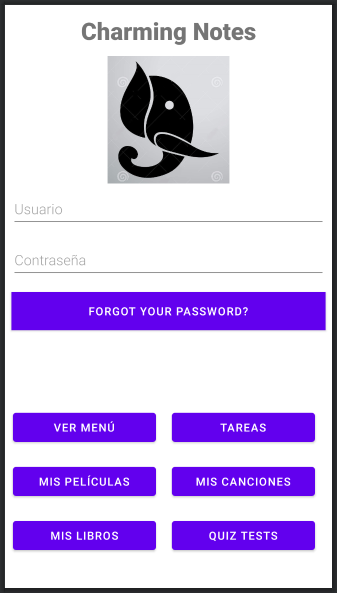
Charming Notes

**Introducción**

Realizaremos el diseño de una aplicación para móvil en lenguaje kotlin usando:

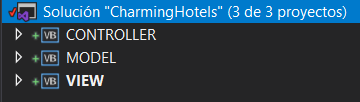
* arquitectura MVM (Android Studio)
* gestión de datos (BBDD SqLite)

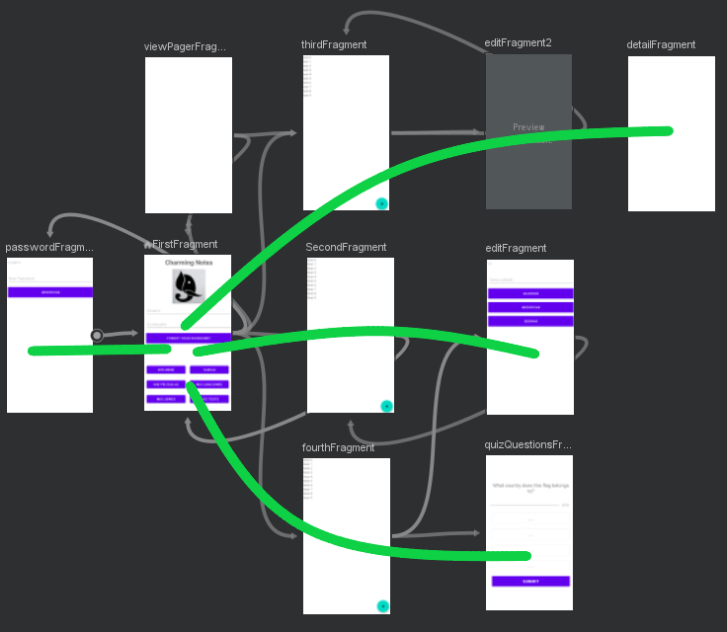
El proyecto consta de una aplicación para gestionar una pequeña agenda de diversas tareas (notas, artículos, documentos de viajes, biblioteca personal, miscelánea, …)



Utilizando el IDE Android Studio crearemos una ***solución*** con el nombre de la aplicación que constará de las siguientes capas según la arquitectura tradicional MVC:

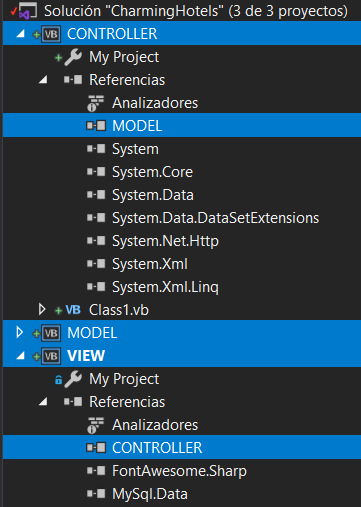
* VIEW -> un proyecto de tipo aplicación de Windows Forms para la capa de presentación.
* CONTROLLER -> un proyecto de tipo biblioteca de clases para la capa de control lógico.
* MODEL -> un proyecto de tipo biblioteca de clases para la capa de acceso de datos o persistencia.





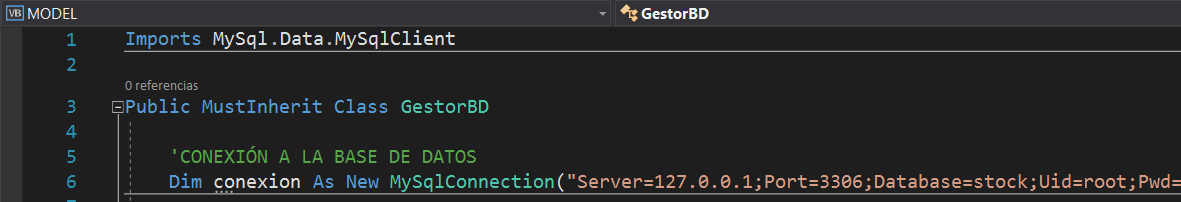
Agregaremos las referencias entre capas donde una capa superior sólo conoce la capa inmediata inferior.

* La capa de presentación sólo conoce la capa de control
* La capa de control sólo tiene acceso a la capa de datos
* La capa de datos no referencia a ninguna capa

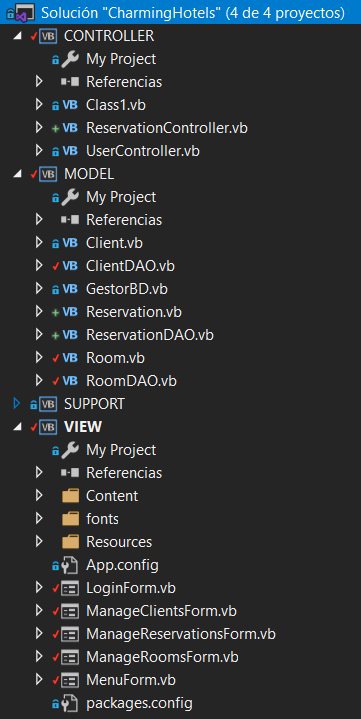
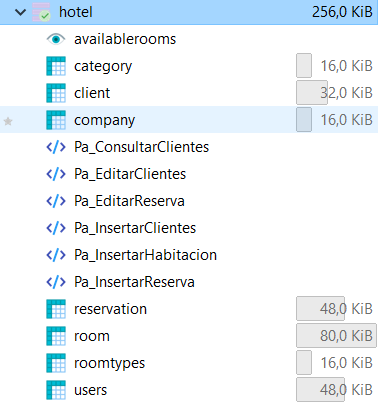


En la capa de datos crearemos una clase abstracta (mustInherit) con el nombre GestoBD que controlará la conexión a SQL mediante:

* la importación de la librería correspondiente (Mysql.Data.MysqlClient)
* métodos que establecen la conexión

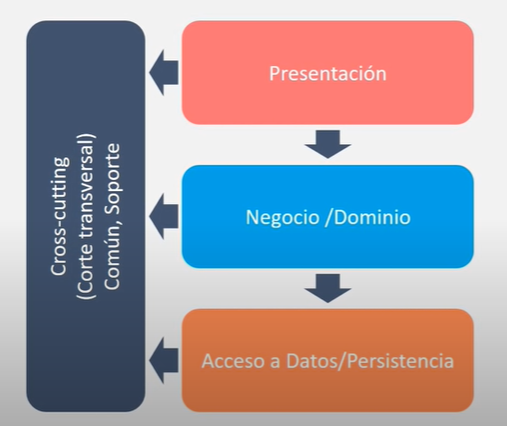


Las posteriores clases DAO, que gestionen objetos de acceso a datos, derivarán de esta clase abstracta para así acceder a sus métodos

En la estructura MVC, para realizar validaciones, permisos o privilegios de usuarios según roles, podemos añadir una 4ª capa común a todas las principales, cuya responsabilidad es la gestión de seguridad en operaciones como:

* Autenticación, autorización
* Almacenamiento en caché
* Comunicación
* Gestión de la configuración, gestión de excepciones
* Registro e instrumentación
* Administración del estado
* Validación, …

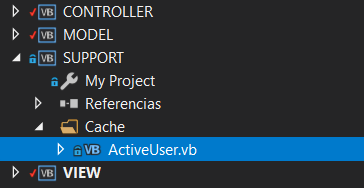
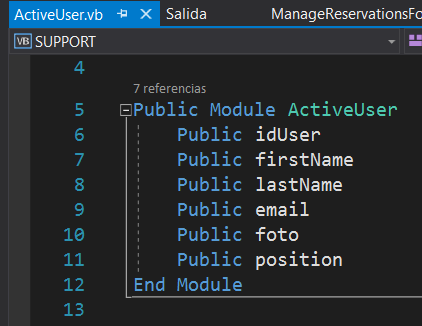


Aquí, guardaremos los datos del usuario logeado en una carpeta *memoria caché* para aplicar las validaciones necesarias en cualquier capa principal.

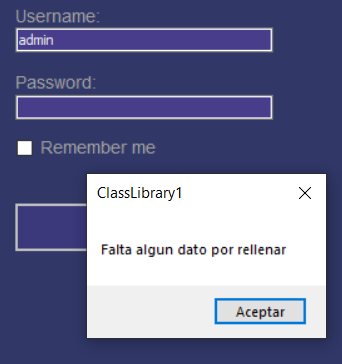
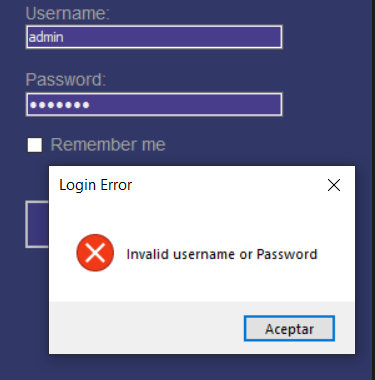
Los módulos son similares a las clases estáticas o clases compartidas

'donde los valores de una clase son permanentes mientras

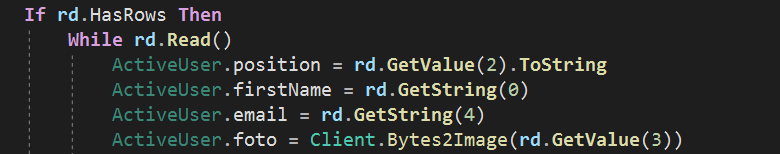
' la aplicación exista

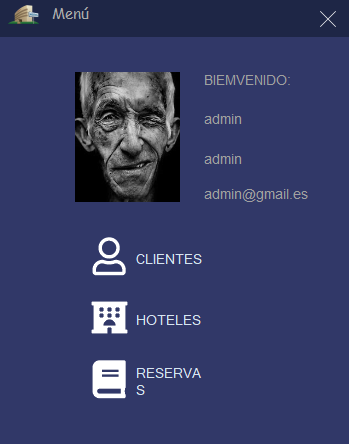
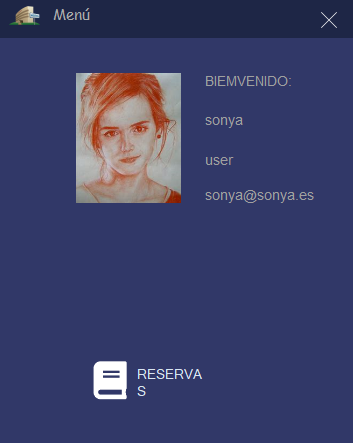
En el formulario de inicio se tendrán que logear los clientes para acceder al menú.

* Llamando al controlador UserController (CONTROLLER), que realizando alguna que otra validación ejecutará la debida consulta en la clase ClientDAO del MODEL
*  

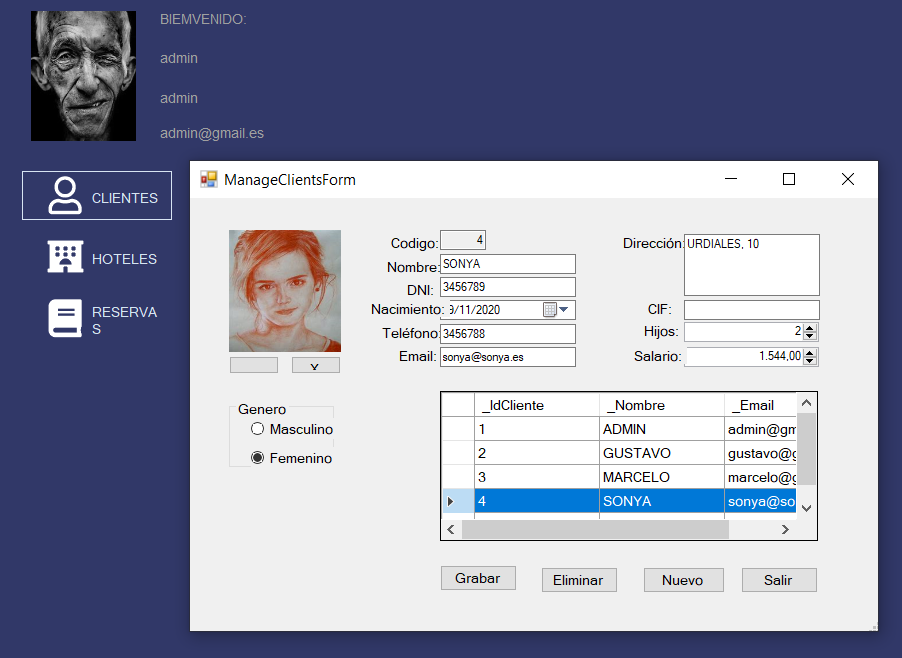
Cuando se accede al Menú aparecerán los datos del cliente en la cabecera gracias a este registro en la memoria cache de la capa soporte.



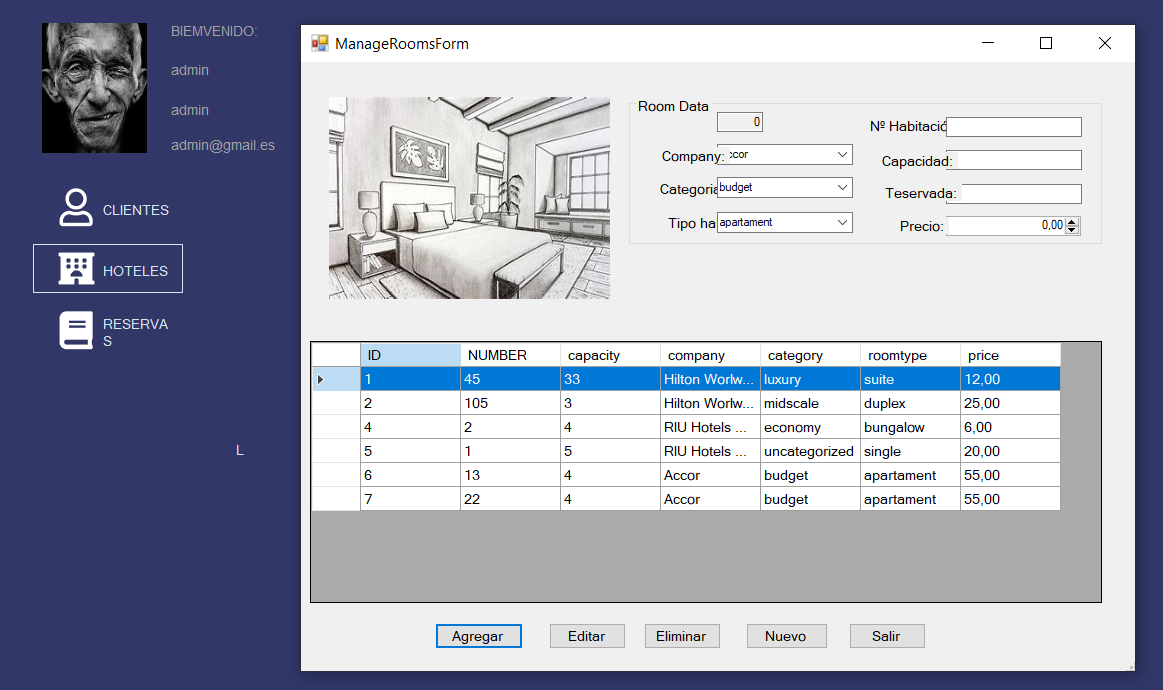
Cada usuario tendrá acceso a determinadas acciones dependiendo del rol (positionn) asignado.

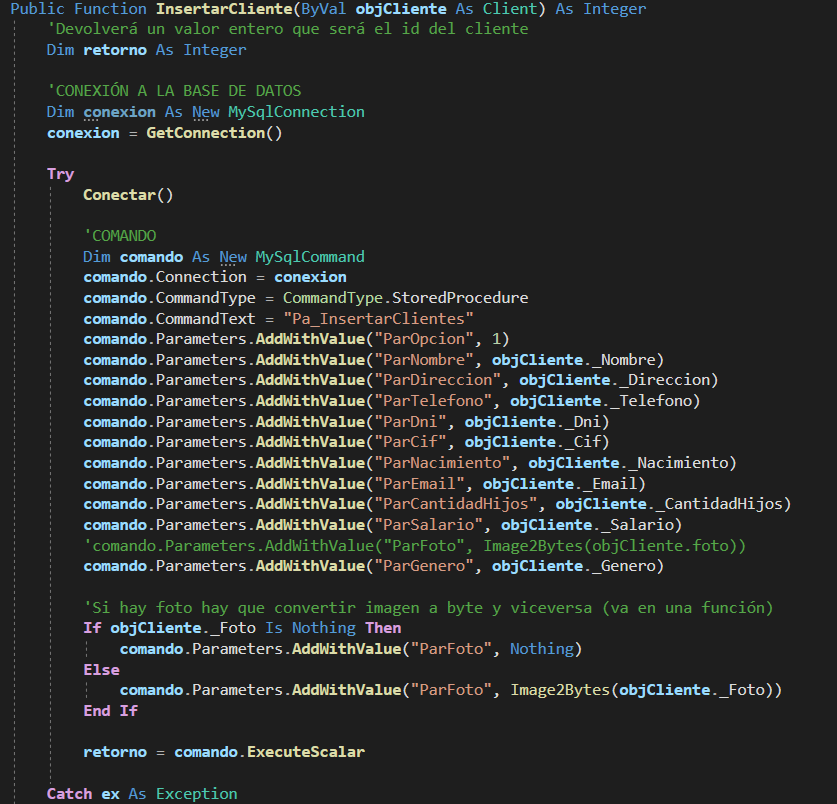
 

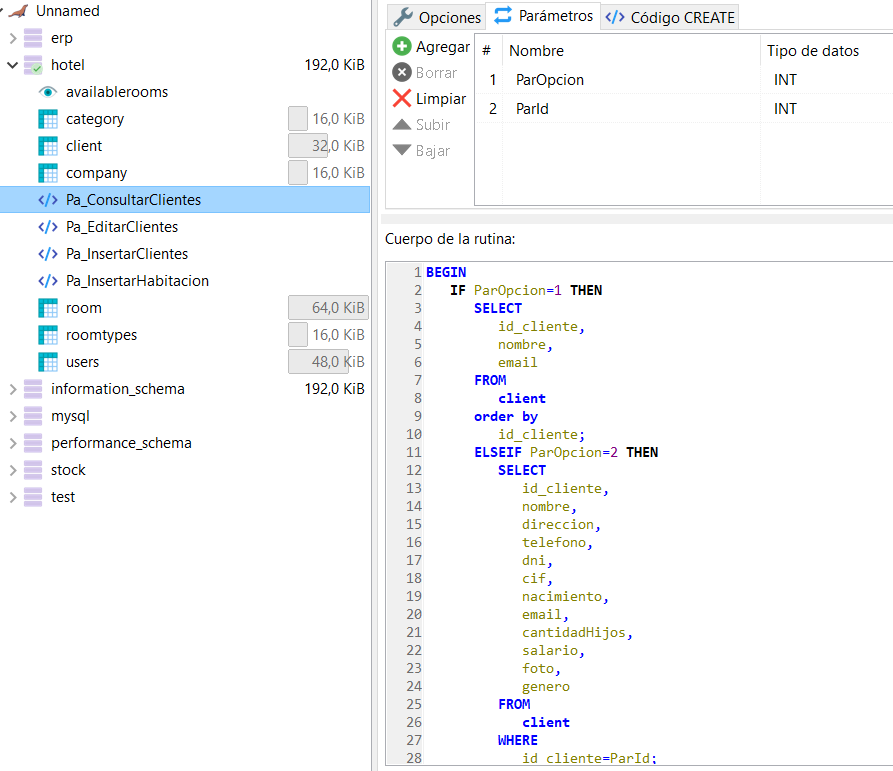
El administrador tendrá acceso a los datos de los clientes y de las habitaciones registradas en la base de datos y tendrá permiso para realizar cualquier operación CRUD necesaria.



* Mediante la selección de la fila en la tabla, se refrescarán los datos en los casilleros y así realizar la operación CRUD deseada.

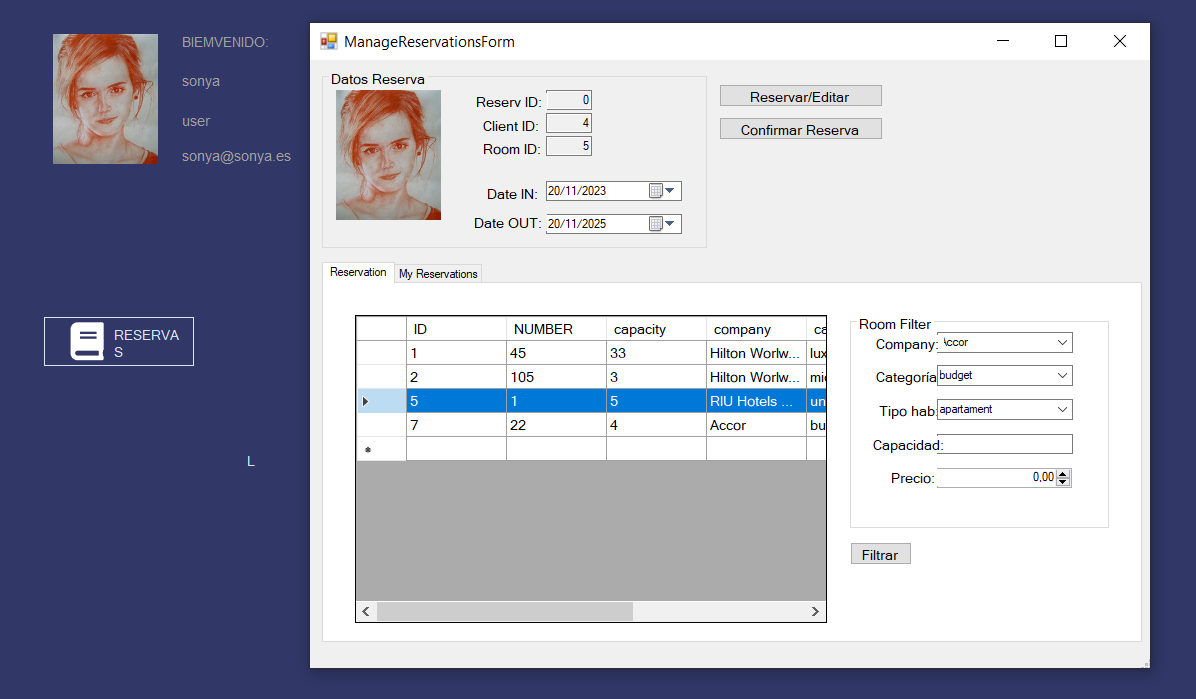


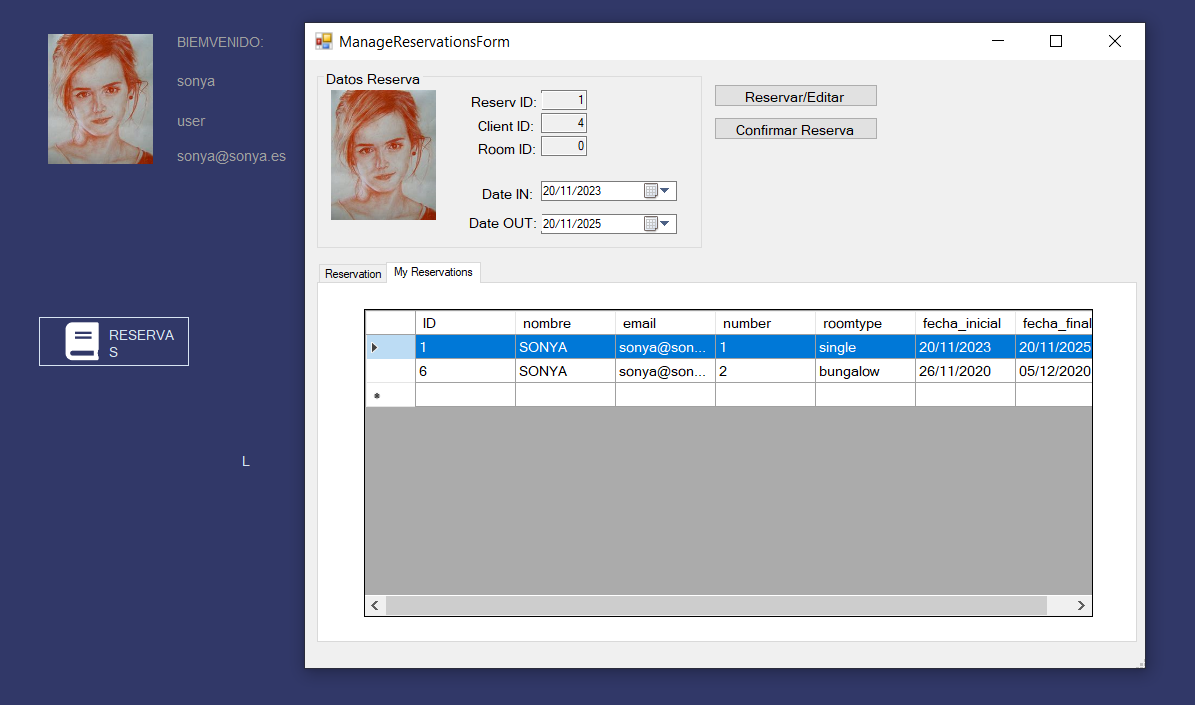




Los clientes sólo tendrán acceso al formulario de reservas donde:

* podrán elegir de la tabla: las habitaciones disponibles (reserved = false)
* deberán indicar la fecha de entrada y de salida (validadas)
* podrán filtrar la tabla según requisitos deseados (categoría, precio, nº de camas,…)





Los datos de las reservas realizadas quedan registradas en la tabla *reservation* que es la relación de las tablas *clientes* y *habitaciones.*

---------------------ANOTACIONES VARIAS------------------

Is it ever more efficient to use sql=string.format(…) rather tan a sqlcommand with parameter?

In this list, the last one item has more priority tan previous in terms of sql injection:

* simple query
* query with string.format
* parameterized sql command query (prepared statements)
* sql stored procedure (SP)

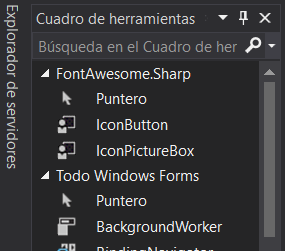
You should use parameters for simplicity, readability and to avoid SQL injections

SqlCommand cmd = new SqlCommand("SELECT \* FROM Signup WHERE pasword = @password AND username = @username", con);

cmd.Parameters.AddWithValue("@password", PasswordTextBox.Text);

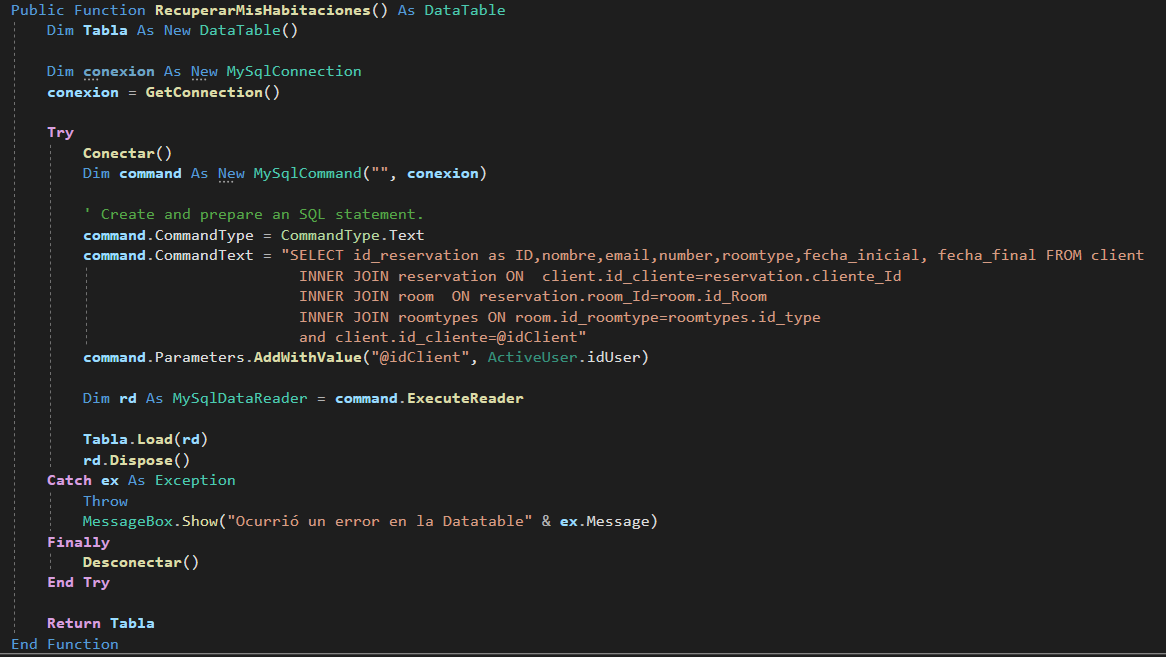
cmd.Parameters.AddWithValue("@username", UserNameTextBox.Text);

Realización de un formulario moderno y multicolor con la integración del paquete Nuget (Font.Awesome.Sharp) que incluye comandos propios.



Public Shared Function RecuperarTodosLosClientes() As BindingList(Of Client)

GridClientes.DataSource = ClienteController.RecuperarTodosLosClientes



Private Sub GridClientes\_SelectionChanged(sender As Object, e As EventArgs) Handles GridClients.SelectionChanged

Private Sub cmbCompany\_SelectedIndexChanged(sender As Object, e As EventArgs) Handles cmbCompany.SelectedIndexChanged

-------------------------------------------------------------------

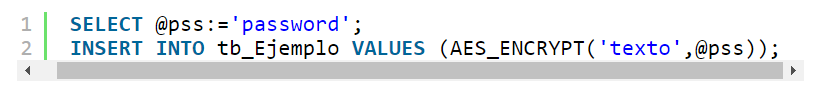
El uso de la encriptación AES en MySQL (la más segura de la que dispone el famoso SGBD) es sencillo, y se realiza mediante dos funciones:

* AES\_ENCRYPT
* AES\_DECRYPT.

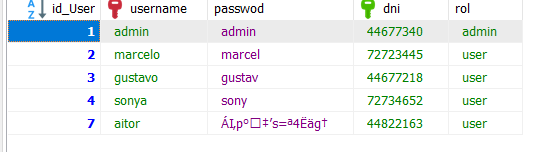
Permite usar AES\_128 o AES\_256 dependiendo de la fuente que usemos

* INSERT INTO tb\_Ejemplo VALUES(AES\_ENCRYPT('valor a encriptar como cadena', 'contraseña'));
* SELECT AES\_DECRYPT(COLUMNA\_ENCRIPTADA, 'contraseña') FROM tb\_Ejemplo;

Por seguridad, podemos guardar la contraseña en una variable del servidor para no enviarla en cada comunicación con la base de datos.



El tipo de datos de esta columna encriptada debe ser Binary/VarBinary



<https://donnierock.com/2012/09/10/encriptacion-aes-en-mysql-y-mariadb/>