### **1. Estructura del Proyecto**

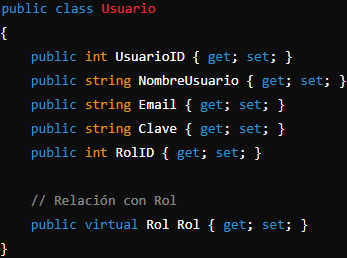
El proyecto sigue una estructura de **capas** separadas por responsabilidad, aplicando los principios SOLID para asegurar un código limpio. Las capas principales incluyen:

* **Modelos (Models)**: Aquí se definen las entidades del sistema (usuarios, roles, permisos).
* **Controladores (Controllers)**: Manejan la lógica de negocio y la comunicación entre la vista y los modelos.
* **Vistas (Views)**: Interfaz de usuario que muestra las páginas web al usuario.
* **Servicios (Services)**: Contienen la lógica de aplicación que interactúa con la base de datos.

### **2. Modelos: Clases y Relaciones**

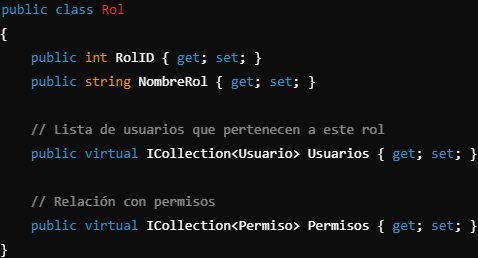
Los **modelos** representan las tablas de la base de datos. Veamos algunos ejemplos de las clases clave:

Clase Usuario



* Esta clase representa la tabla Usuarios.
* La propiedad RolID se utiliza como clave foránea para asociar un usuario con un rol.
* La propiedad Clave almacena la contraseña del usuario, que generalmente debe ser almacenada de forma segura (hash).

#### **Clase Rol**

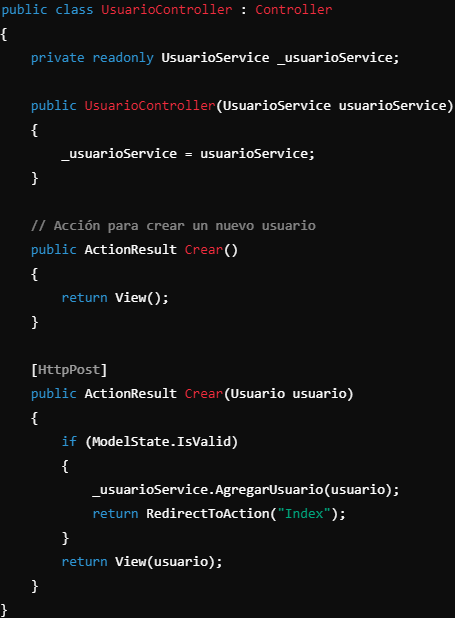


* La clase Rol define los roles dentro del sistema, como "Administrador", "Usuario" o "Editor".
* Esta clase tiene relaciones con las clases Usuario y Permiso, lo que permite asignar permisos a cada rol.

### **3. Controladores: Lógica de Negocio**

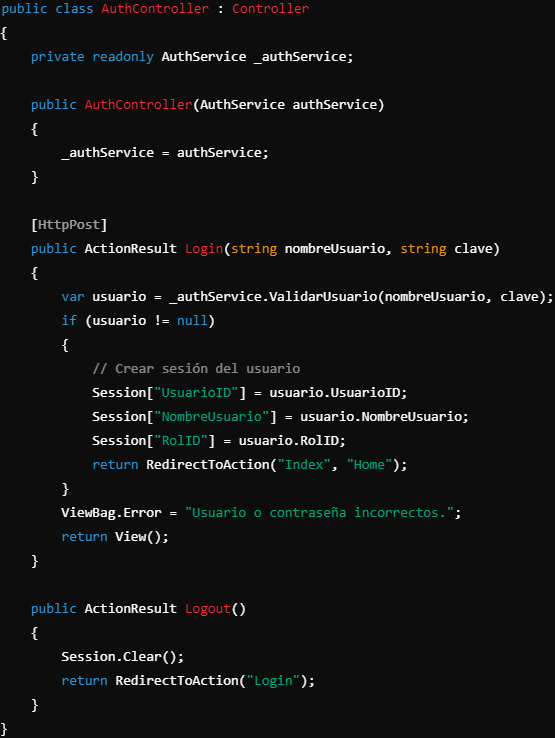
Los **controladores** en ASP.NET MVC actúan como intermediarios entre el modelo y la vista. Los métodos dentro de los controladores reciben solicitudes HTTP y responden con las vistas adecuadas.

#### **Controlador de Usuarios**



* El controlador de usuarios maneja las operaciones CRUD (Crear, Leer, Actualizar, Eliminar) sobre los usuarios.
* El método Crear muestra un formulario para agregar un nuevo usuario y, al recibir la información vía POST, la valida y luego la guarda utilizando un servicio.

Controlador de Autenticación

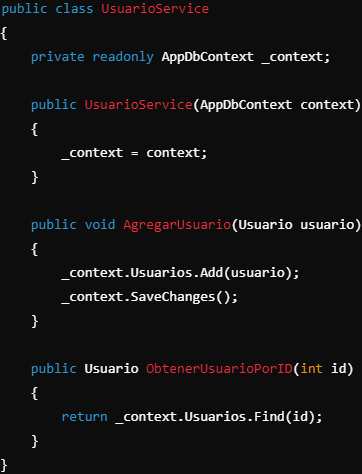


* El **AuthController** maneja la autenticación de usuarios. El método Login valida las credenciales del usuario utilizando el servicio de autenticación.
* Si la autenticación es correcta, se crean las variables de sesión necesarias para gestionar el acceso del usuario durante su sesión.
* El método Logout borra las variables de sesión y redirige al usuario a la página de inicio de sesión.

### **4. Servicios: Lógica de Aplicación**

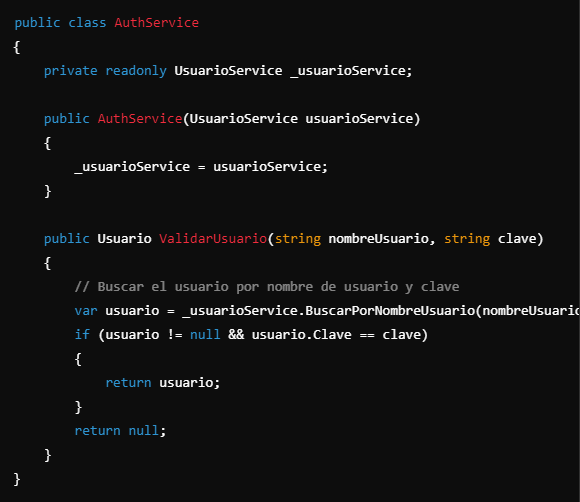
Los **servicios** encapsulan la lógica de negocio del sistema. Por ejemplo, el servicio de usuarios y el servicio de autenticación.

#### **Servicio de Usuarios**



* Este servicio gestiona la lógica para agregar usuarios, obtener usuarios por su identificador, etc.
* El uso del patrón de repositorio aquí se centra en la manipulación directa de las entidades de la base de datos mediante Entity Framework.

#### **Servicio de Autenticación**

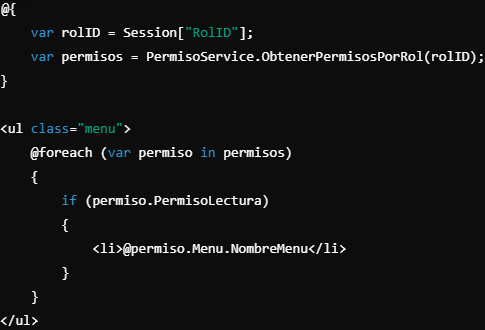


* El servicio de autenticación se encarga de validar las credenciales del usuario.
* Si el nombre de usuario y la contraseña coinciden con los datos almacenados en la base de datos, devuelve un objeto Usuario.

### **5. Interfaz Dinámica: Menús Basados en Permisos**

El sistema muestra menús dinámicos dependiendo de los permisos que tenga el usuario autenticado.

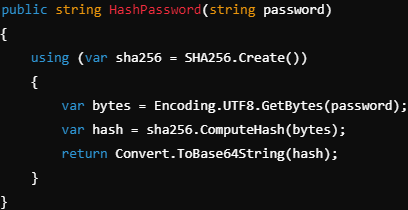
#### **Generación del Menú Dinámico**



* Este código en la vista genera un menú que solo muestra las opciones de menú que el usuario tiene permiso para ver.
* PermisoService.ObtenerPermisosPorRol(rolID) obtiene todos los permisos asociados al rol del usuario.

### **6. Seguridad Adicional: Hashing de Contraseñas**

Para una mayor seguridad, las contraseñas de los usuarios deben estar **hasheadas** antes de almacenarlas en la base de datos. Un ejemplo de cómo hacerlo sería:



* Este método toma la contraseña del usuario y la transforma en un **hash** utilizando el algoritmo SHA-256.
* De esta forma, las contraseñas nunca se almacenan en texto plano en la base de datos.