UNIVERSIDAD PRIVADA DE TACNA

INGENIERIA DE SISTEMAS

TITULO:

Desarrollo de Solución de Mejora de RandyStore

CURSO:

BASE DE DATOS II

DOCENTE:

Ing. Patrick Cuadros Quiroga

Integrantes:

Mejía Rodriguez, Julio Oliver	(2010037899)
Paredes Catacora, Randi Angel	(2013047246)
Herrera Amezquita, Derian Francisco	(2017059489)
Lipa Calabilla, Abraham	(2019064039)

Índice

Resumen

El presente proyecto , desarrollo de una sistema de control de clientes, personal e inventario del gimnasio Randys busca contribuir automatizar los procesos del area de ventas de productos deportivos. Tambien busca llevar un control de los clientes y de el personal con el que cuenta la tienda.

Abstract

The present project, development of a control system for clients, personnel and Randys gym inventory seeks to help automate processes in the area sales of sports products. It also seeks to keep track of customers and staff with whom the company has store.

1. Introduccion

2. Titulo

Sistema de control de inventario y personal RandyStore.

3. Autores

- Mejía Rodriguez, Julio Oliver
- Paredes Catacora, Randi Angel
- Herrera Amezquita, Derian Francisco
- Lipa Calabilla, Abraham

4. Planteamiento del problema

4.1. Problema

La tienda RandyStore no cuenta con un control sobre los productos y ventas que realizan sus empleados por lo que estos guardan infromacion en lugares no muy confiables como cuadernos los cuales se pueden dañar y afectar a la tienda.

4.2. Justificación

La implementación del sistema RandyStore tiene una gran importancia debido a que va a mejorar notablemente la calidad de los datos de ventas generadas por el personal de ventas para mejorar el control de los registros de los productos y de las ventas, así como la seguridad de los datos.

4.3. Alcance

El alcance del proyecto sera a el gerente y los empleados que trabajan en la tienda para que puedan monitoriar y/o controlar los ingresos y ventas de productos relacionados a la tienda.

5. Objetivos

5.1. General

Ayudar a llevar un control sobre el stock de la tienda y mejorar el control de las ventas realizadas por el personal de ventas de la tienda.

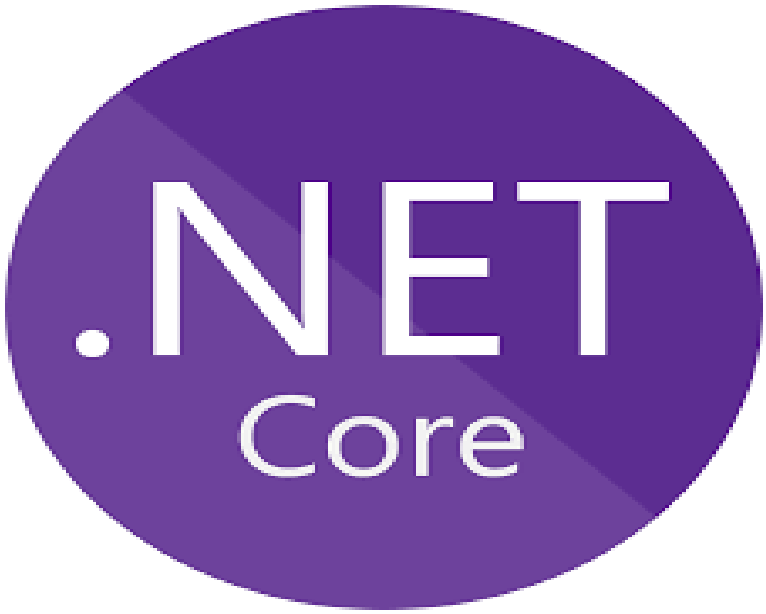
5.2. Específicos

- Añadir seguridad a el sistema mediante un login.
- Control sobre productos ,empleados y clientes.

6. Referencias Teoricas

6.1. Net Core

NET Core es un Framework informático administrado, gratuito y de código abierto para los sistemas operativos Windows, Linux y MacOS. Es un sucesor multiplataforma de .NET Framework. El proyecto es desarrollado principalmente por Microsoft bajo la Licencia MIT.



6.2. ASP.NET Core

ASP.NET Core es un framework web gratis de código abierto y con un mayor rendimiento que ASP.NET, desarrollado por Microsoft y la comunidad. Es un framework modular que se ejecuta completo tanto en el.NET Framework de Windows como en multiplataforma. NET Core.

6.3. MongoDB

MongoDB es un sistema de base de datos NoSQL, orientado a documentos y de código abierto.

En lugar de guardar los datos en tablas, tal y como se hace en las bases de datos relacionales, MongoDB guarda estructuras de datos BSON (una especificación similar a JSON) con un esquema dinámico, haciendo que la integración de los datos en ciertas aplicaciones sea más fácil y rápida.

6.4. API

La interfaz de programación de aplicaciones, conocida también por la sigla API, en inglés, application programming interface, es un conjunto de subrutinas, funciones y procedimientos que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción.

6.5. Controller

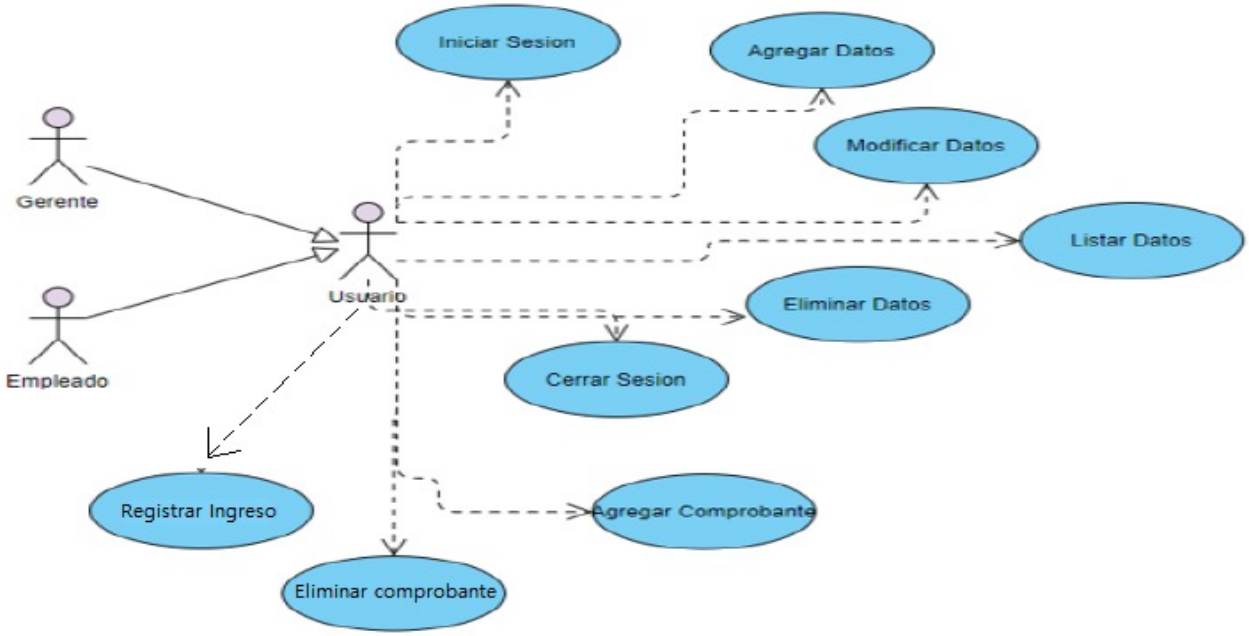
Un Controller o Controlador es una clase que deriva de la clase ControllerBase y es usado por una Web API para manejar las solicitudes.

6.6. Inyección de dependencias

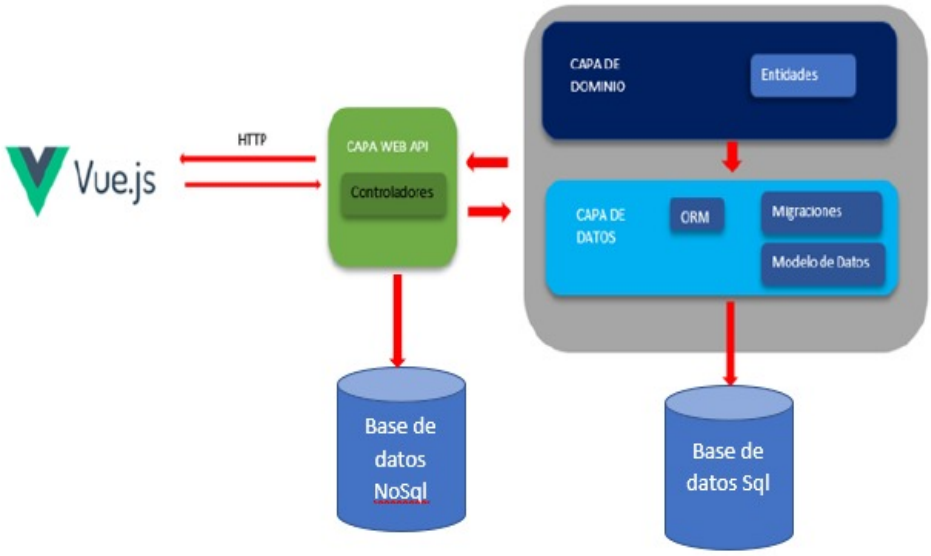
Es un patrón de diseño orientado a objetos, en el que se suministran objetos a una clase en lugar de ser la propia clase la que cree dichos objetos. Esos objetos cumplen contratos que necesitan nuestras clases para poder funcionar (de ahí el concepto de dependencia).

7. Desarrollo de Solución de Mejora

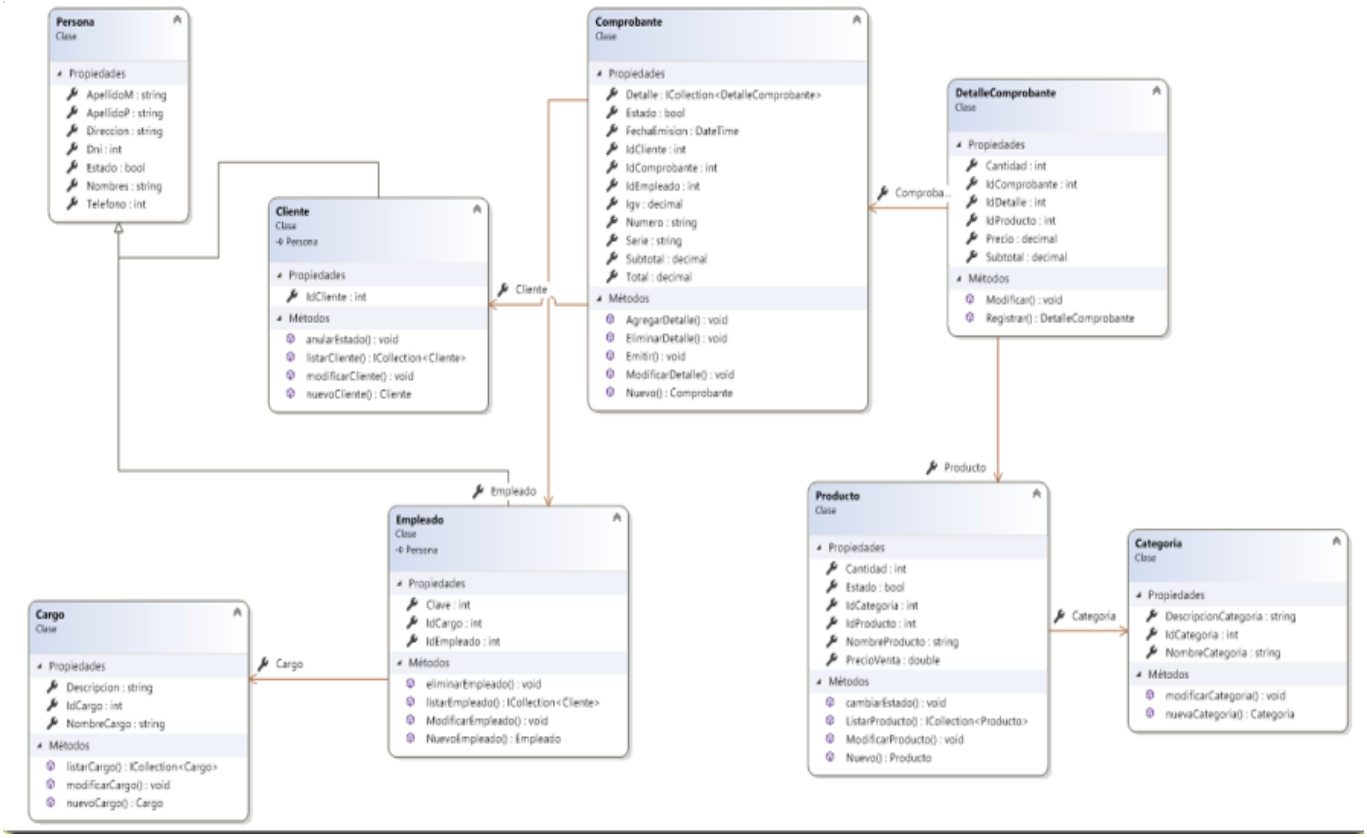
7.1. Casos de Uso de la aplicación



7.2. Diagrama de Arquitectura de la aplicación



7.3. Diagrama de Clases de la aplicación



8. Diccionario de datos

BASE DE DATOS RELACIONAL

Tabla de cargos de empleados

Nombre de la Tabla:	Cargo	**Fecha creacion:** 18/11/2020
Descripción:	Tabla que contendra los tipos de cargos de empleado	
CAMPO	**TIPO DE DATO**	**DESCRIPCION**
IdCargo	int	Clave unica de cargo
NombreCargo	varchar	Nombre del cargo
Descripcion	varchar	Descripcion del cargo
		Campos clave: IdCargo

Tabla de categorias de producto

Nombre de la Tabla:	Categoria	**Fecha creacion:** 18/11/2020
Descripción:	Tabla que contendra los tipos de productos	
CAMPO	**TIPO DE DATO**	**DESCRIPCION**
IdCategoria	int	Clave unica de categoria
Descripcion	varchar	Descripcion de la categoria
		Campos clave: IdCategoria

Tabla de cliente

Nombre de la Tabla:	Cliente	**Fecha creacion:** 18/11/2020
Descripción:	Tabla que tendra los datos personales del cliente	
CAMPO	**TIPO DE DATO**	**DESCRIPCION**
IdCliente	int	Clave unica del cliente
Nombre	varchar	Nombre completo del cliente
Apaterno	varchar	Apellido paterno del cliente
Amaterno	varchar	Apellido materno del cliente
Direccion	varchar	Direccion del cliente
Dni	int	Numero de dni del cliente
Celular	int	Numero de telefono del cliente
		Campos clave: IdEmpleado

Tabla de comprobante

Nombre de la Tabla:	Comprobante	**Fecha creacion:** 18/11/2020
Descripción:	Tabla que contendra los comprobantes realizados	
CAMPO	**TIPO DE DATO**	**DESCRIPCION**
IdComprobante	int	Clave unica del comprobante
FechaEmision	DateTime	Fecha de creacion del comprobante
Total	decimal	Total de dinero a pagar
IdCliente	int	Clave referencial a la tabla cliente
IdEmpleado	int	Clave referencial a la tabla empleado
		Campos clave:IdComprobante,IdEmpleado,IdCliente

Tabla del detalle del comprobante

Nombre de la Tabla:	DetalleComprobante	**Fecha creacion:** 18/11/2020
Descripción:	Tabla que tendra el detalle de cada comprobante	
CAMPO	**TIPO DE DATO**	**DESCRIPCION**
IdDetalle	int	Clave unica de detalle
IdComprobante	int	Clave referencial a la tabla comprobante
IdProducto	int	Clave referencial a la tabla producto
Precio	decimal	Costo unitario del producto
Cantidad	int	Cantidad del producto vendido
Total	decimal	Total a pagar por los productos
		Campos clave:IdComprobante,IdDetalle,IdProducto

Tabla del empleado

Nombre de la Tabla:	Empleado	**Fecha creacion:** 18/11/2020
Descripción:	Tabla que contendra datos personales del empleado	
CAMPO	**TIPO DE DATO**	**DESCRIPCION**
IdEmpleado	int	Clave unica del empleado
Nombre	varchar	Nombre completo del empleado
Apaterno	varchar	Apellido paterno del empleado
Amaterno	varchar	Apellido materno del empleado
Direccion	varchar	Direccion del empleado
Dni	int	Numero de dni del empleado
Celular	int	Numero de telefono del empleado
Clave	int	Clave generada para el empleado
IdCargo	int	Clave referencial a la tabla cargo
		Campos clave: IdEmpleado, IdCargo

Tabla del producto

Nombre de la Tabla:	Producto	Fecha creacion: 18/11/2020
Descripción:	Tabla que tendra informacion de los productos	
CAMPO	TIPO DE DATO	DESCRIPCION
IdProducto	int	Clave principal de producto
NombreProducto	varchar	Nombre del producto
Cantidad	varchar	Cantidad de stock del producto
PrecioVenta	varchar	Precio de venta del producto
IdCategoria	varchar	Clave referencial a la tabla Categoria
		Campos clave: IdProducto,IdCategoria

BASE DE DATOS NO RELACIONAL

Nombre de la colección	Ingreso	Fecha creacion: 13/12/2020
Descripcion:	Colección donde se guardan los registros de accesos al sistema	
CAMPO	TIPO DE DATO	DESCRIPCION
id	ObjectId	Clave unica de Ingreso
usuario	string	Nombre del usuario
horaIngreso	Date	Fecha del acceso al sistema

9. Cadena de conexion

Para la conexion a la base de datos no relacional guardamos los datos de conexión de MongoDB en el fichero appsettings.json. Guardamos el servidor, la base de datos y la colección en que se guardará.

```
"IngresoSetting": {
  "Server": "mongodb+srv://m001-student:derian123@cluster0.bnn1n.mongodb.net/Randy?retryWrites=true&w=majority",
  "DataBase": "Randy",
  "Collection": "Ingresos"
}
```

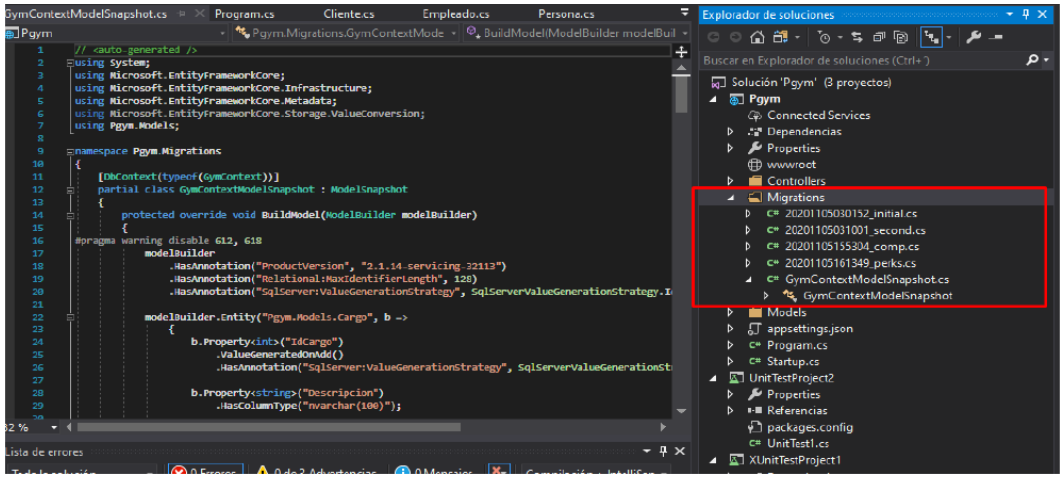
En Startup.cs, configuramos el servicio con la configuración en el json.

```
public void ConfigureServices(IServiceCollection services)
{



    services.AddDbContext<GymContext>(options => options.UseSqlServer(Configuration.GetConnectionString("Conexion")));
    services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.Version_2_1).AddJsonOptions(options =>
    {
        //Eliminar referencias ciclicas
        options.SerializerSettings.ReferenceLoopHandling = Newtonsoft.Json.ReferenceLoopHandling.Ignore;

    });
    services.AddCors();

    services.Configure<IngresoSetting>
        (Configuration.GetSection(nameof(IngresoSetting)));
    services.AddSingleton<InterfaceIngreso>
        (d => d.GetRequiredService<IOptions<IngresoSetting>>().Value);
    services.AddSingleton<IngresoServices>();
}
```

En el controlador, declaramos dos operaciones: Get (que devuelve todos los registros de Ingreso) y Post (que permite registrar un ingreso).

```
using Microsoft.AspNetCore.Mvc;

namespace Api_Web.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    public class IngresosController : ControllerBase
    {
        public IngresoServices _ingresos;

        public IngresosController(IngresoServices ingresos)
        {
            _ingresos = ingresos;
        }

        [HttpGet]
        public ActionResult<List<Ingreso>> Get()
        {
            return _ingresos.Get();

        }
        [HttpPost]
        public ActionResult<Ingreso> Create(Ingreso ing)
        {
            _ingresos.Create(ing);
            return Ok(ing);
        }

    }
}
```

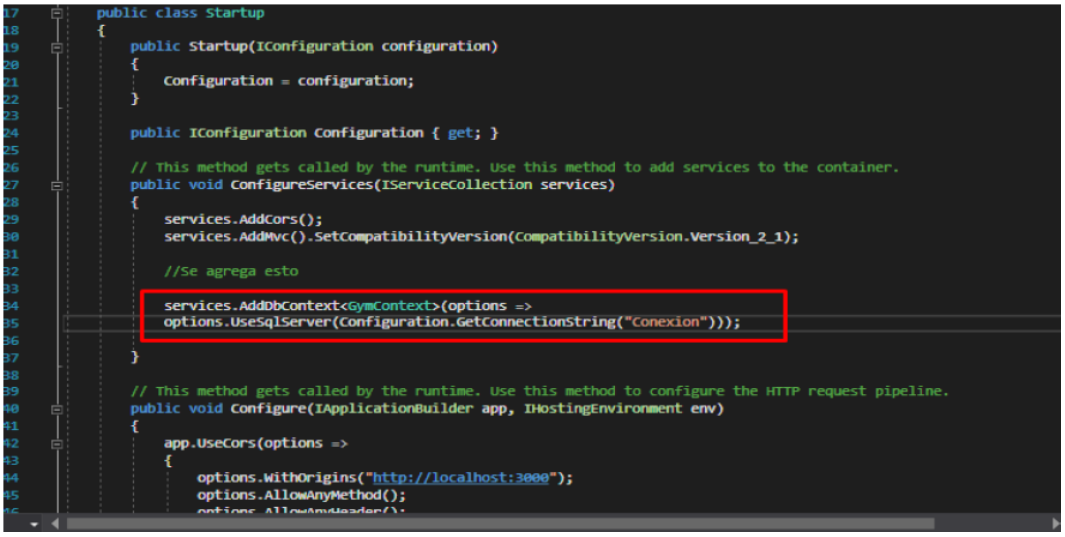
Para la conexion a la base de datos relacionas SqlServer usamos. Dbcontext donde esta el contexto de nuestra base de datos y las migraciones prueba que realizamos.



Cadena de conexion para el orm que va en el json appsettings.

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Warning"
    }
  },
  "AllowedHosts": "*",

  "ConnectionStrings": {


    "Conexion": "Server =localhost; Database=Agym; Trusted_Connection=True; MultipleActiveResultSets=True;"


  }
}
```

Luego en la clase startup se coloca la cadena de conexion string que hicimos anteriormente para que el sistema lo reconozca y nuestro DBcontext tambien.



```
public class Startup
{
    public Startup(IConfiguration configuration)
    {
        Configuration = configuration;
    }

    public IConfiguration Configuration { get; }

    // This method gets called by the runtime. Use this method to add services to the container.
    public void ConfigureServices(IServiceCollection services)
    {
        services.AddCors();
        services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.Version_2_1);

        //Se agrega esto

        services.AddDbContext<GymContext>(options =>
        options.UseSqlServer(Configuration.GetConnectionString("Conexion")));

    }

    // This method gets called by the runtime. Use this method to configure the HTTP request pipeline.
    public void Configure(IApplicationBuilder app, IHostingEnvironment env)
    {
        app.UseCors(options =>
        {
            options.WithOrigins("http://localhost:3000");
            options.AllowAnyMethod();
```

Nuestras migraciones se realizaron para SQLServer.