

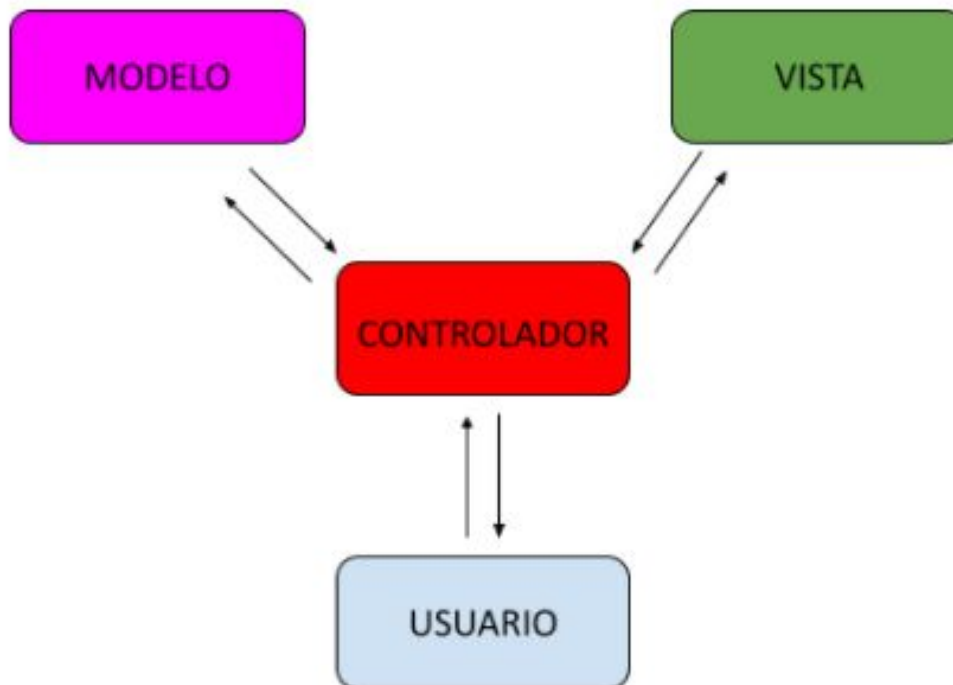


MVC

Model View Controller - Continuación

Fundamentos del Modelo Vista Controlador MVC

Modelo Vista Controlador (MVC) es un estilo de arquitectura de software que separa los datos de una aplicación, la interfaz de usuario, y la lógica de control en tres componentes distintos.

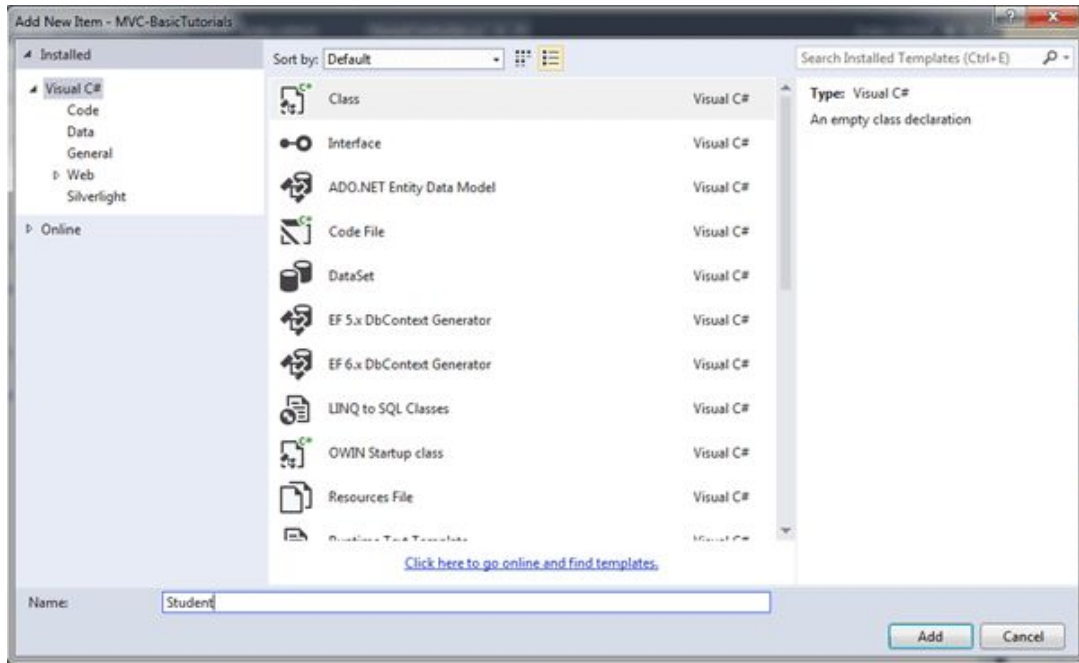


- El **Modelo** contiene una representación de los datos que maneja el sistema, su lógica de negocio, y sus mecanismos de persistencia.
- La **Vista**, o interfaz de usuario, compone la información que se envía al cliente y los mecanismos de interacción con éste.
- El **Controlador**, actúa como intermediario entre el Modelo y la Vista, gestionando el flujo de información entre ellos y las transformaciones para adaptar los datos a las necesidades de cada uno.

Repaso del modelo

- Acceder a la capa de almacenamiento de datos. Lo ideal es que el modelo sea independiente del sistema de almacenamiento.
- Define las reglas de negocio (la funcionalidad del sistema). Un ejemplo de regla puede ser: "Si la mercancía pedida no está en el almacén, consultar el tiempo de entrega estándar del proveedor".

Crear clases de modelo



- Las clases de modelo representan datos específicos de dominio y lógica empresarial en la aplicación MVC.
- Representa la forma de los datos como propiedades públicas y la lógica empresarial como métodos.
- En la aplicación ASP.NET MVC, todas las clases del modelo deben crearse en la carpeta del modelo.

Nomenclatura sugerida para los modelos (clases)

Elección de palabras

- ✓ UTILICE nombres fácil de leer e interpretar.
- ✗ NO use caracteres de subrayado, guiones ni ningún otro carácter no alfanumérico.
- ✗ EVITE el uso de identificadores que entren en conflicto con palabras clave de lenguajes de programación ampliamente utilizados.
- ✓ UTILICE nombres semánticamente interesantes en lugar de palabras clave específicas del lenguaje para los nombres de tipo. Ej. `GetLength` es mejor que `GetInt`
- ✗ Evitar nombres específicos del lenguaje

Uso de abreviaturas y acrónimos

- ✗ NO use abreviaturas ni contracciones como parte de los nombres de identificador.
- ✗ NO use acrónimos que no se acepten ampliamente, e incluso si lo están, solo cuando sea necesario.

Nombres

- ✓ UTILICE la notación Pascal para nombrar espacios de nombres, clases, métodos, estructuras, enumeraciones y propiedades públicas.
- ✓ UTILICE la notación Camel para nombrar campos, variables.

Archivos

- ✓ UTILICE un archivo por clase.
- ✓ ESPECIFIQUE el modificador de acceso de forma explícita.

DEMO

Ejemplo: Clase Persona

1- Hacer click derecho en la carpeta Models , luego seleccionamos el menú Agregar y luego seleccionamos la opción Clase.

2- Editar la clase Persona como sigue:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;

namespace WebApplication1.Models
{
    public class Persona
    {
        private int id;
        private string nombre;
        private string apellido;
        private string domicilio;
        private int id_localidad;

        public Persona()
        {
        }
    }
}
```

```
public Persona(int id, string nombre, string apellido, string domicilio, int
id_localidad)
{
    Nombre = nombre;
    Id = id;
    Apellido = apellido;
    Domicilio = domicilio;
    Id_localidad = id_localidad;
}

public string Nombre { get => nombre; set => nombre = value; }
public int Id { get => id; set => id = value; }
public string Apellido { get => apellido; set => apellido = value; }
public string Domicilio { get => domicilio; set => domicilio = value; }
public int Id_localidad { get => id_localidad; set => id_localidad = value; }
}
}
```

¿Con esto respondemos a las funciones
definidas previamente para el modelo?

Para debatir en clase...

Crear la conexión a la base de datos

En el archivo **web.config** creamos la configuración necesaria para luego, conectarnos a la base de datos:

Sintaxis:

```
<connectionStrings>  
  <add name="nombreDeLaConexion"  
connectionString="Server=nombreDelServidorSQL;Database=nombreDeLaBase  
DeDatos;User Id=nombreUsuario;Password=passwordUsuario;" />  
</connectionStrings>
```

Ejemplo:

```
<connectionStrings>  
  <add name="BDLocal" connectionString="Server=localhost\SQLEXPRESS;Database=db_example;User  
Id=sa;Password=1234;" />  
</connectionStrings>
```

Manipular los datos

Para manipular los datos, creamos una clase Gestor.

Si bien puedes definir la gestión o manipulación de los datos en la misma clase modelo, es una buena práctica mantener la independencia con el sistema de base de datos.

DEMO

Crear clase GestorPersona

Ejemplo: Gestor Persona

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Data.SqlClient;
using System.Configuration;

namespace WebApplication1.Models
{
    public class GestorPersona
    {
        public void AgregarPersona(Persona persona)
        {
            string StrConn = ConfigurationManager.ConnectionStrings["BDLocal"].ToString();

            using (SqlConnection conn = new SqlConnection(StrConn))
            {
                conn.Open();

                SqlCommand comm = conn.CreateCommand();
                comm.CommandText = "insertar_persona";
                comm.CommandType = CommandType.StoredProcedure;
                comm.Parameters.Add(new SqlParameter("@nombre", persona.Nombre));
                comm.Parameters.Add(new SqlParameter("@apellido", persona.Apellido));
                comm.Parameters.Add(new SqlParameter("@domicilio", persona.Domicilio));
                comm.Parameters.Add(new SqlParameter("@id_localidad", persona.Id_localidad));
                comm.ExecuteNonQuery();
            }
        }
    }
}
```

```
public void ModificarPersona(Persona persona)
{
    string StrConn = ConfigurationManager.ConnectionStrings["BDLocal"].ToString();

    using (SqlConnection conn = new SqlConnection(StrConn))
    {
        conn.Open();

        SqlCommand comm = conn.CreateCommand();
        comm.CommandText = "modificar_persona";
        comm.CommandType = CommandType.StoredProcedure;
        comm.Parameters.Add(new SqlParameter("@nombre", persona.Nombre));
        comm.Parameters.Add(new SqlParameter("@apellido", persona.Apellido));
        comm.Parameters.Add(new SqlParameter("@domicilio", persona.Domicilio));
        comm.Parameters.Add(new SqlParameter("@id_localidad", persona.Id_localidad));
        comm.Parameters.Add(new SqlParameter("@id", persona.Id));

        comm.ExecuteNonQuery();
    }
}
```

Ejemplo: Gestor Persona

```
public void Eliminar(int id)
{
    string StrConn =
    ConfigurationManager.ConnectionStrings["BDLocal"].ToString();

    using (SqlConnection conn = new SqlConnection(StrConn))
    {
        conn.Open();

        SqlCommand comm = new SqlCommand("eliminar_persona", conn);
        comm.CommandType = System.Data.CommandType.StoredProcedure;
        comm.Parameters.Add(new SqlParameter("@id", id));

        comm.ExecuteNonQuery();
    }
}
```

```
public Persona ObtenerPorId(int id)
{
    Persona persona = null;
    string StrConn = ConfigurationManager.ConnectionStrings["BDLocal"].ToString();

    using (SqlConnection conn = new SqlConnection(StrConn))
    {
        conn.Open();

        SqlCommand comm = conn.CreateCommand();
        comm.CommandText = "obtener_persona";
        comm.CommandType = CommandType.StoredProcedure;
        comm.Parameters.Add(new SqlParameter("@id", id));

        SqlDataReader dr = comm.ExecuteReader();

        if (dr.Read())
        {
            string nombre = dr.GetString(1).Trim();
            string apellido = dr.GetString(2).Trim();
            string domicilio = dr.GetString(3).Trim();
            int id_localidad = dr.GetInt32(4);

            persona = new Persona(id, nombre, apellido, domicilio, id_localidad);
        }

        dr.Close();
    }

    return persona;
}
```

Ejemplo: Gestor Persona

```
public List<Persona> ObtenerPersonas()
{
    List<Persona> lista = new List<Persona>();
    string StrConn = ConfigurationManager.ConnectionStrings["BDLocal"].ToString();

    using (SqlConnection conn = new SqlConnection(StrConn))
    {
        conn.Open();

        SqlCommand comm = conn.CreateCommand();
        comm.CommandText = "obtener_personas";
        comm.CommandType = CommandType.StoredProcedure;

        SqlDataReader dr = comm.ExecuteReader();
        while (dr.Read())
        {
            int id = dr.GetInt32(0);
            string nombre = dr.GetString(1).Trim();
            string apellido = dr.GetString(2).Trim();
            string domicilio = dr.GetString(3).Trim();
            int id_localidad = dr.GetInt32(4);

            Persona persona = new Persona(id, nombre, apellido, domicilio, id_localidad);
            lista.Add(persona);
        }

        dr.Close();
    }

    return lista;
}
```

Repaso de Controller

1. El controlador maneja las solicitudes de URL entrantes. El enrutamiento MVC envía solicitudes al controlador apropiado y al action method según la URL y las rutas configuradas.
2. Todos los métodos públicos de la clase Controller se denominan métodos de acción o action methods.
3. El nombre de la clase de controlador debe terminar con "Controller".

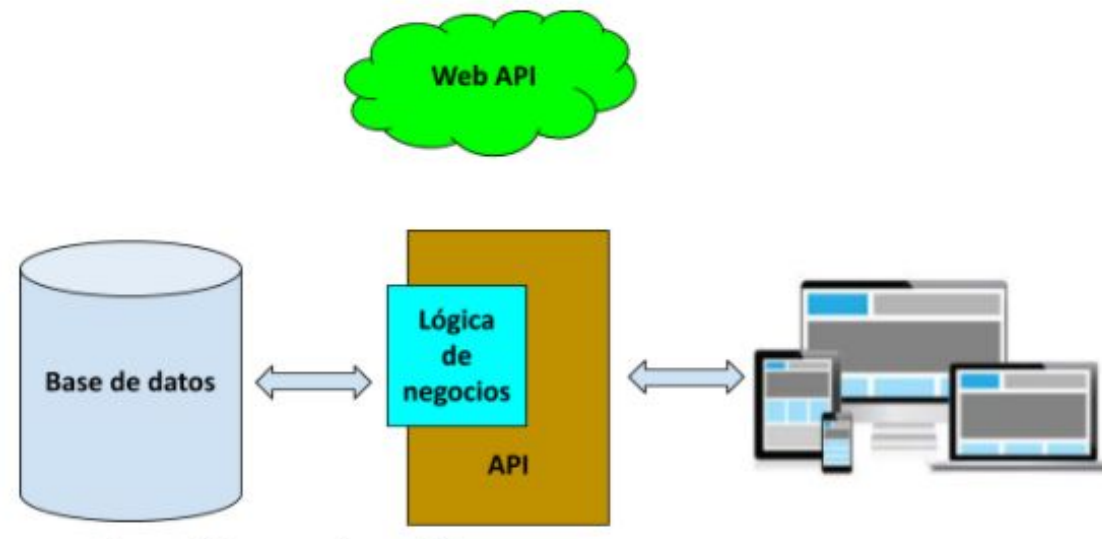
Nosotros, crearemos un API Rest (que será nuestro controller)

¿Qué es una API?

Para debatir en clase...

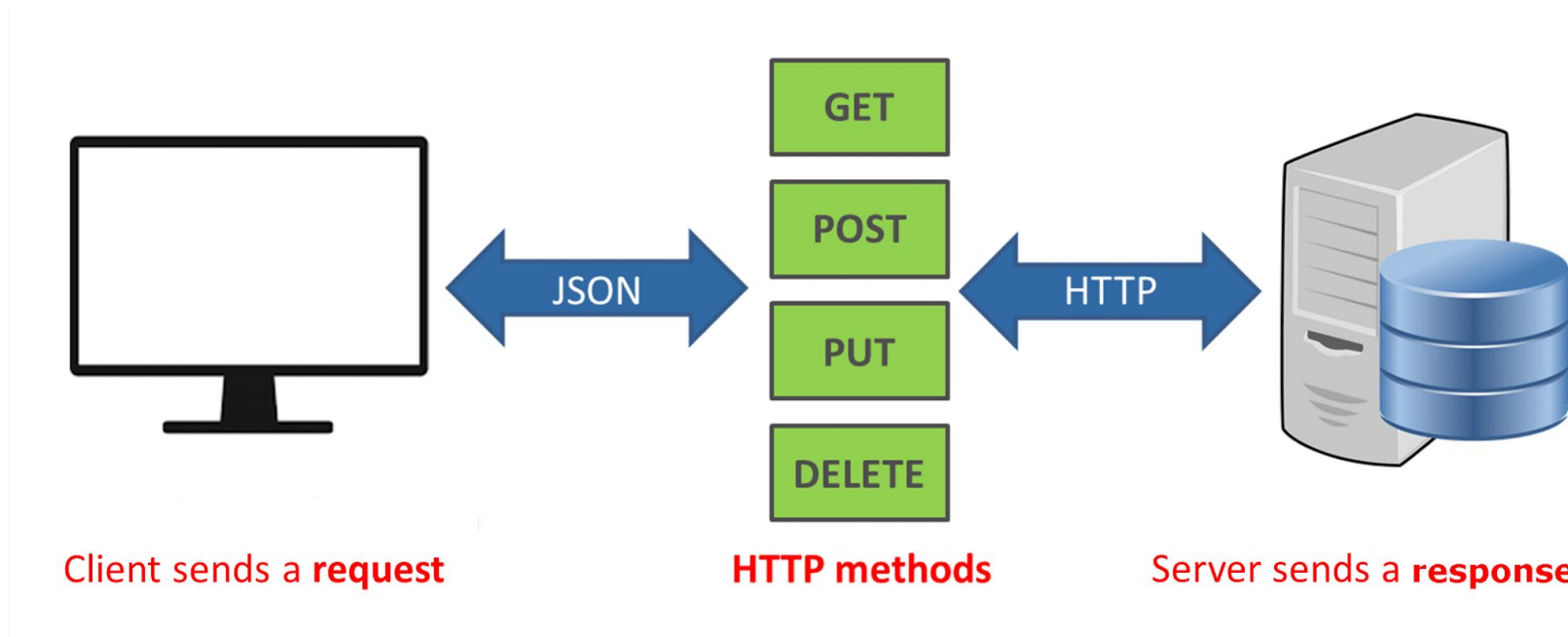
API

Una API (Application Programming Interface) es un conjunto de algoritmos que brindan acceso a distintas funcionalidades de un software. Son utilizadas para brindar información a sistemas informáticos externos al software que tiene la conexión y el acceso a los datos. Se considera como el contrato entre el proveedor de los datos y el usuario.



API Rest

Un servicio REST no es una arquitectura de software, sino que es una interfaz entre sistemas que utilicen el protocolo HTTP para realizar el intercambio de datos en formatos XML y JSON.



Métodos HTTP

A continuación se enumeran los métodos de petición http más utilizados:

- **GET**, permite recuperar recursos del servidor (datos). Si la respuesta es positiva (200 OK), el método GET devuelve la representación del recurso en un formato concreto: HTML, XML, JSON u otro. De lo contrario, si la respuesta es negativa, devuelve 404 (not found) o 400 (bad request).
- **POST**, permite crear o ejecutar acciones sobre recursos del servidor. Generalmente se utiliza este método cuando se envían datos de un formulario al servidor. Si la respuesta es positiva, el método POST devuelve 201 (created).
- **PUT**, permite modificar recursos del servidor (aunque permite también crear). Si la respuesta es positiva, el método PUT devuelve 201(created) o 204 (no response).
- **DELETE**, permite eliminar recursos del servidor. Si la respuesta es positiva, el método DELETE devuelve 200 junto con un body response, o 204 sin body.

JSON

JSON (JavaScript Object Notation) es un formato de intercambio de datos de sencillo entendimiento ya que es posible leer, escribir y analizar los objetos sin mayores complicaciones.

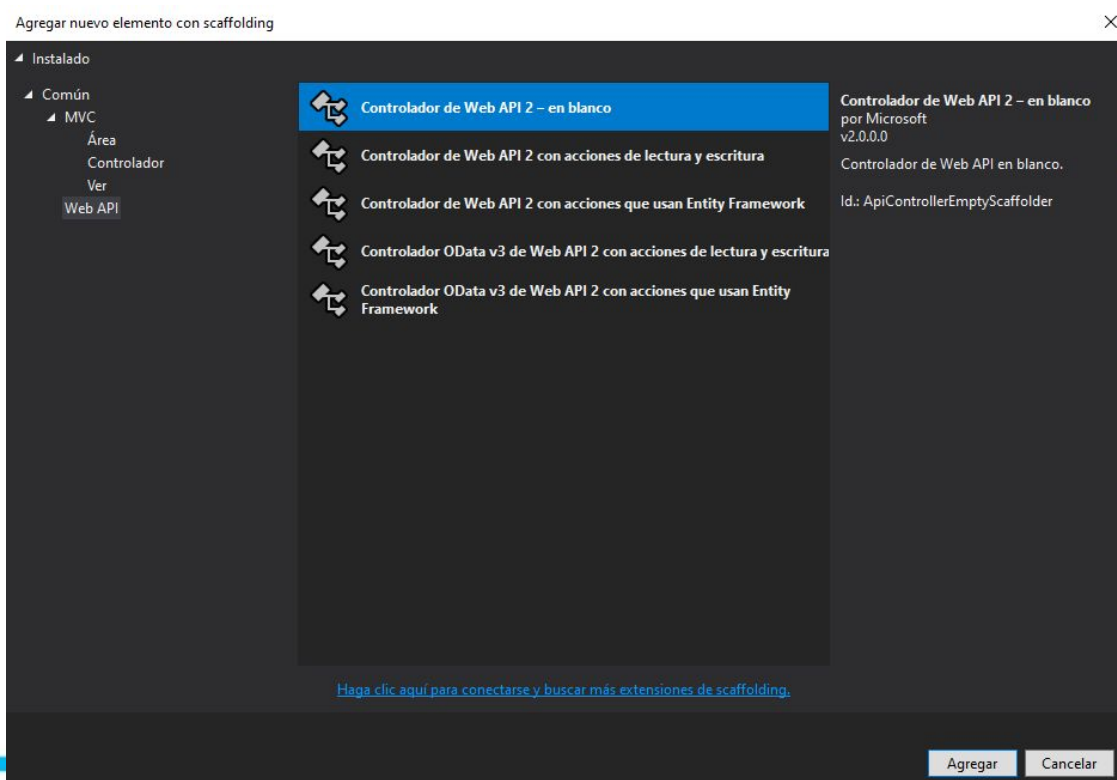
```
{  
  "results": [  
    {  
      "id": 1,  
      "name": "Renegade Internet",  
      "username": "renegade",  
      "status": true,  
      "recycle": false,  
      "notes": "",  
      "information": {  
        "company": "Renegade Internet",  
        "name": "Mike Cherichetti",  
        "title": "",  
        "email": "mike@renegadeinternet.com",  
        "website": "http://www.renegadeinternet.com/"  
      }  
    }  
  ]  
}
```

 JSON
JavaScript Object Notation

Crear un API Rest (controlador)

1- Hacer click derecho en la carpeta Controllers, luego seleccionamos el menú Agregar y luego seleccionamos la opción Controlador....

2- Luego, seleccionar el menú Web API y luego la opción Controlador de Web API 2 - con acciones de escritura y lectura o, en blanco y hacemos click en Agregar.



El controlador creado será el encargado de comunicarse con nuestro modelo para realizar las operaciones CRUD en la base de datos.

Crear un API Rest (controlador)

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Net;
using System.Net.Http;
using System.Web.Http;
using PILWebAPI.Models;

namespace PILWebAPI.Controllers
{
    public class PersonaController : ApiController
    {
        // GET: api/Persona
        public IEnumerable<Persona> Get()
        {
            GestorPersona gPersona = new GestorPersona();
            return gPersona.ObtenerPersonas();
        }

        // GET: api/Persona/5
        public Persona Get(int id)
        {
            GestorPersona gPersona = new GestorPersona();
            return gPersona.ObtenerPorId(id);
        }
    }
}
```

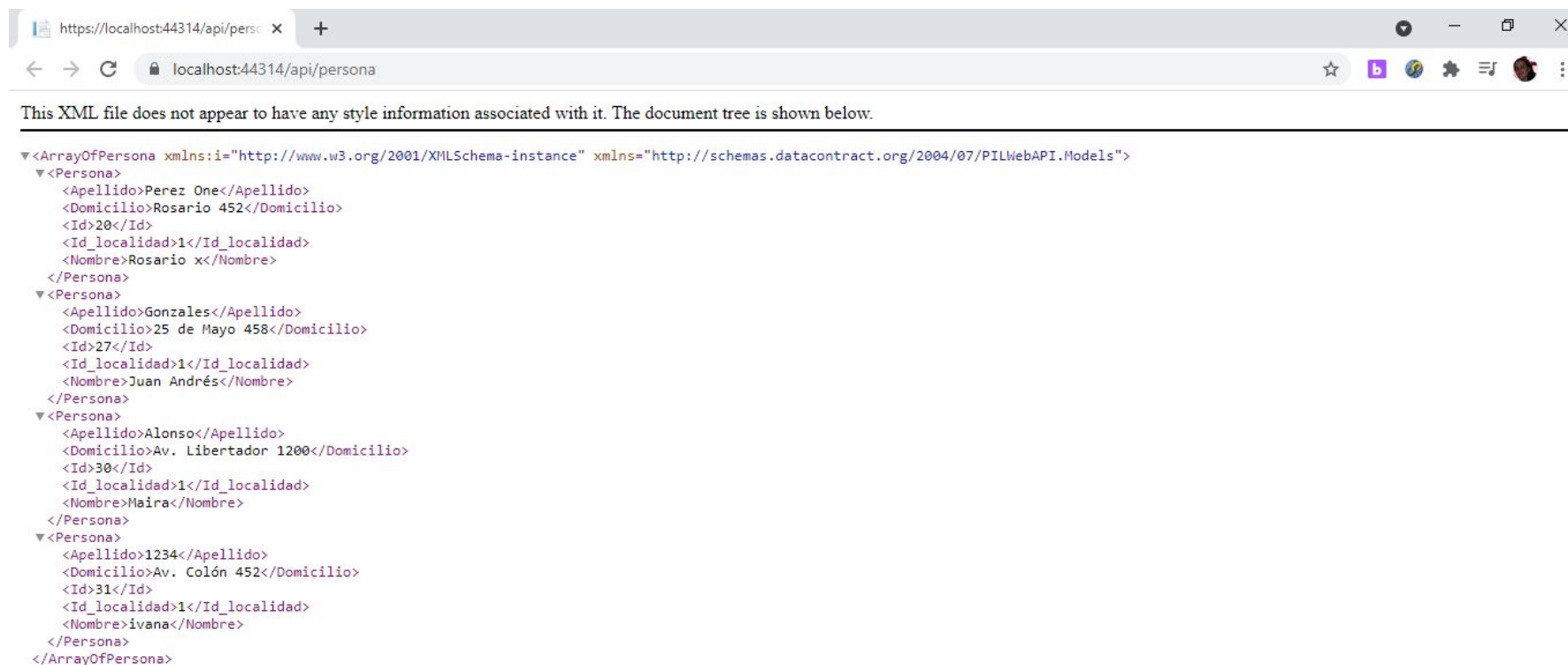
```
// POST: api/Persona
public void Post([FromBody]Persona value)
{
    GestorPersona gPersona = new GestorPersona();
    gPersona.AgregarPersona(value);
}

// PUT: api/Persona/5
public void Put(int id, [FromBody]string value)
{
}

// DELETE: api/Persona/5
public void Delete(int id)
{
    GestorPersona gPersona = new GestorPersona();
    gPersona.Eliminar(id);
}
}
```

Crear un API Rest (controlador)

Finalmente, puedes evaluar tu API Rest, ejecutando la aplicación y escribiendo la ruta a la api como sigue:

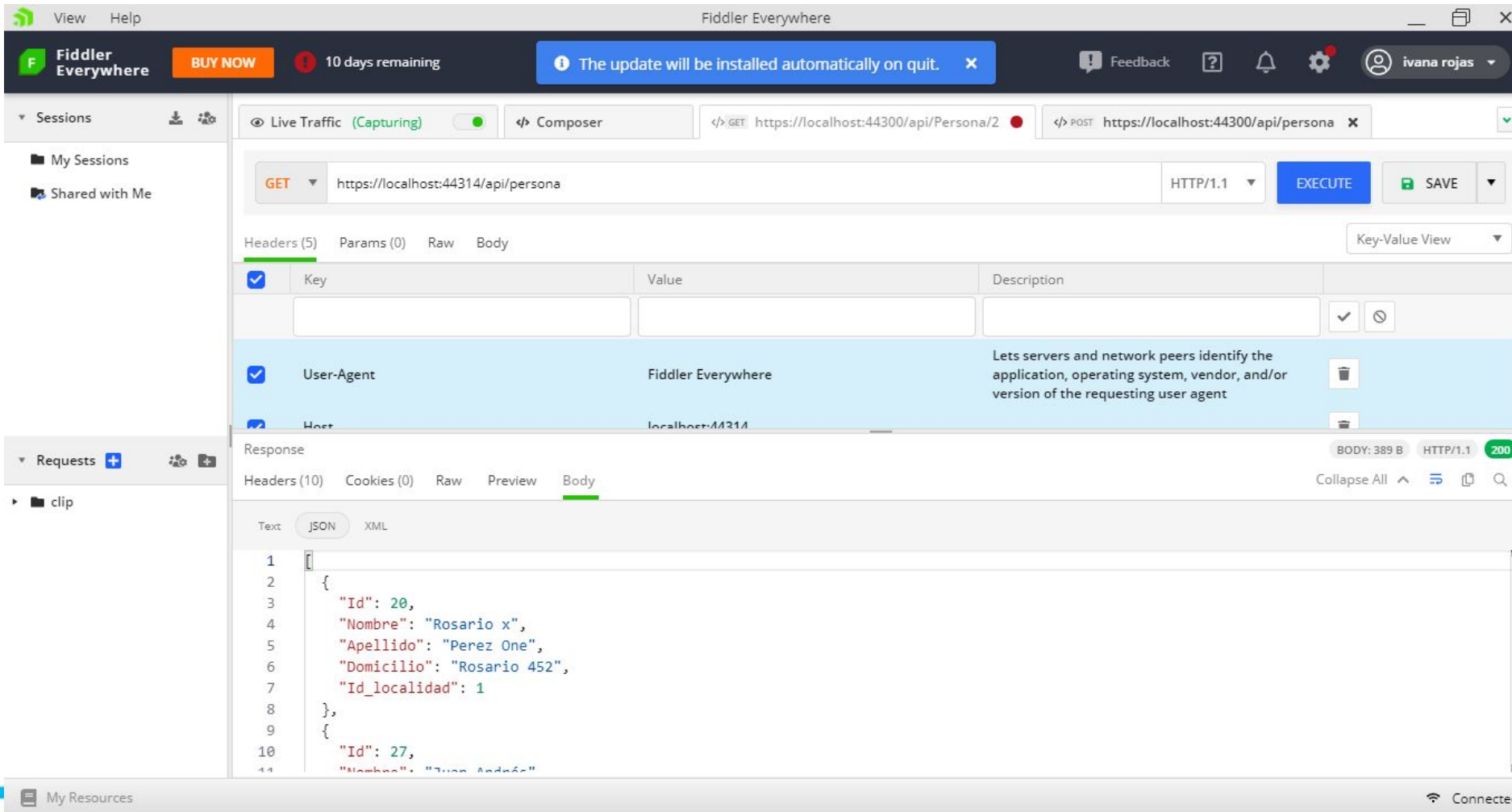


The screenshot shows a web browser window with the address bar displaying `https://localhost:44314/api/persona`. The page content indicates that the XML file does not have any style information associated with it. The XML document tree is displayed below, showing an array of four person objects. Each object contains fields for Apellido, Domicilio, Id, Id_localidad, and Nombre.

```
<ArrayOfPersona xmlns:i="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://schemas.datacontract.org/2004/07/PILWebAPI.Models">
  <Persona>
    <Apellido>Perez One</Apellido>
    <Domicilio>Rosario 452</Domicilio>
    <Id>20</Id>
    <Id_localidad>1</Id_localidad>
    <Nombre>Rosario x</Nombre>
  </Persona>
  <Persona>
    <Apellido>Gonzales</Apellido>
    <Domicilio>25 de Mayo 458</Domicilio>
    <Id>27</Id>
    <Id_localidad>1</Id_localidad>
    <Nombre>Juan Andrés</Nombre>
  </Persona>
  <Persona>
    <Apellido>Alonso</Apellido>
    <Domicilio>Av. Libertador 1200</Domicilio>
    <Id>30</Id>
    <Id_localidad>1</Id_localidad>
    <Nombre>Maira</Nombre>
  </Persona>
  <Persona>
    <Apellido>1234</Apellido>
    <Domicilio>Av. Colón 452</Domicilio>
    <Id>31</Id>
    <Id_localidad>1</Id_localidad>
    <Nombre>ivana</Nombre>
  </Persona>
</ArrayOfPersona>
```


Crear un API Rest (controlador)

También puedes evaluar tu API Rest utilizando Postman o Fiddler.



The screenshot displays the Fiddler Everywhere application interface. The top bar includes the Fiddler Everywhere logo, a "BUY NOW" button, a "10 days remaining" timer, and a notification that the update will be installed automatically on quit. The main interface is divided into several sections:

- Sessions:** A sidebar on the left showing "My Sessions" and "Shared with Me".
- Composer:** The central area for crafting HTTP requests. It shows a GET request to `https://localhost:44314/api/persona` with a status of "Capturing".
- Headers:** A table below the URL bar showing headers for the selected request. The headers are:

Key	Value	Description
✓		
✓	User-Agent	Fiddler Everywhere
✓	Host	localhost:44314

The description for the User-Agent header is: "Lets servers and network peers identify the application, operating system, vendor, and/or version of the requesting user agent".

- Response:** The bottom section shows the response to the request. It is a JSON object with the following structure:

```
1 {
2   {
3     "Id": 20,
4     "Nombre": "Rosario x",
5     "Apellido": "Perez One",
6     "Domicilio": "Rosario 452",
7     "Id_localidad": 1
8   },
9   {
10    "Id": 27,
11    "Nombre": "Juan Andres"
```

The response status is 200 (OK) and the body size is 389 B. The bottom status bar indicates "Connected".



REFERENCIAS

<https://www.tutorialsteacher.com/mvc/asp.net-mvc-tutorials>

<https://si.ua.es/es/documentacion/asp-net-mvc-3/>