UNIVERSIDAD PRIVADA-DE-TACNA

INGENIERIA DE SISTEMAS

TITULO:

Trabajo Final de Unidad

CURSO:

BASE DE DATOS II

DOCENTE(ING):

Patrick Cuadros Quiroga

Integrantes:

Orlando Antonio Acosta Ortiz	(2015052775)
Orestes Ramirez Ticona	(2015053236)
Nilson Laura Atencio	(2015053846)
Roberto Zegarra Reyes	(2010036175)
Richard Cruz Escalante	(2013047247)
Wilfredo Vilca Chambilla	(2006028540)

Índice

1. PROBLEMATICA

El siguiente trabajo se desarrolla con el Sistema de Cajero Automatico, el cual busca que el cualquier cajero del Banco Pichincha pueda realizar acciones de forma mas automatizada y con toda seguridad:

- La Gestion de Cajero, Retiros, Depositos, Consultas
- Gestión de las usuarios que cuenten con una cuenta en el banco PICHINCHA.
- La venta de distintos tipos de planes.
- La realización de reportes sobre las transacciones realizadas del día, del mes, y sobre los clientes correspondientes a cada local.
- El sistema permitirá realizar la autentificación de los usuarios.

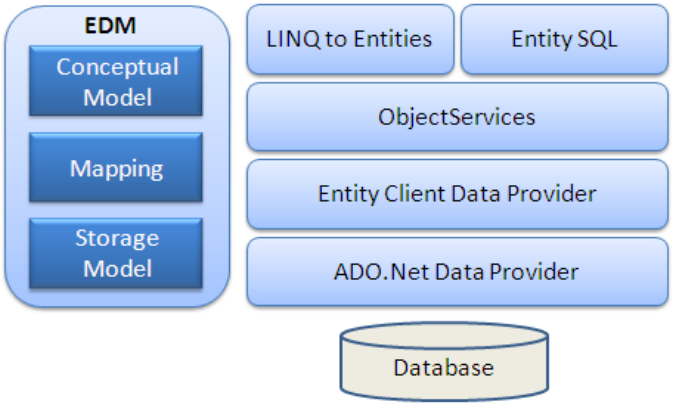
La motivación principal de este proyecto es que los cajeros del Banco Pichincha tengan una automatizacion mas eficaz, lo que le permitirá realizar sus tareas sin inconvenientes.

2. MARCO TEORICO

2.1. 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]

Su arquitectura:



2.2. 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.

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 tecnologias 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 probaar el codigo de una aplicación a lo largo de su desarrollo. Estas pruebas permiten asegurar que losmétodos 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

Parte 1: Consultando Datos

3.1. ANALISIS

3.2. DISEÑO

3.3. PRUEBAS

- Crear un proyecto de pruebas a nuestra solución.
- Dentro de la clase, agregamos el siguiente metodo ObtenerAlEstudianteConIDUno.

```
public void ObtenerAlEstudianteConIDUno()
{
    var ctx = new SchoolDBEntities();
    var student = ctx.Students.Find(1);
}
```

- Agregamos el siguiente metodo BuscarAlPrimerEstudianteConElNombreBill.

```
public void BuscarAlPrimerEstudianteConElNombreBill()
{
    using (var ctx = new SchoolDBEntities())
    {
        var student = (from s in ctx.Students
                       where s.StudentName == "Bill"
                       select s).FirstOrDefault<Student>();
    }
}
```

- Agregamos el siguiente metodo BuscarEstudiantesAgrupadosPorEstandar.
- Agregamos el siguiente metodo ObetenerListadoDeEstudiantesOrdenadosPorNombre.
- Agregamos el siguiente metodo BuscarTodosLostudiantesConElEstandarUno.

Parte 2: Guardando Datos

- Agregar al proyecto de pruebas una clase de pruebas TestUnit02, agregar el método InsertarEstudianteSatisfactoriamente, tomar como referencia el siguiente código.
- Agregar el método ActualizarElPrimerEstudianteSatisfactoriamente, tomar como referencia el siguiente código.

```
public void BuscarEstudiantesAgrupadosPorEstandar()
{
    using (var ctx = new SchoolDBEntities())
    {
        var students = ctx.Students.GroupBy(s => s.StandardId);
        foreach (var groupItem in students)
        {
            Console.WriteLine(groupItem.Key);
            foreach (var stud in groupItem)
                Console.WriteLine(stud.StudentId);
        }
    }

}
```

```
public void ObetenerListadoDeEstudiantesOrdenadosPorNombre()
{
    using (var ctx = new SchoolDBEntities())
    {
        var students = from s in ctx.Students
                       orderby s.StudentName ascending
                       select s;
    }
}
```

- Agregar el método EliminarElPrimerEstudianteSatisfactoriamente, tomar como referencia el siguiente código.
- Agregar el método: AgregarTresEstudiantesSatisfactoriamente, tomar como referencia el siguiente código:
- Finalmente agregar el método de pruebas: EliminarTresEstudiantesSatisfactoriamente, tomar como referencia el siguiente código:

```
public void BuscarTodosLostudiantesConElEstandarUno()
{
    using (var ctx = new SchoolDBEntities())
    {
    var anonymousObjResult = from s in ctx.Students
    where s.StandardId == 1
    select new {
    Id = s.StudentId,
    Name = s.StudentName
        };
    }
}
```

```
0 referencias
public void TInsertarEstudianteSatisfactoriamente()
{
    using (var context = new SchoolDBEntities())
    {
        var std = new Student()
        {
            FirstName = "Bill",
            LastName = "Gates"
        };
        context.Students.Add(std);
        context.SaveChanges();
    }
```

```
0 referencias
public void ActualizarElPrimerEstudianteSatisfactoriamente()
{
    using (var context = new SchoolDBEntities())
    {
        var std = context.Students.First<Student>();
        std.FirstName = "Steve";
        context.SaveChanges();
    }
}
0 referencias
```

```
public void EliminarElPrimerEstudianteSatisfactoriamente()
{
    using (var context = new SchoolDBEntities())
    {
        var std = context.Students.First<Student>();
        context.Students.Remove(std);
        context.SaveChanges();
    }
}
```

```
0 referencias
public void AgregarTresEstudiantesSatisfactoriamente()
{
    IList<Student> newStudents = new List<Student>();
    {
        new Student() { StudentName = "Steve" };
        new Student() { StudentName = "Bill" };
        new Student() { StudentName = "James" };
    };
    using (var context = new SchoolDBEntities())
    {
        context.Students.AddRange(newStudents);
        context.SaveChanges();
    }
}
```

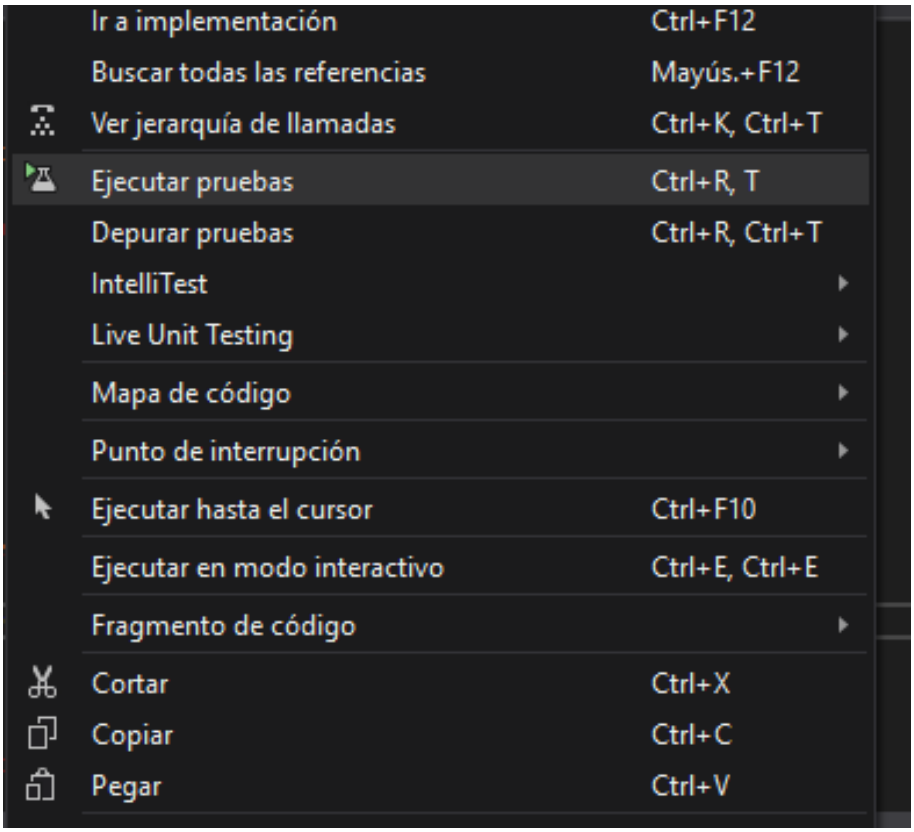
```
public void EliminarTresEstudiantesSatisfactoriamente()
{
    IList<Student> studentsToRemove = new List<Student>();
    {
        new Student() { StudentID = 1, StudentName = "Steve" };
        new Student() { StudentID = 2, StudentName = "Bill" };
        new Student() { StudentID = 3, StudentName = "James" };
    };

using (var context = new SchoolDBEntities())
    {
     context.Students.RemoveRange(studentsToRemove);
    context.SaveChanges();
    }
```

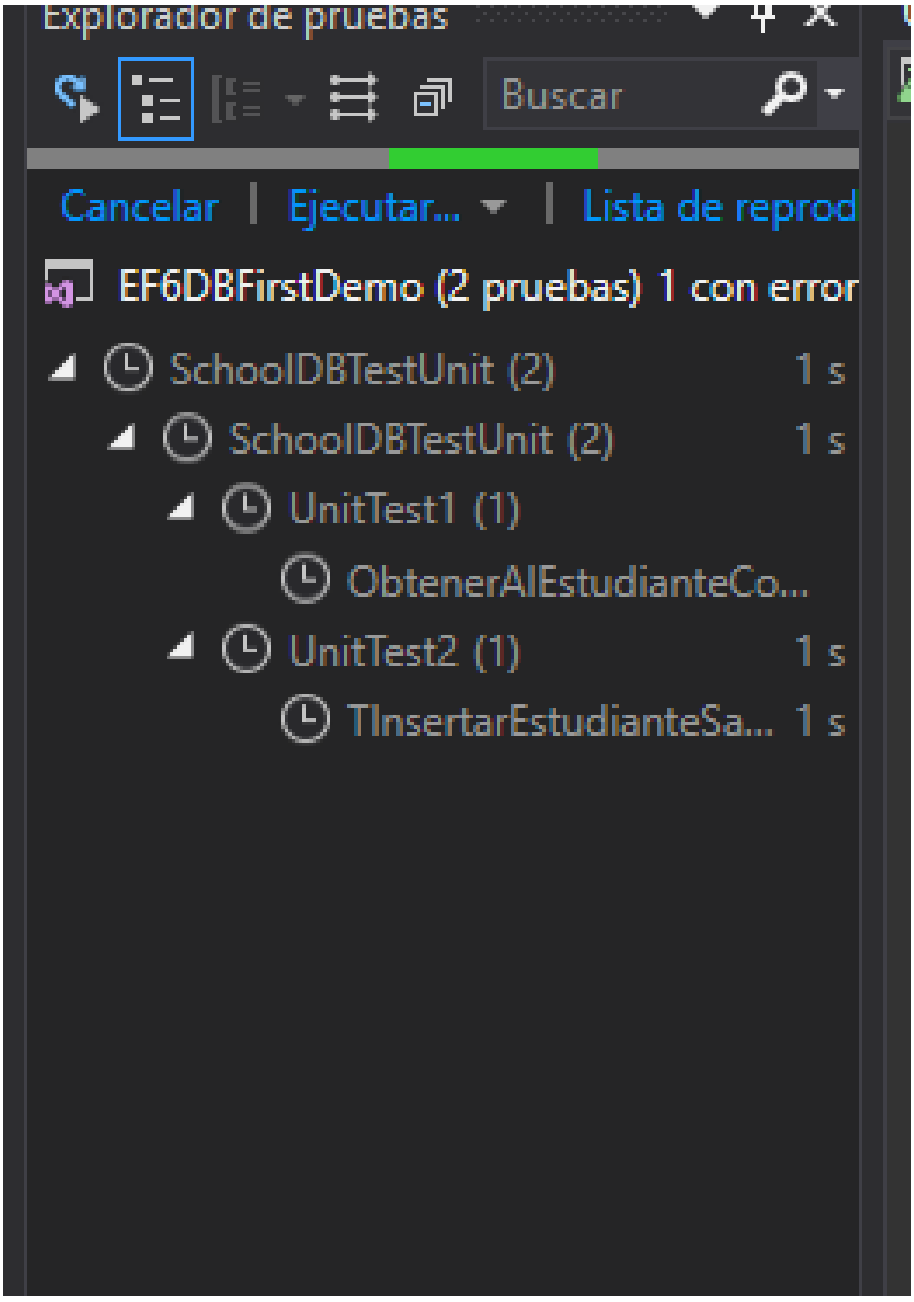
4. ANALISIS E INTERPRETACION DE RESULTADOS

-Al compilar el proyecto de prueba, las pruebas aparecen en el Explorador de pruebas. Si el Explorador de pruebas no está visible, elija Prueba en el menú de Visual Studio, elija Ventanas y, después, Explorador de pruebas. -Se puede elegir Ejecutar todas para ejecutar todas las pruebas o bien Ejecutar para elegir un subconjunto de pruebas que se desea ejecutar. Después de ejecutar un conjunto de pruebas, aparecerá un resumen de la serie de pruebas en la parte inferior de la ventana Explorador de pruebas. Seleccione una prueba para ver los detalles de esa prueba en el panel inferior.

– Al compilar el proyecto de prueba.



– Explorador de pruebas.


Explorador de pruebas
Buscar
Cancelar | Ejecutar... | Lista de reprod
EF6DBFirstDemo (2 pruebas) 1 con error
SchoolDBTestUnit (2) 1 s
SchoolDBTestUnit (2) 1 s
UnitTest1 (1)
ObtenerAlEstudianteCo...
UnitTest2 (1) 1 s
TInsertarEstudianteSa... 1 s

5. CONCLUSIONES

Las pruebas unitarias nos pueden ayudar a testear los detalles de nuestro programa son el primer paso en la realización de pruebas de un software, siendo el paso más importante en remover errores y defectos. Existen muchas herramientas que ayudan al equipo de testing de un proyecto. Elegir las más adecuadas no es una tarea sencilla. A través de las herramientas de testing uno puede automatizar y optimizar las pruebas de los programas, evitando pérdidas de tiempo significativas. Con la estandarización y el uso de buenas prácticas (ej.: documentación precisa), las pruebas unitarias pueden ser muy efectivas, resultando en la reducción de esfuerzo y el incremento de la calidad de los productos.

6. REFERENCIAS

[1] Torres, M. (2012). PROGRAMACION ORIENTADA A OBJETOS CON VISUAL BASIC 2012. Editorial Macro. Pág. 370[[1]

[2] Elman, J. y Lavin, M. (2014). Django ligero: utilizando REST, WebSockets y Backbone. Editor .ºReilly Media, Inc. Pág. 61

[3] Hugon, J. (2014). C# 5: desarrolle aplicaciones Windows con Visual Studio 2013. Ediciones ENI. Pág. 269