

UNIVERSIDAD PRIVADA-DE-TACNA



INGENIERIA DE SISTEMAS

TITULO:

**TRABAJO FINAL DE UNIDAD**

**CURSO:**

BASE DE DATOS II

**DOCENTE(ING):**

Patrick Cuadros Quiroga

Integrantes:

|                                |              |
|--------------------------------|--------------|
| Arlyn Cotrado Coaquira         | (2016054466) |
| Yaneth Virginia Aquino Huallpa | (2017059286) |
| Sharon Sosa Bedoya             | (2016054460) |
| Marlon Villegas Arando         | (2015053890) |

# Índice

|                                  |           |
|----------------------------------|-----------|
| <b>1. PROBLEMA</b>               | <b>1</b>  |
| <b>2. MARCO TEORICO</b>          | <b>2</b>  |
| 2.1. API REST . . . . .          | 2         |
| 2.2. Entity Framework . . . . .  | 2         |
| 2.3. Consultas LINQ . . . . .    | 2         |
| 2.4. Pruebas Unitarias . . . . . | 2         |
| <b>3. DESARROLLO</b>             | <b>3</b>  |
| 3.1. Analisis . . . . .          | 3         |
| 3.2. Diseño . . . . .            | 3         |
| 3.3. Pruebas . . . . .           | 4         |
| 3.4. API/Postman . . . . .       | 8         |
| <b>4. REFERENCIAS</b>            | <b>14</b> |

# 1. PROBLEMA

El siguiente trabajo se desarrolla con el Sistema de Gestion de Gimnasio, el cual busca que el gimnasio adquiera una mejor organización ahorrando tiempo y trabajo para recolección de información, permitiendo:

- La gestión de los entrenadores, miembros, planes, usuarios.
- Gestión de las clases que se brinden asignando un entrenador a una clase, evitando el cruce de horarios en diferentes sucursales.
- La venta de distintos tipos de planes.
- La realización de reportes sobre las ventas realizadas del día, del mes, y sobre los clientes correspondientes a cada local.
- El sistema permitirá realizar la autenticación de los usuarios.

La motivación principal de este proyecto es que el gimnasio adquiera una mejor organización, lo que le permitirá realizar sus tareas sin inconvenientes.

## 2. MARCO TEORICO

### 2.1. API REST

Ha pasado más de una década desde que Roy Fielding, un científico informático estadounidense y uno de los autores principales de la especificación HTTP, introdujo Representational State Transfer (REST) como un estilo de arquitectura. A lo largo de los años, REST ha ganado impulso gracias a su popularidad para la creación de servicios web.

Al no tener estado, y al crear identificadores únicos para permitir el almacenamiento en caché, la creación de capas y la capacidad de lectura, las API REST hacen uso de los verbos HTTP existentes (GET, POST, PUT y DELETE) para crear, actualizar y eliminar nuevos recursos. El término REST se usa con frecuencia para describir cualquier URL que devuelva JSON en lugar de HTML.[3]

### 2.2. Entity Framework

Entity Framework es un marco de ORM de código abierto para aplicaciones .NET admitidas por Microsoft. Permite a los desarrolladores trabajar con datos utilizando objetos de clases específicas del dominio sin centrarse en las tablas y columnas de la base de datos subyacente donde se almacenan estos datos. Con el Entity Framework, los desarrolladores pueden trabajar en un nivel más alto de abstracción cuando tratan con datos, y pueden crear y mantener aplicaciones orientadas a datos con menos código en comparación con las aplicaciones tradicionales.[2]

### 2.3. Consultas LINQ

Permite consultar cualquier objeto enumerable en ADO.NET mediante el uso del modelo de programación de Language-Integrated Query (LINQ) [1] para recuperar datos de la base de datos subyacente. El proveedor de la base de datos traducirá estas consultas LINQ al lenguaje de consulta específico de la base de datos (por ejemplo, SQL para una base de datos relacional).

Hay 3 tecnologías ADO.NET LINQ distintas:

- LINQ to DataSet: proporciona una capacidad de consulta mas rica y optimizada sobre DataSet.
- LINQ to SQL: permite consultar directamente los esquemas de las base de datos de SQL Server.
- LINQ to Entities: permite consultar Entity Data Model.

### 2.4. Pruebas Unitarias

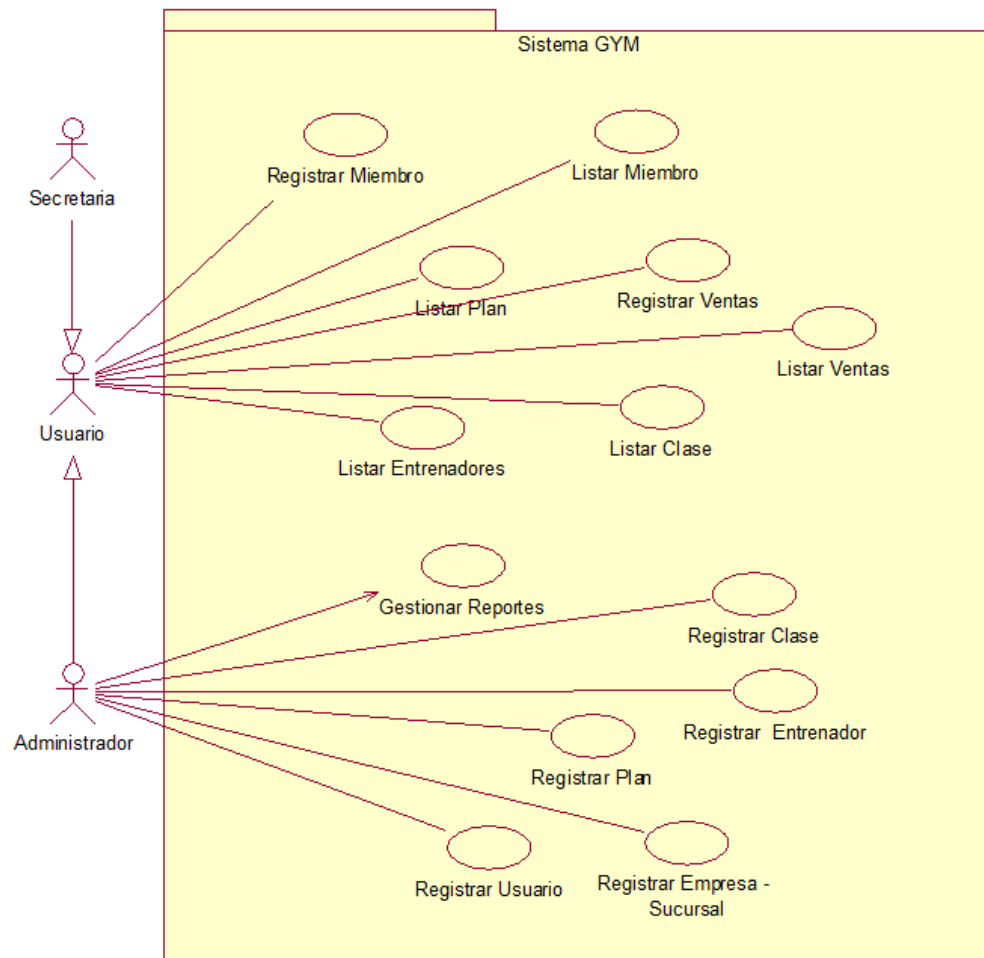
Las pruebas unitarias son la mejor forma de probar el código de una aplicación a lo largo de su desarrollo. Estas pruebas permiten asegurar que los métodos devuelven resultados correctos, teniendo en cuenta los argumentos que se le pasan y, además, se pueden utilizar para hacer Test Driven Development. Se trata de una técnica en la que se escriben antes que las clases y los métodos.[4]

No obstante, tras realizar la integración con otros módulos deberá revisarse de nuevo la interfaz.

### 3. DESARROLLO

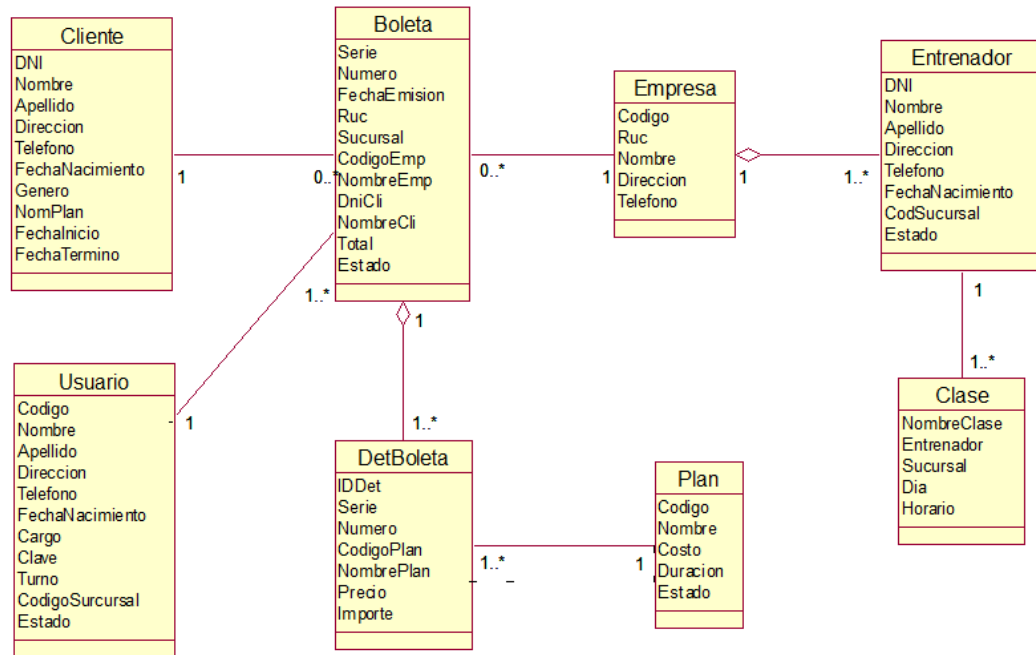
#### 3.1. Analisis

- Casos de Uso

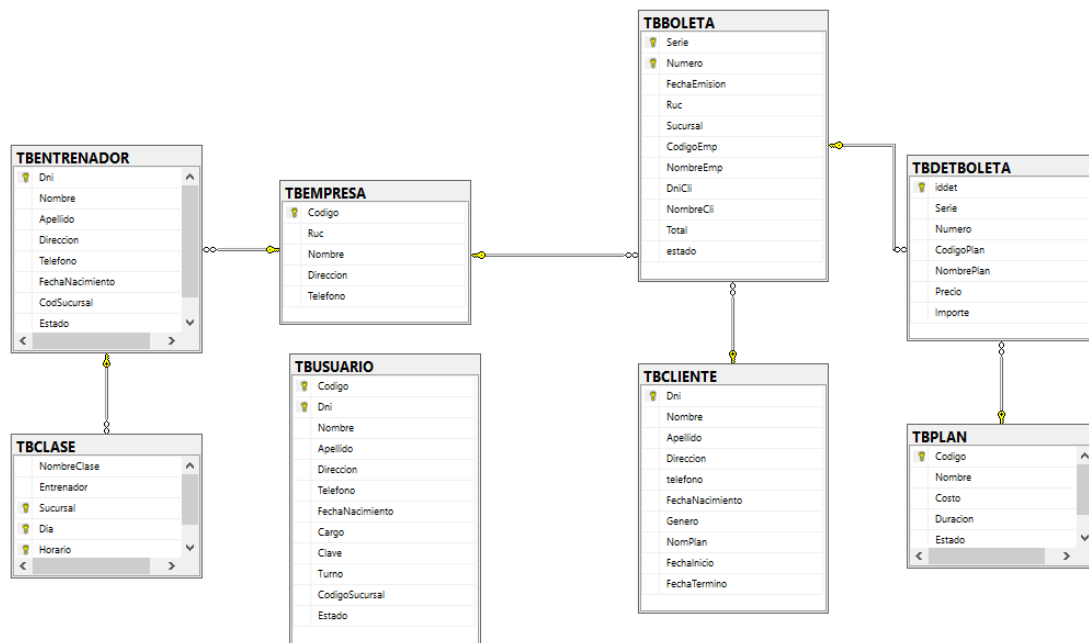


#### 3.2. Diseño

- Diagrama de Clases



– Modelo Entidad Relación



### 3.3. Pruebas

Las pruebas realizadas fueron las siguientes:

– ObtenerAlEntrenadorPorCodigo

```

0 referencias
public void ObtenerAlEntrenadorPorCodigo()
{
    var ctx = new GymContext();
    var entrenador = ctx.TBENTRENADORES.Find("70405060");
    Assert.IsNotNull(entrenador);
}

```

Sentencia SQL generada:

```

RPC:Completed      exec sp_executesql N'SELECT TOP (2)... Ent
<
exec sp_executesql N'SELECT TOP (2)
[Extent1].[Dni] AS [Dni],
[Extent1].[Nombre] AS [Nombre],
[Extent1].[Apellido] AS [Apellido],
[Extent1].[Direccion] AS [Direccion],
[Extent1].[Telefono] AS [Telefono],
[Extent1].[FechaNacimiento] AS [FechaNacimiento],
[Extent1].[CodSucursal] AS [CodSucursal],
[Extent1].[Estado] AS [Estado]
FROM [dbo].[TBENTRENADOR] AS [Extent1]
WHERE [Extent1].[Dni] = @p0',N'@p0 varchar(8000)',@p0='70405060'

```

– AgregarDosPlanesdeGimnasio

```

public void AgregarDosPlanesdeGimnasio()
{
    IList<TBPLAN> nuevoPlan = new List<TBPLAN>()
    {
        new TBPLAN() {Codigo = "P02", Nombre = "Plan 4 meses+ 3 meses gratis", Costo = 200, Duracion = 3, Estado = "Activo" },
        new TBPLAN() {Codigo = "P03", Nombre = "Planverano", Costo = 250, Duracion = 3, Estado = "Activo" }
    };
    using (var context = new GymContext())
    {
        context.TBPLANS.AddRange(nuevoPlan);
        context.SaveChanges();
    }
    Assert.IsNotNull(nuevoPlan);
}

```

Sentencia SQL generada:

```

RPC:Completed      exec sp_executesql N'INSERT [dbo].[TBPLAN]([Codigo], [Nombre], [Costo], [Duraci...
<
exec sp_executesql N'INSERT [dbo].[TBPLAN]([Codigo], [Nombre], [Costo], [Duracion], [Estado])
VALUES (@0, @1, @2, @3, @4)
',N'@0 varchar(3),@1 varchar(30),@2 decimal(6,2),@3 int,@4 varchar(10)',@0='P02',@1='Plan 4 meses+ 3 meses
gratis',@2=200.00,@3=3,@4='Activo'

```

```
RPC:Completed      exec sp_executesql N'INSERT [dbo].[TBPLAN]([Codigo], [Nombre], [Costo], [Duraci...
<
exec sp_executesql N'INSERT [dbo].[TBPLAN]([Codigo], [Nombre], [Costo], [Duracion], [Estado])
VALUES (@0, @1, @2, @3, @4)
',N'@0 varchar(3),@1 varchar(30),@2 decimal(6,2),@3 int,@4 varchar(10)',@0='P03',@1='Planverano',@2=250.00,@3=3,@4='Activo'
```

– ObtenerListaDeClasesPorNombreAscendiente

```
public void ObtenerListaDeClasesPorNombreAscendiente()
{
    using (var ctx = new GymContext())
    {
        var clases = ctx.TBCLASE.OrderBy(s => s.NombreClase).ToList();
    }
}
```

Sentencia SQL generada:

```
SQL:BatchCompleted      SELECT      [Extent1].[Sucursal] AS [Sucursal],
<
SELECT
    [Extent1].[Sucursal] AS [Sucursal],
    [Extent1].[Dia] AS [Dia],
    [Extent1].[Horario] AS [Horario],
    [Extent1].[NombreClase] AS [NombreClase],
    [Extent1].[Entrenador] AS [Entrenador]
FROM [dbo].[TBCLASE] AS [Extent1]
ORDER BY [Extent1].[NombreClase] ASC
```

– BuscarTodaslasBoletasPorEmpleado



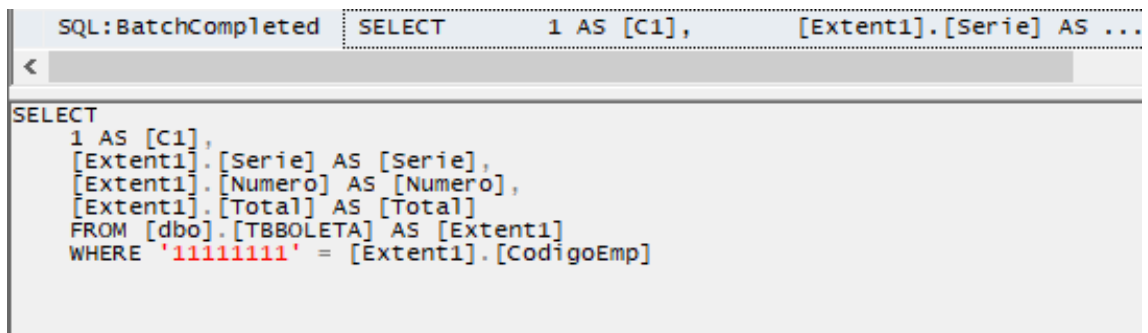
```

public void BuscarTodaslasBoletasPorEmpleado()
{
    using (var ctx = new GymContext())
    {
        var anonymousObjResult = from s in ctx.TBBOLETA
                                where s.CodigoEmp == "3"
                                select new
                                {
                                    SerieBoleta = s.Serie,
                                    NumeroBoleta = s.Numero,
                                    Detalle = s.detalleboleta
                                };

        foreach (var obj in anonymousObjResult)
        {
            Console.WriteLine(obj.SerieBoleta, obj.NumeroBoleta);
        }
    }
}

```

Sentencia SQL generada:



SQL:BatchCompleted SELECT 1 AS [C1], [Extent1].[Serie] AS ...

```

SELECT
    1 AS [C1],
    [Extent1].[Serie] AS [Serie],
    [Extent1].[Numero] AS [Numero],
    [Extent1].[Total] AS [Total]
FROM [dbo].[TBBOLETA] AS [Extent1]
WHERE '11111111' = [Extent1].[CodigoEmp]

```

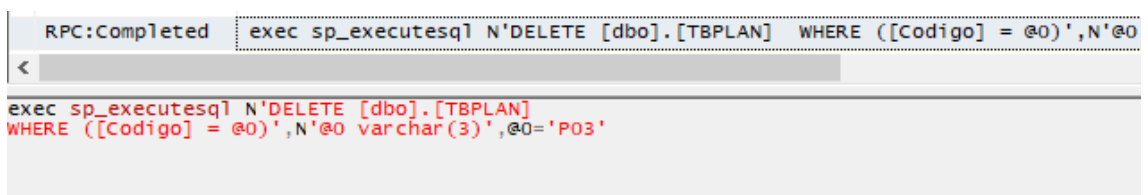
– EliminarUnPlanDeGimnasioPorCodigo

```

0 referencias
public void EliminarUnPlanDeGimnasioPorCodigo()
{
    using (var context = new GymContext())
    {
        var eliminarplan = context.TBPLANS.Where(c => c.Codigo == "P03").FirstOrDefault();
        context.TBPLANS.Remove(eliminarplan);
        context.SaveChanges();
    }
}

```

Sentencia SQL generada:



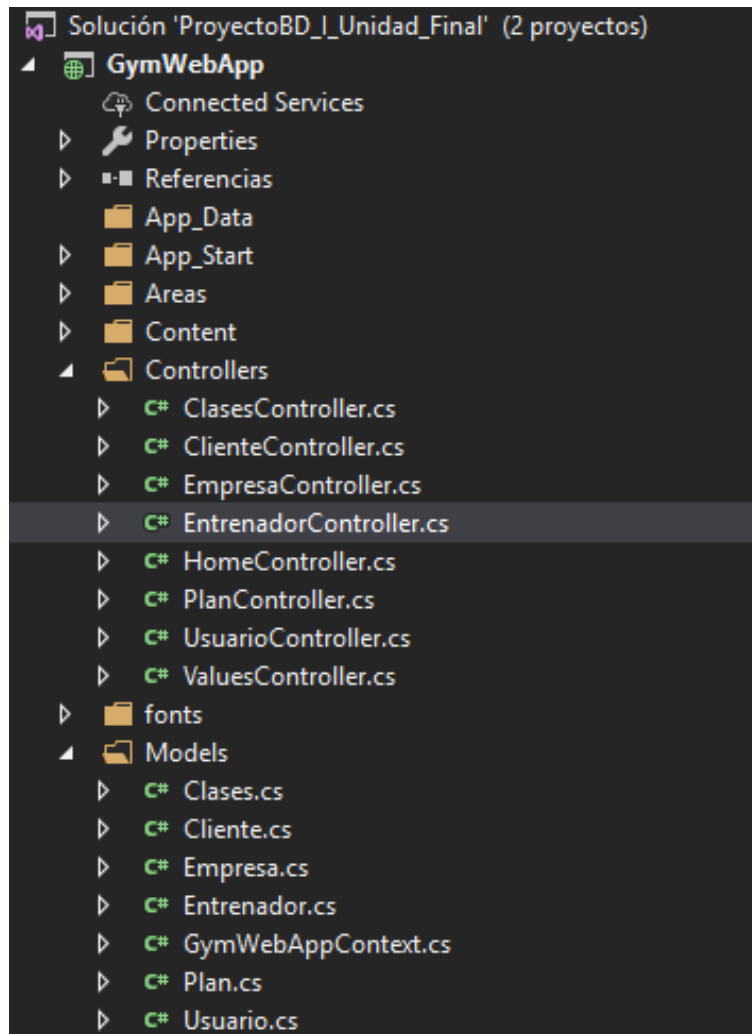
RPC:Completed exec sp\_executesql N'DELETE [dbo].[TBPLAN] WHERE ([Codigo] = @0)',N'@0

```

exec sp_executesql N'DELETE [dbo].[TBPLAN]
WHERE ([Codigo] = @0)',N'@0 varchar(3)',@0='P03'

```

### 3.4. API/Postman



- Del controlador Cliente, hacemos clic en POST Cliente para agregar un registro.

#### Cliente

| API                                       | Description                 |
|---|-----------------------------|
| <a href="#">GET api/Cliente</a>           | No documentation available. |
| <a href="#">GET api/Cliente?Dni={Dni}</a> | No documentation available. |
| <a href="#">POST api/Cliente</a>          | No documentation available. |
| <a href="#">PUT api/Cliente?Dni={Dni}</a> | No documentation available. |
| <a href="#">DELETE api/Cliente/{id}</a>   | No documentation available. |

Copiamos el formato de la cadena, y luego la pegamos en Postman, donde agregaremos un nuevo cliente.

application/json, text/json

Sample:

```
{
  "Dni": 1,
  "Nombre": "sample string 2",
  "Apellido": "sample string 3",
  "Direccion": "sample string 4",
  "Telefono": "sample string 5",
  "FechaNacimiento": "2019-05-02T01:18:23.4520698-05:00",
  "Genero": "sample string 7",
  "NomPlan": "sample string 8",
  "FechaInicio": "2019-05-02T01:18:23.4520698-05:00",
  "FechaTermino": "2019-05-02T01:18:23.4520698-05:00"
}
```

POST http://localhost:52844/api/Cliente Send

```
3  "Nombre": "Luis",
4  "Apellido": "Fernandez Medina",
5  "Direccion": "Calle Junin 1974",
6  "Telefono": "645478",
7  "FechaNacimiento": "1999-05-24",
8  "Genero": "M",
9  "NomPlan": "Verano",
10 "FechaInicio": "2019-05-02",
11 "FechaTermino": "2019-05-28"
12 }
```

Body Cookies Headers (11) Test Results Status: 201 Created Time: 48498 ms Size: 664 B

Pretty Raw Preview JSON

```
1 {
2   "Id": 1,
3   "Nombre": "Luis",
4   "Apellido": "Fernandez Medina",
5   "Direccion": "Calle Junin 1974",
6   "Telefono": "645478",
7   "FechaNacimiento": "1999-05-24T00:00:00",
8   "Genero": "M",
9   "NomPlan": "Verano",
10  "FechaInicio": "2019-05-02T00:00:00",
11  "FechaTermino": "2019-05-28T00:00:00"
12 }
```

– Comprobamos que ha sido agregado el nuevo registro en la tabla con GET.

GET http://localhost:52844/api/Cliente Send

Body Cookies Headers (10) Test Results Status: 200 OK Time: 1463 ms Size: 613 B

Pretty Raw Preview JSON

```
1 {
2   {
3     "Id": 1,
4     "Nombre": "Luis",
5     "Apellido": "Fernandez Medina",
6     "Direccion": "Calle Junin 1974",
7     "Telefono": "645478",
8     "FechaNacimiento": "1999-05-24T00:00:00",
9     "Genero": "M",
10    "NomPlan": "Verano",
11    "FechaInicio": "2019-05-02T00:00:00",
12    "FechaTermino": "2019-05-28T00:00:00"
13  }
14 }
```

- Del controlador Usuario, hacemos clic en POST Usuario para agregar un registro.

## Usuario

| API                                     | Description                 |
|---|-----------------------------|
| <a href="#">GET api/Usuario</a>         | No documentation available. |
| <a href="#">POST api/Usuario</a>        | No documentation available. |
| <a href="#">PUT api/Usuario/{id}</a>    | No documentation available. |
| <a href="#">DELETE api/Usuario/{id}</a> | No documentation available. |

Copiamos el formato de la cadena, y luego la pegamos en Postman, donde agregaremos un nuevo usuario.

application/json, text/json

Sample:

```
{
  "Id": 1,
  "Nombre": "sample string 2",
  "Apellido": "sample string 3",
  "Direccion": "sample string 4",
  "Telefono": "sample string 5",
  "FechaNacimiento": "2019-05-02T02:13:18.8851426-05:00",
  "Cargo": "sample string 7",
  "Clave": "sample string 8",
  "Turno": "sample string 9"
}
```

POST

http://localhost:52844/api/Usuario

Send

```

2  "Nombre": "Arlyn",
3  "Apellido": "Cotrado",
4  "Direccion": "calle junin",
5  "Telefono": "458421",
6  "FechaNacimiento": "1999-05-18",
7  "Cargo": "Administrador",
8  "Clave": "admi",
9  "Turno": "Tarde"
10 }
```

Body

Cookies

Headers (11)

Test Results

Status: 201 Created

Time: 980 ms

Size: 601 B

Pretty

Raw

Preview

JSON

```

1 {
2   "Id": 1,
3   "Nombre": "Arlyn",
4   "Apellido": "Cotrado",
5   "Direccion": "calle junin",
6   "Telefono": "458421",
7   "FechaNacimiento": "1999-05-18T00:00:00",
8   "Cargo": "Administrador",
9   "Clave": "admi",
10  "Turno": "Tarde"
11 }
```

- Del controlador Entrenador, hacemos clic en POST Entrenador para agregar un registro.

## Entrenador

| API  | Description                 |
|--|-----------------------------|
| <a href="#">GET api/Entrenador</a>         | No documentation available. |
| <a href="#">POST api/Entrenador</a>        | No documentation available. |
| <a href="#">PUT api/Entrenador/{id}</a>    | No documentation available. |
| <a href="#">DELETE api/Entrenador/{id}</a> | No documentation available. |

Copiamos el formato de la cadena, y luego la pegamos en Postman, donde agregaremos un nuevo entrenador.

application/json, text/json

Sample:

```
{
  "Id": 1,
  "Nombre": "sample string 2",
  "Apellido": "sample string 3",
  "Direccion": "sample string 4",
  "Telefono": "sample string 5",
  "FechaNacimiento": "2019-05-02T02:18:17.4416173-05:00",
  "CodSucursal": "sample string 7",
  "Estado": "sample string 8"
}
```

POST ▼
http://localhost:52844/api/Entrenador
Send ▼

```
1 {
2   "Nombre": "Fiorella",
3   "Apellido": "Juarez",
4   "Direccion": "Avenida Vigil 1487",
5   "Telefono": "487950",
6   "FechaNacimiento": "1997-08-04",
7   "CodSucursal": "2",
8   "Estado": "Activo"
9 }
```

Body   Cookies   Headers (11)   Test Results
Status: 201 Created   Time: 184 ms   Size: 598 B

Pretty   Raw   Preview
JSON ▼
≡

```
1 {
2   "Id": 1,
3   "Nombre": "Fiorella",
4   "Apellido": "Juarez",
5   "Direccion": "Avenida Vigil 1487",
6   "Telefono": "487950",
7   "FechaNacimiento": "1997-08-04T00:00:00",
8   "CodSucursal": "2",
9   "Estado": "Activo"
10 }
```

- Del controlador Plan, hacemos clic en POST Plan para agregar un registro.

| Plan   |                             |
|--|-----------------------------|
| API  | Description                 |
| <a href="#">GET api/Plan</a>                 | No documentation available. |
| <a href="#">POST api/Plan</a>                | No documentation available. |
| <a href="#">PUT api/Plan?IdPlan={IdPlan}</a> | No documentation available. |
| <a href="#">DELETE api/Plan/{id}</a>         | No documentation available. |

Copiamos el formato de la cadena, y luego la pegamos en Postman, donde agregaremos un nuevo plan.

application/json, text/json

Sample:

```
{
  "Id": 1,
  "Nombre": "sample string 2",
  "Costo": 3.1,
  "Duracion": 4,
  "Estado": "sample string 5"
}
```

The screenshot shows the Postman interface for a POST request to `http://localhost:52844/api/Plan`. The request body is set to `JSON (application/json)` and contains the following JSON:

```
{
  "Nombre": "Verano",
  "Costo": 30,
  "Duracion": 1,
  "Estado": "Inactivo"
}
```

The response status is `201 Created`, with a time of `724 ms` and a size of `478 B`. The response body is displayed in JSON format:

```
{
  "Id": 1,
  "Nombre": "Verano",
  "Costo": 30,
  "Duracion": 1,
  "Estado": "Inactivo"
}
```

- Del controlador Empresa, hacemos clic en POST Empresa para agregar un registro.

## Empresa

| API                                     | Description                 |
|---|-----------------------------|
| <a href="#">GET api/Empresa</a>         | No documentation available. |
| <a href="#">POST api/Empresa</a>        | No documentation available. |
| <a href="#">PUT api/Empresa/{id}</a>    | No documentation available. |
| <a href="#">DELETE api/Empresa/{id}</a> | No documentation available. |

Copiamos el formato de la cadena, y luego la pegamos en Postman, donde agregaremos un nuevo empresa.

application/json, text/json

Sample:

```
{
  "Id": 1,
  "Ruc": "sample string 2",
  "Nombre": "sample string 3",
  "Direccion": "sample string 4",
  "Telefono": "sample string 5"
}
```

POST http://localhost:52844/api/Empresa Send

none form-data x-www-form-urlencoded raw binary JSON (application/json)

```
1 {
2   "Ruc": "15116131111",
3   "Nombre": "Compañia",
4   "Direccion": "Avenida Bolognesi 144",
5   "Telefono": "487444"
6 }
```

Body Cookies Headers (11) Test Results Status: 201 Created Time: 104 ms Size: 519 B

Pretty Raw Preview JSON

```
1 {
2   "Id": 1,
3   "Ruc": "15116131111",
4   "Nombre": "Compañia",
5   "Direccion": "Avenida Bolognesi 144",
6   "Telefono": "487444"
7 }
```

## 4. REFERENCIAS

- [ 1 ] Torres, M. (2012). PROGRAMACION ORIENTADA A OBJETOS CON VISUAL BASIC 2012. Editorial Macro. Pág. 370[[1]
- [ 2 ] Entity Framework Tutorial. What is Entity Framework?. Recuperado de <https://www.entityframeworktutorial.net/what-is-entityframework.aspx>
- [ 3 ] Elman, J. y Lavin, M. (2014). Django ligero: utilizando REST, WebSockets y Backbone. Editor .o'Reilly Media, Inc. Pág. 61
- [ 4 ] Hugon, J. (2014). C# 5: desarrolle aplicaciones Windows con Visual Studio 2013. Ediciones ENI. Pág. 269