



# PRACTICA 07 – E-LEARNING

Construir un cliente de un Servicio RESTful creado conASP.Net Web API.

DESARROLLO RÁPIDO DE APLICACIONES

Por Juan José Camacho Hidalgo  
UNIVERSIDAD DE ALMERÍA

El objetivo de esta solución es la construcción de un cliente para el servicio RESTful creado anteriormente con ASP.Net Web API. En este caso con WPF.

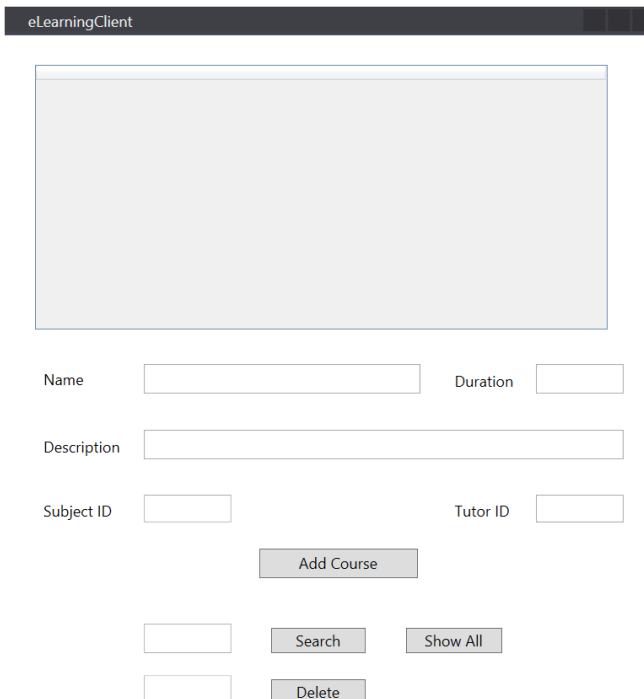
¿Qué es el cliente? Un cliente es una conexión a un servicio o servidor, y con el que interactúa. (es decir, solicita un servicio). Un servidor es aquel que proporciona el servicio. Por tanto puede tener más de un cliente. Cada cliente actúa como una instancia independiente, aunque puede haber interacciones entre varios clientes si la lógica de la aplicación lo permite.

Para llamar a una WebAPI, se utiliza librerías HttpClient. Para ello, vamos a estudiar la solución eLearningWebAPIClientWpfApp, donde se implementa ya un cliente.

Action	HTTP method	Relative URI
Get a course by ID	GET	/api/courses/id
Create a new course	POST	/api/courses
Update a course	PUT	/api/courses/id
Delete a course	DELETE	/api/courses/id

Algo que me parece importante mencionar, antes de continuar, son las acciones de las que dispondrá un cliente: obtener un curso ordenandolos por ID, crear un nuevo curso, actualizar un curso y borrar un curso. Todas ellas corresponden a los métodos GET, POST, PUT y DELETE.

En primer lugar, mostramos la vista:



Esta es la vista de un cliente donde puede interactuar con los cursos.

En cuanto a los modelos, tenemos el modelo de cursos, CourseModel, el de asignaturas, SubjectModel y el de TutorModel, cada uno con sus respectivos atributos.

Además, se define una enumeración, que consiste en saber si es de género masculino o femenino. (clase Gender)

```

namespace eLearningWebAPIClientWpfApp.Enums
{
    1 referencia | Camacho Hidalgo, Juan Jose, Hace 3 horas | 1 autor, 1 cambio
    public enum Gender
    {
        Male = 0,
        Female = 1
    }
}

```

En cuanto a la lógica de la aplicación, los métodos de los que hace uso un cliente son:

```

2 referencias | Camacho Hidalgo, Juan Jose, Hace 3 horas | 1 autor, 1 cambio
public partial class MainWindow : Window
{
    0 referencias | Camacho Hidalgo, Juan Jose, Hace 3 horas | 1 autor, 1 cambio
    public MainWindow()
    {
        InitializeComponent();
        GetData();
    }

    4 referencias | Camacho Hidalgo, Juan Jose, Hace 3 horas | 1 autor, 1 cambio
    private void GetData()...

    1 referencia | Camacho Hidalgo, Juan Jose, Hace 3 horas | 1 autor, 1 cambio
    private void addCourseButton_Click(object sender, RoutedEventArgs e)...

    1 referencia | Camacho Hidalgo, Juan Jose, Hace 3 horas | 1 autor, 1 cambio
    private void searchButton_Click(object sender, RoutedEventArgs e)...

    1 referencia | Camacho Hidalgo, Juan Jose, Hace 3 horas | 1 autor, 1 cambio
    private void showAllButton_Click(object sender, RoutedEventArgs e)...

    1 referencia | Camacho Hidalgo, Juan Jose, Hace 3 horas | 1 autor, 1 cambio
    private void deleteButton_Click(object sender, RoutedEventArgs e)...
}

```

Como podemos observar, GetData(), que básicamente tiene como objetivo mostrar los datos en la aplicación, además de crear el cliente, y los otros métodos son para añadir curso, buscar, mostrar todo y eliminar, de los que hereda cada Button de la interfaz gráfica. Así se define una aplicación cliente.

```

4 referencias | Camacho Hidalgo, Juan Jose, Hace 3 horas | 1 autor, 1 cambio
private void GetData()
{
    HttpClient webClient = new HttpClient();
    webClient.BaseAddress = new Uri("http://localhost:55853/");

    webClient.DefaultRequestHeaders.Accept.Clear();
    webClient.DefaultRequestHeaders.Accept.Add(
        new MediaTypeWithQualityHeaderValue("application/json"));

    HttpResponseMessage response = webClient.GetAsync("api/courses").Result;

    if (response.IsSuccessStatusCode)
    {
        //var resultString = await response.Content.ReadAsStringAsync();
        //MessageBox.Show(resultString);
        //var courses = JsonConvert.DeserializeObject<IEnumerable<CourseModel>>(resultString);
        //courseDataGrid.ItemsSource = courses;
    }
}

```

Más en ampliación observamos como GetData() hace uso de HttpClient para configurar un cliente del servicio, y a donde lo tiene que llevar en primer lugar.

Para cada método que realiza MainWindow, tiene que crear una instancia de cliente, y según realice, modificarle el BaseAddress, así como realizar la petición. Si algo no va correctamente, gestionaremos los errores mediante MessageBox.Show.

Como continuación de la solución al cliente, se define un proyecto con el patrón MVVM, llamado eLearningWebAPIClientMVVMWpfApp.

Como siempre en este patrón, MainWindow, la vista, no implementa los métodos (comportamiento) de la interfaz gráfica, si no que esto le corresponde al modelo de la vista o modelos de la vista.

```
namespace eLearningWebAPIClientMVVMWpfApp
{
    /// <summary>
    /// Lógica de interacción para MainWindow.xaml
    /// </summary>
    2 referencias | Camacho Hidalgo, Juan Jose, Hace 4 horas | 1 autor, 1 cambio
    public partial class MainWindow : Window
    {
        0 referencias | Camacho Hidalgo, Juan Jose, Hace 4 horas | 1 autor, 1 cambio
        public MainWindow()
        {
            InitializeComponent();
        }
    }
}
```

Dispone una clase Data, donde se implementan todos los métodos necesarios en la vista, mediante la interfaz ICommand, es decir, comandos que interaccionan con el modelo, en base a la vista.

```
0 referencias | Camacho Hidalgo, Juan Jose, Hace 4 horas | 1 autor, 1 cambio
public String Panel1StatusBar { get; set; }
0 referencias | Camacho Hidalgo, Juan Jose, Hace 4 horas | 1 autor, 1 cambio
public String Panel2StatusBar { get; set; }
private ObservableCollection<CourseModel> _CoursesCollection;
4 referencias | Camacho Hidalgo, Juan Jose, Hace 4 horas | 1 autor, 1 cambio
public ObservableCollection<CourseModel> CoursesCollection {...}

private ObservableCollection<StudentModel> _StudentsCollection;
2 referencias | Camacho Hidalgo, Juan Jose, Hace 4 horas | 1 autor, 1 cambio
public ObservableCollection<StudentModel> StudentsCollection {...}
2 referencias | Camacho Hidalgo, Juan Jose, Hace 4 horas | 1 autor, 1 cambio
public void StudentInCourse() {...}

0 referencias | Camacho Hidalgo, Juan Jose, Hace 4 horas | 1 autor, 1 cambio
public Data() {...}

0 referencias | Camacho Hidalgo, Juan Jose, Hace 4 horas | 1 autor, 1 cambio
public ICommand SearchCommand {...}

5 referencias | Camacho Hidalgo, Juan Jose, Hace 4 horas | 1 autor, 1 cambio
public CourseModel CurrentCourse {...}
2 referencias | Camacho Hidalgo, Juan Jose, Hace 4 horas | 1 autor, 1 cambio
private void Search() {...}
1 referencia | Camacho Hidalgo, Juan Jose, Hace 4 horas | 1 autor, 1 cambio
public bool InstantSearch { get; set; }

2 referencias | Camacho Hidalgo, Juan Jose, Hace 4 horas | 1 autor, 1 cambio
public string SearchName {...}

0 referencias | Camacho Hidalgo, Juan Jose, Hace 4 horas | 1 autor, 1 cambio
public ICommand AddCourse {...}
0 referencias | Camacho Hidalgo, Juan Jose, Hace 4 horas | 1 autor, 1 cambio
public ICommand DeleteCourse {...}
0 referencias | Camacho Hidalgo, Juan Jose, Hace 4 horas | 1 autor, 1 cambio
public ICommand SaveChanges {...}
```

Por otro lado, incluye la clase ViewModelBase que implementa la interfaz INotifyPropertyChanged, para que en cuanto algo cambie, salte un evento de notificación, y además RelayCommand para comprobar si puede ejecutar un comando.

Los mensajes que muestra la app, se muestran mediante MsgBoxService que implementa IMsgBoxService, con el método Show (mostrar). Usa un ServiceLocator para las instancias del modelo de datos.

Los modelos son iguales que la aplicación anterior. Pero la vista si cambia. La vista se trata de Data.xaml, y no incluye nada en su constructor, ya que eso corresponde al modelo. La vista es la siguiente:

Cursos y sus alumnos

Data

⬆ Search Filter

Nombre

Buscar

☐ Búsqueda instantánea

Nombre Duracion Descripcion Asignatura Tutor

Nombre Apellidos Email Nombre de usuario Contraseña

Nombre

Duración

Descripción

Add Course

Delete Course

Save Changes