueva entidad; o redirigir a una nueva pestaña. También se encuentra añadida una comprobación donde una entidad no puede entrar en una vista mediante URL si no se ha iniciado sesión anteriormente.

- **Gestión de usuarios:**

Se puede hallar una tabla dentro de la base de datos donde el usuario puede entrar o crear su cuenta para acceder a las demás vistas de la aplicación. Estos usuarios pueden editar y eliminar su información, además de crear y eliminar sus citas.

- **Gestión de peluqueros:**

Se encuentra una tabla dentro de la base de datos donde el peluquero puede entrar a su cuenta para acceder a su propia información personal y editarla, además de ver sus próximas citas.

- **Gestión de peluquerías:**

Se ubica una tabla dentro de la base de datos donde la peluquería puede entrar a su cuenta para realizar múltiples acciones, como crear, editar o eliminar un peluquero; crear, editar y eliminar sus propios servicios; editar o eliminar sus propios horarios; o ver las citas programadas de la peluquería.

- **Gestión de grupos de peluquerías:**

Hay una tabla dentro de la base de datos donde el grupo de peluquerías puede entrar para crear tantas peluquerías como desee controlar, además de editar su información. Adicionalmente, el mismo grupo puede editar su información, y a su vez eliminar peluquerías o a el propio grupo, eliminando así todas las peluquerías que estén relacionadas con él.

- **Gestión de servicios:**

Existe una tabla dentro de la base de datos con una plantilla predeterminada para que las peluquerías puedan añadir precios competentes en sus distintos servicios. No obstante, las mismas pueden añadir sus propios servicios personalizados donde podrá editarlos o eliminarlos, la peluquería decide qué servicios da a sus usuarios.

- **Gestión de Horarios:**

Se ha creado una tabla dentro de la base de datos con una plantilla prevista con todas las horas laborales que puede tener una peluquería, para que así ellas mismas pueden seleccionar las horas que mejor se adapten a su empresa.

- **Gestión de citas:**

Se localiza una tabla dentro de la base de datos donde se registran todas las citas procedentes de los usuarios, donde el mismo podrá elegir el peluquero, horario y servicios que deseé. En su portal, podrá ver todas las citas pendientes y cancelarlas en el momento.

- **Diseño de la parte gráfica:**

El diseño total de la aplicación se ha realizado utilizando la librería de Bootstrap 5, con poco código de CSS3, para dar estilos personalizados fuera de la primera librería. La aplicación es funcional, usable, intuitiva y sencilla de usar para el usuario.

- **Validación de la entrada de datos:**

Se valida toda la información de datos que introduzca el usuario por parte del cliente, y se muestran mensajes de error correspondientes al campo que se está validando.

- **Validación de citas:**

Se valida que el usuario no tenga ya una cita el mismo día si ya tiene una, evitando que un usuario pueda hacer todas las citas que quiera un día.

- **Validación del código de estado:**


Se validan todas las URLs del proyecto, una vez que se intenta acceder a una parte de la aplicación mediante URL y no está serializada la acción, saldrá un página de error según el código de estado que da en la aplicación.

Líneas futuras

- **Implementación UUID:**
Una propuesta para mejorar la seguridad de la base de datos y su profesionalidad, sería implementar el sistemas UUID (Universally Unique Identifier), cambiándolo por el sistema de IDs de la base de datos.
- **Implementación de pago en web:**
Una propuesta a futuro sería añadir una vista donde el usuario pudiera elegir con qué método desearía pagar, en la tienda o de manera online a través de su banco, paypal o lo que el usuario deseara con lo que pagar, haciendo así que el cliente no tenga que llegar a la peluquería y no se preocupe por los pagos en el caso de que seleccione la opción de pago en línea.
- **Implementar imágenes en los peluqueros:**
Hacer que las propias peluquerías puedan meter una imagen del peluquero para que los usuarios puedan ver y saber con el peluquero que van a tener una cita.
- **Implementar valoraciones en las peluquerías:**
Darle a los usuarios la posibilidad de valorar la peluquería mediante un sistema de 0 a 5 y así poder hacer un filtrado en la aplicación mediante valoración.

WebGrafía

- WebGrafía relacionada con la presentación del proyecto.
 - [Canva](#)
 - [Presentación de Canva](#)
- WebGrafía relacionada con la estructura del proyecto.
 - [Screaming Architecture](#)
 - [Scaffolding para la representación de las vistas](#)
- WebGrafía relacionada con el repositorio del proyecto.
 - [MyReserve - GitHub](#)
 - [Git - Software](#)
 - [Manual de como usar Git y Github \(Video\)](#)
- WebGrafía relacionada con los estilos del proyecto.
 - [Font Awesome](#)
 - [Normalize.css](#)
 - [DaFont](#)
 - [Bootstrap 5](#)

- WebGrafía relacionada con los diagramas de la aplicación.
 -  Diagramas MyReserve ← Enlace para ver todos los diagramas.
 - [mermaid](#)
 - [Editor del diagrama UML](#)
 - [Mirar la image formato SVG](#)
 - [draw.io](#)
 - [excalidraw](#)
- WebGrafía relacionada con la base de datos
 - [UUID \(Universally Unique Identifier\)](#)
- WebGrafía relacionada con el BackEnd de la aplicación.
 - [Base de datos de los países y regiones](#)
 - [Peticiones AJAX desde .NET](#)
 - [Flatpickr](#)
 - [Pasar una lista del controlador a una petición AJAX](#)
 - [Pasar el valor de checkbox al controlador](#)
 - [Cláusula OUTPUT](#)
 - [Video de una página de error personalizada en .NET](#)
 - [Video para realizar un dump de la Base de Datos](#)
 - [Refactoring Guru - Patrones de diseño](#)

My Reserve

Por: Adrián Iglesias Fernández