



# CREA TU PRIMER CHAT EN TIEMPO REAL

## Descripción

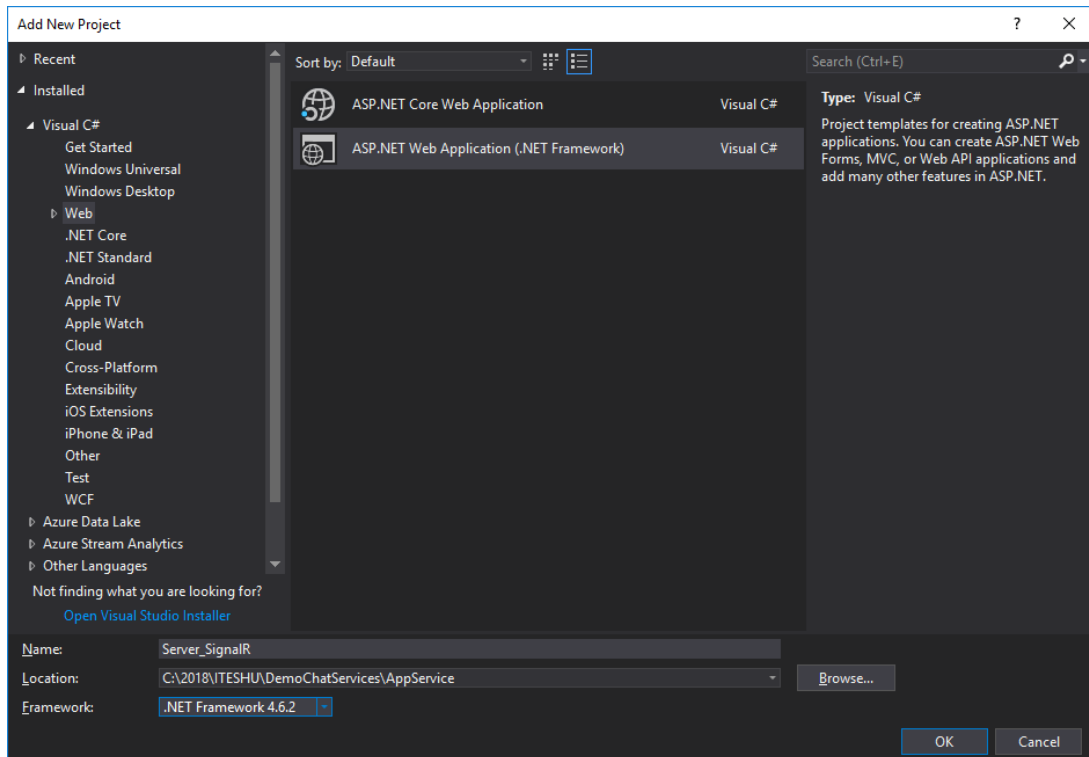
En este documento se lleva a cabo los pasos que son realizados para crear chat en tiempo real, utilizando ASP.NET SignalR como servidor y Xamarin Forms como clientes

Martín Basurto Díaz  
mbdz.msi@hotmail.com

## Aplicación servidor

### Paso 1 Crear la aplicación servidor

Abrimos Visual Studio -> Nuevo proyecto -> Visual C# -> Web -> ASP.NET Web Application (.NET Framework). Asignamos un nombre al nuevo proyecto, una ubicación, seleccionamos la plantilla Web API y damos OK.



New ASP.NET Web Application - Server\_SignalR

ASP.NET 4.6.2 Templates

Empty Web Forms MVC **Web API** Single Page Application

Azure API App Azure Mobile App

A project template for creating RESTful HTTP services that can reach a broad range of clients including browsers and mobile devices.

[Learn more](#)

Change Authentication

Authentication: **No Authentication**

Add folders and core references for:

☐ Web Forms ☒ MVC ☒ Web API

☐ Enable Docker support (Requires [Docker for Windows](#))

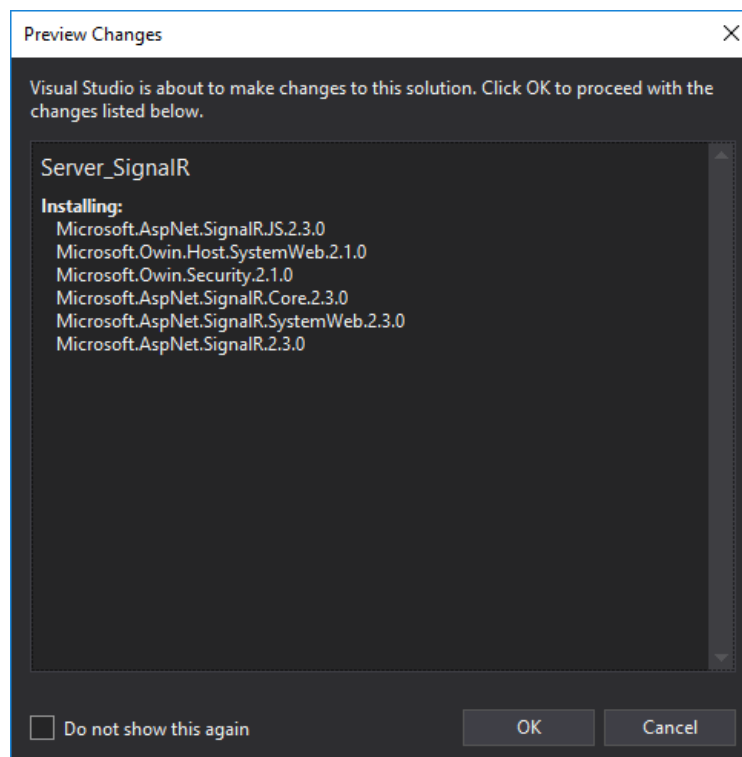
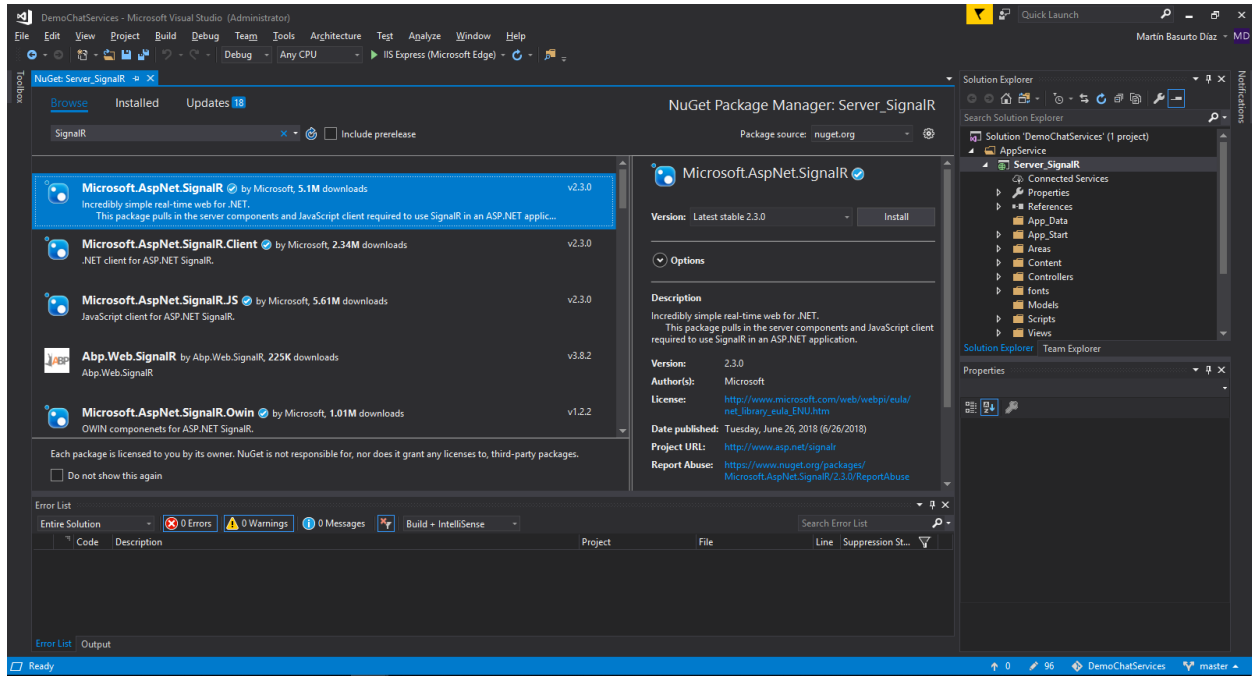
☐ Add unit tests

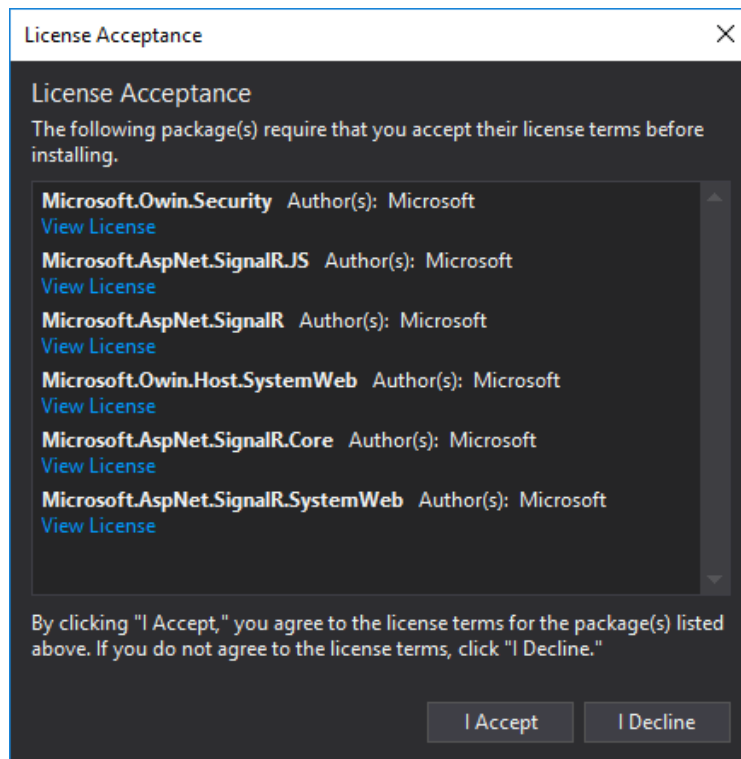
Test project name:

OK Cancel

## Paso 2 Agregando paquetes Nuget

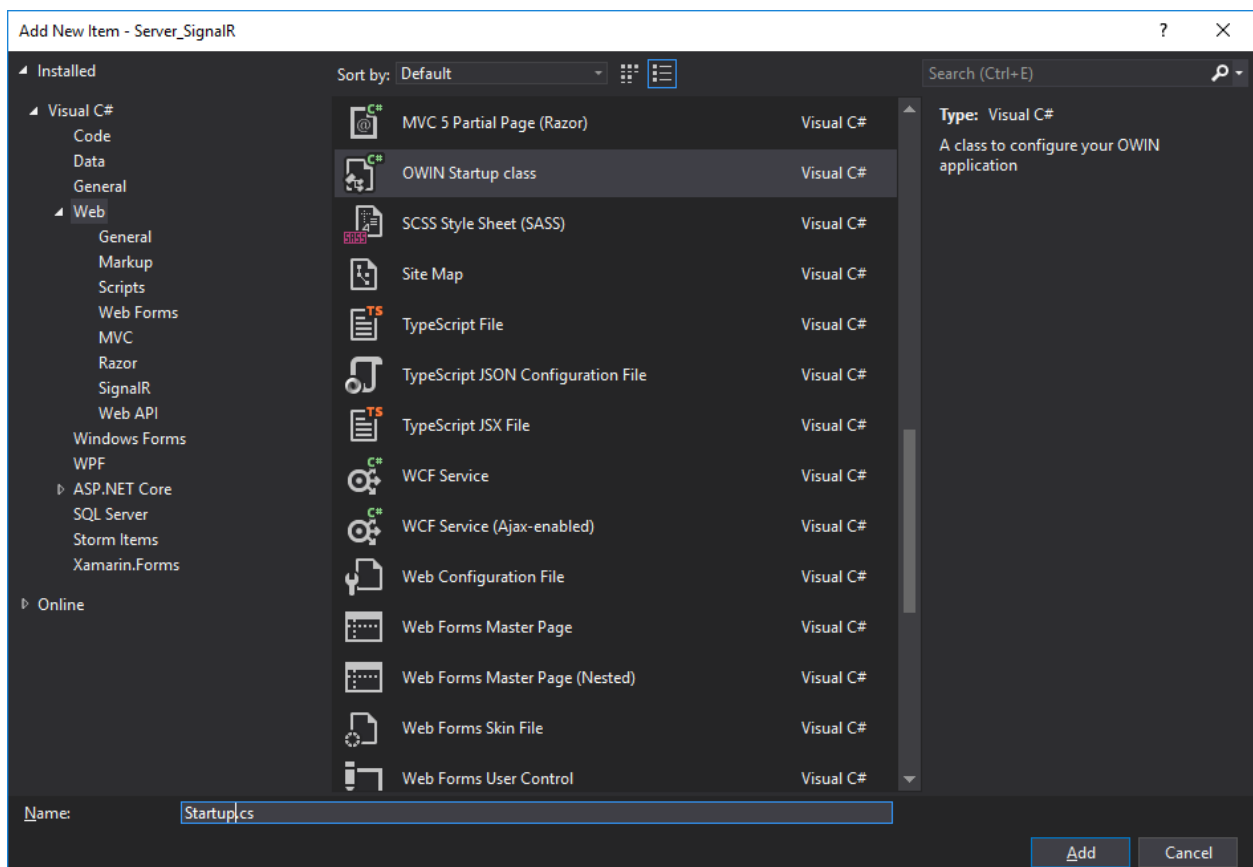
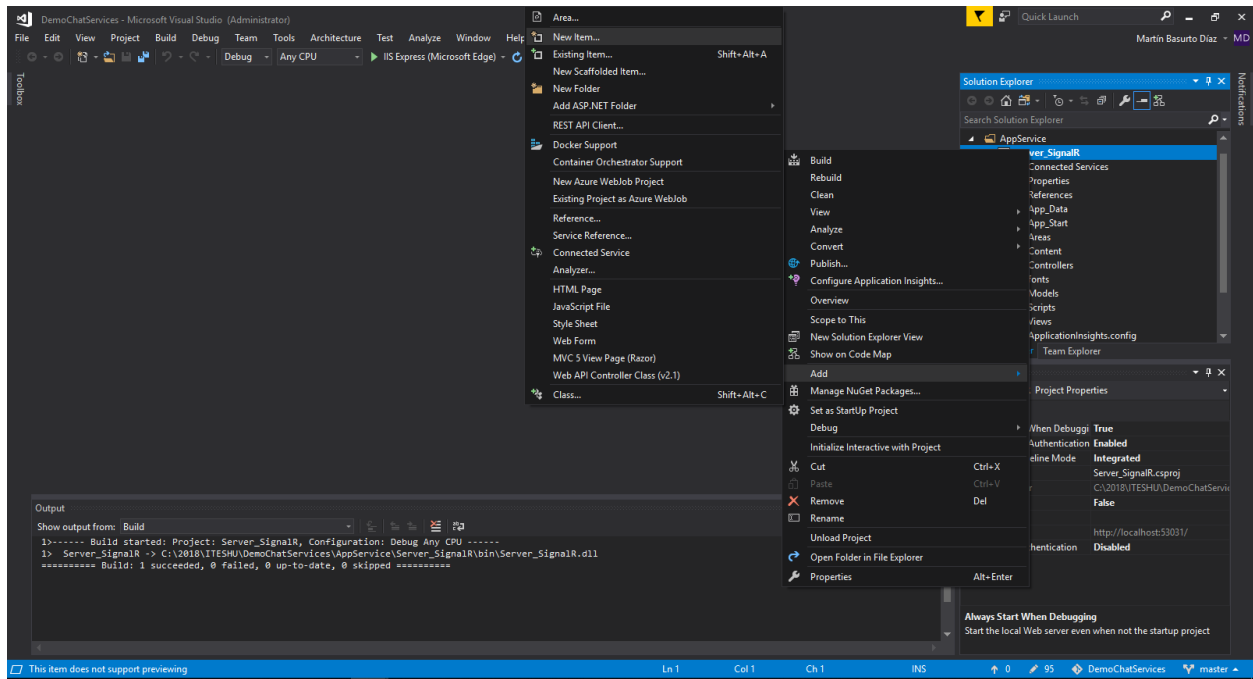
Vamos a nuestro Explorador de soluciones -> Nombre del proyecto -> Administrador de paquetes Nuget -> Explorar. Buscamos el paquete Nuget Microsoft.AspNet.SignalR y damos instalar. Aceptamos los cambios y licencias.





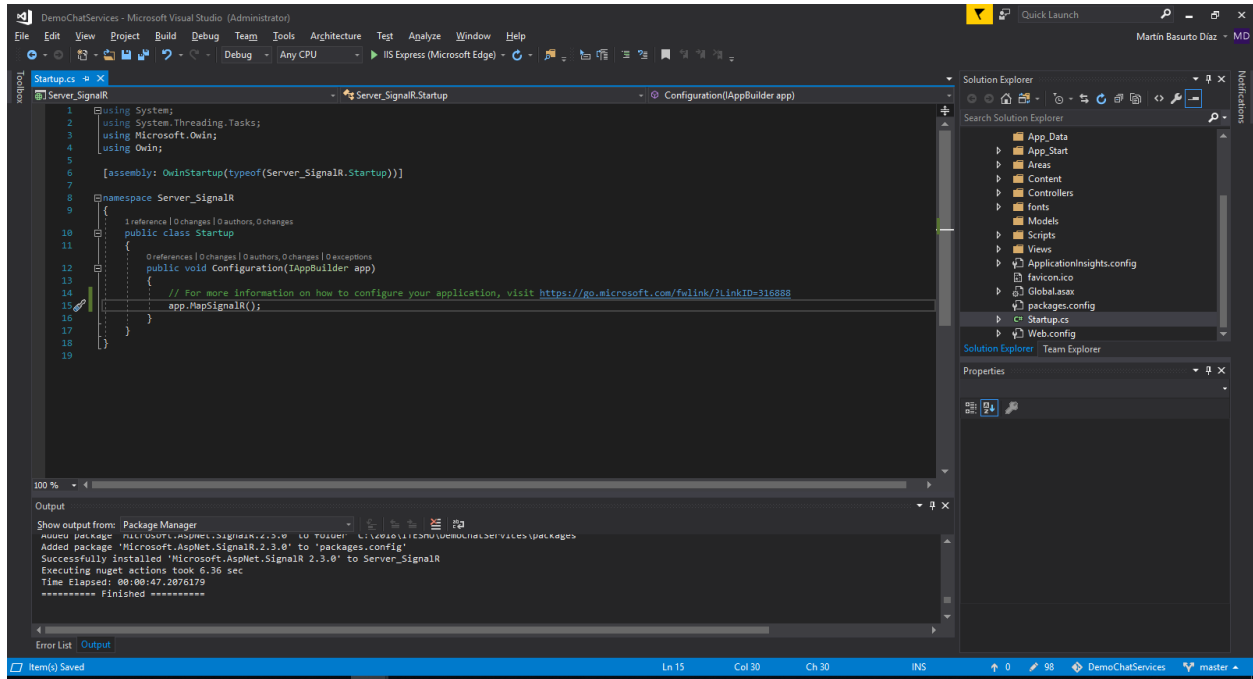
### Paso 3 Configurando la clase Startup

Vamos a nuestro Explorador de soluciones -> Nombre del proyecto -> Agregar -> Nuevo elemento. Seleccionamos la plantilla OWIN Startup class, le asignamos el nombre de Startup.cs y damos Agregar.



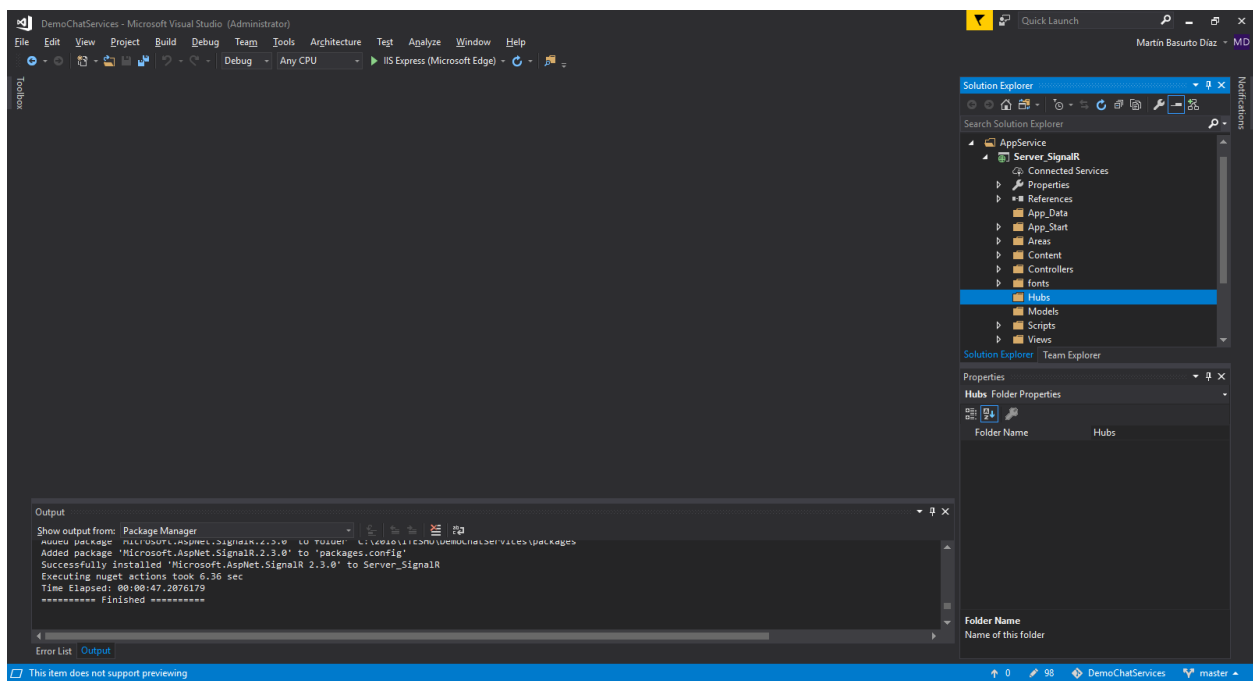
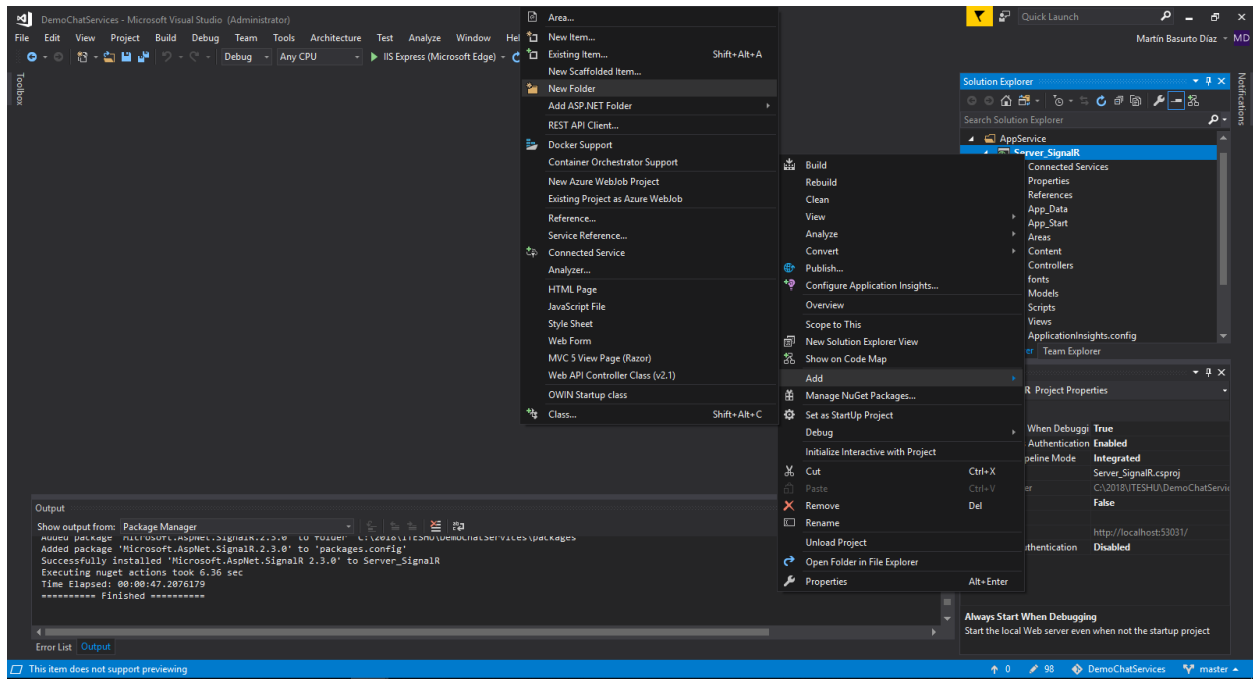
En la clase escribimos en método Configuration el siguiente código.

```
app.MapSignalR();
```



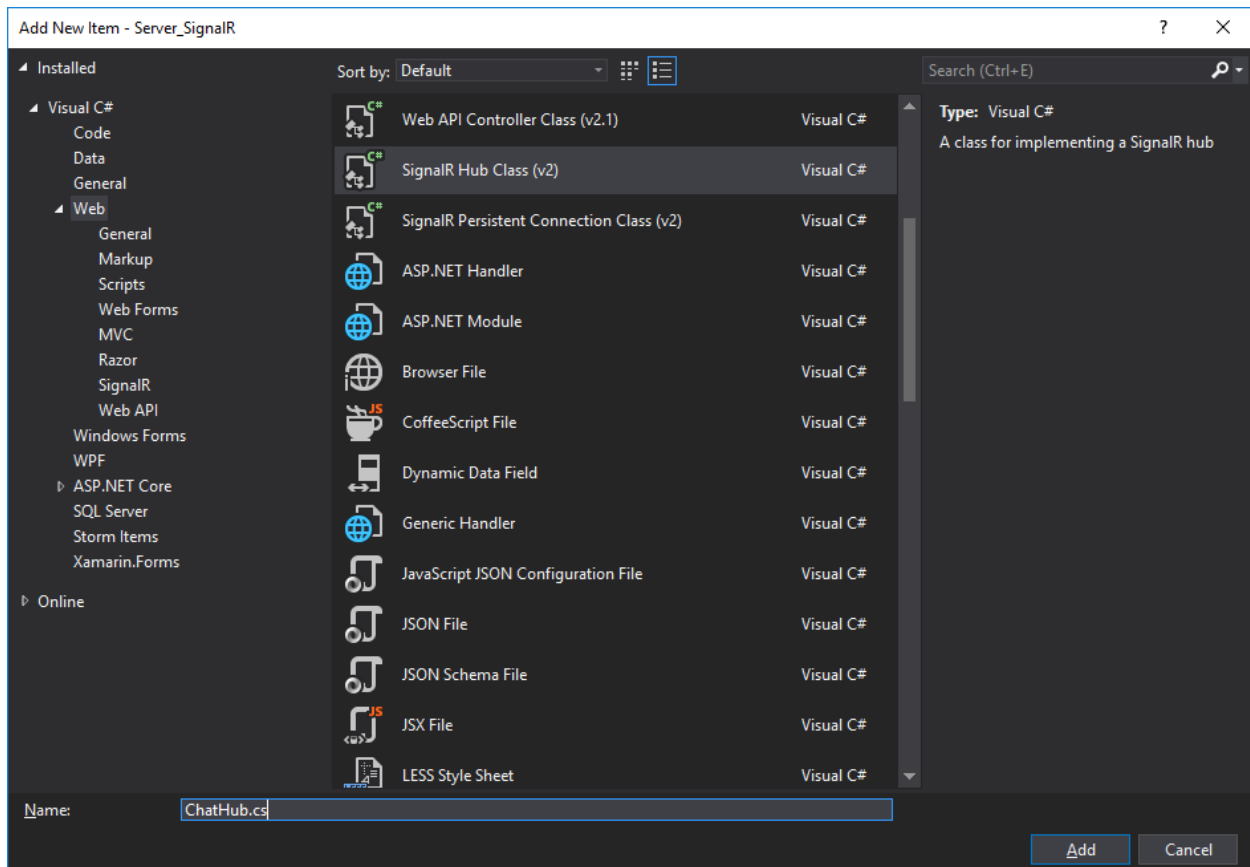
## Paso 4 Configuración del Hub

Vamos a nuestro Explorador de soluciones -> Nombre del proyecto -> Agregar -> Nueva carpeta. Le asignamos el nombre de Hubs.





Vamos a nuestro Explorador de soluciones -> Nombre del proyecto -> Hubs -> Agregar -> Nuevo elemento. Seleccionamos la plantilla SignalR Hub Class (v2), le asignamos el nombre de ChatHub.cs y damos Agregar.

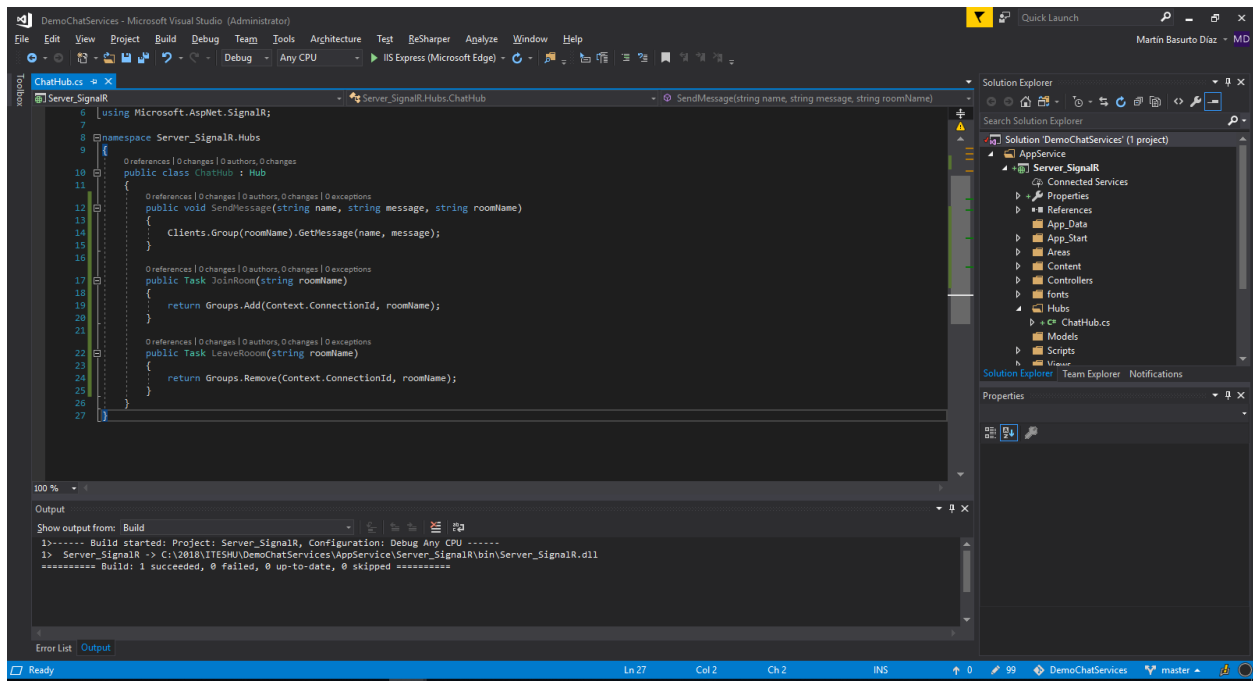


En la clase ChatHub.cs escribimos el siguiente código.

```
public void SendMessage(string name, string message, string roomName)
{
    Clients.Group(roomName).GetMessage(name, message);
}

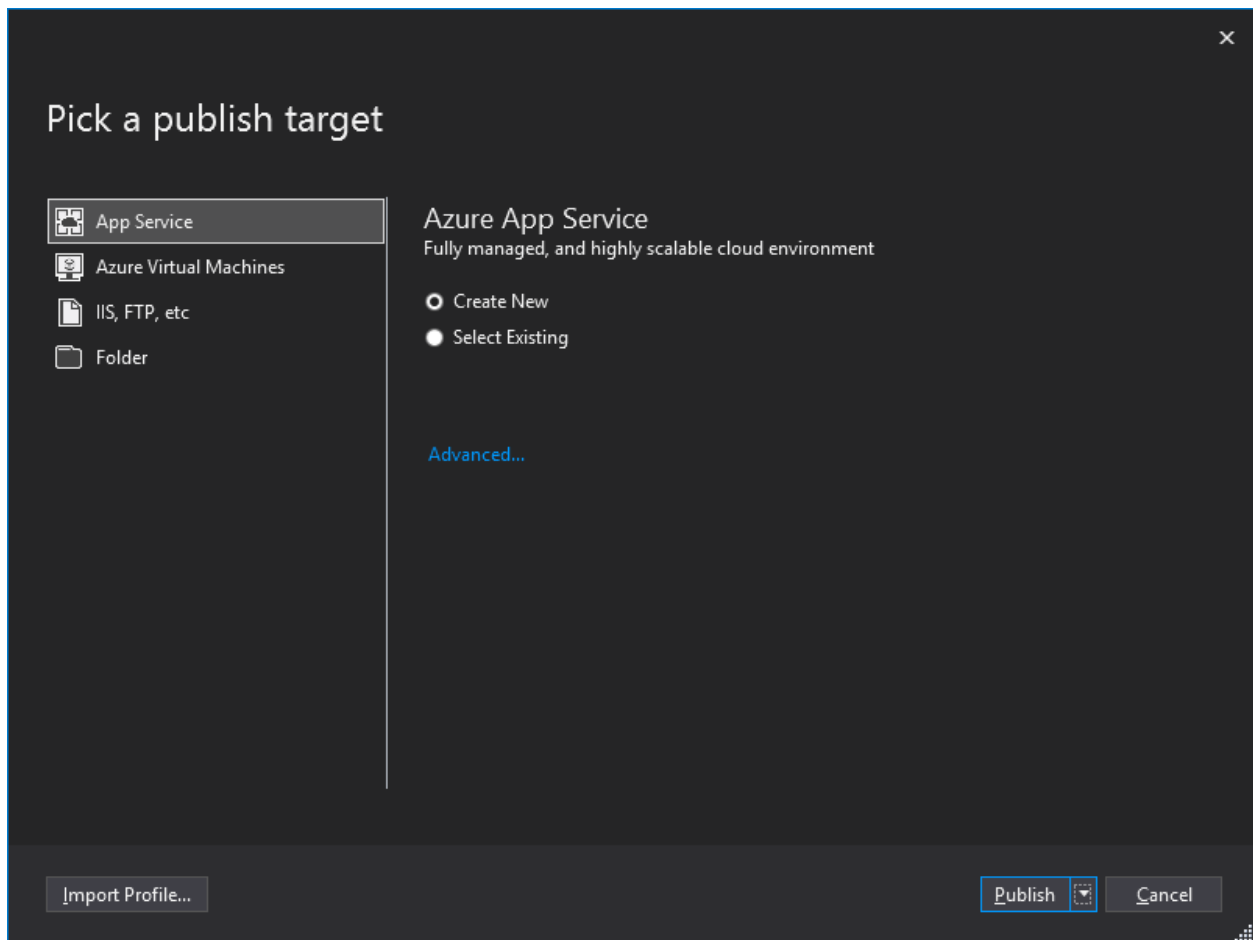
public Task JoinRoom(string roomName)
{
    return Groups.Add(Context.ConnectionId, roomName);
}

public Task LeaveRoom(string roomName)
{
    return Groups.Remove(Context.ConnectionId, roomName);
}
```




### Paso 5 Publicación de la aplicación servidor


Vamos a nuestro Explorador de soluciones -> Nombre del proyecto -> Publicar. Seleccionamos App Service -> Crear nuevo y damos Publicar. Inicialmente cargará nuestras suscripciones de Azure.



## Create App Service

Host your web and mobile applications, REST APIs, and more in Azure

 Microsoft account  
mbd.97@hotmail.com

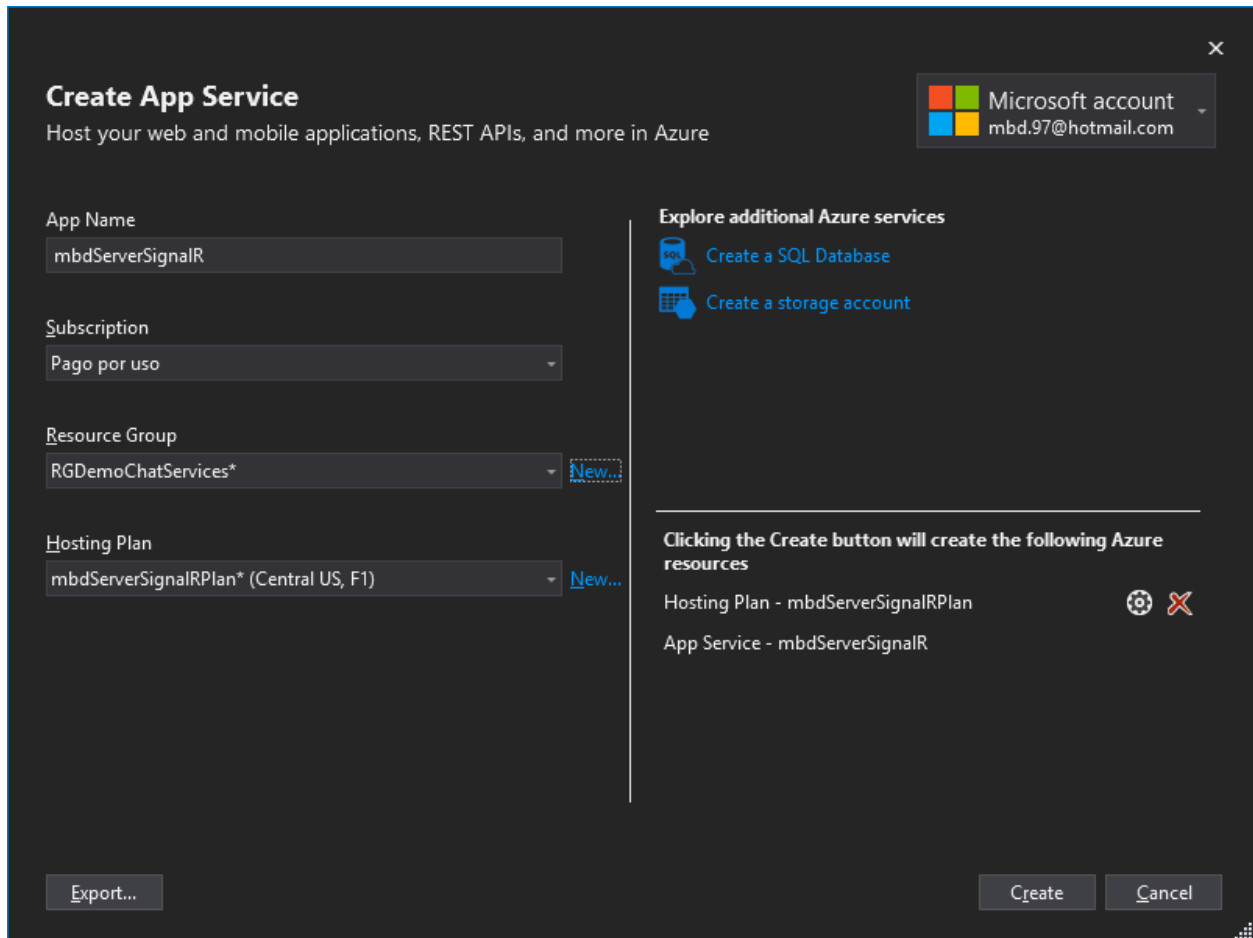
 Loading Subscriptions...

Export...

Create

Cancel

Asignamos un nombre a la app, seleccionamos una suscripción, creamos un nuevo grupo de recursos, y generamos un plan hosting y damos en crear.



**Create App Service**  
Host your web and mobile applications, REST APIs, and more in Azure

Microsoft account  
mbd.97@hotmail.com

App Name  
mbdServerSignalR

Subscription  
Pago por uso

Resource Group  
RGDemoChatServices\* [New...](#)

Hosting Plan  
mbdServerSignalRPlan\* (Central US, F1) [New...](#)

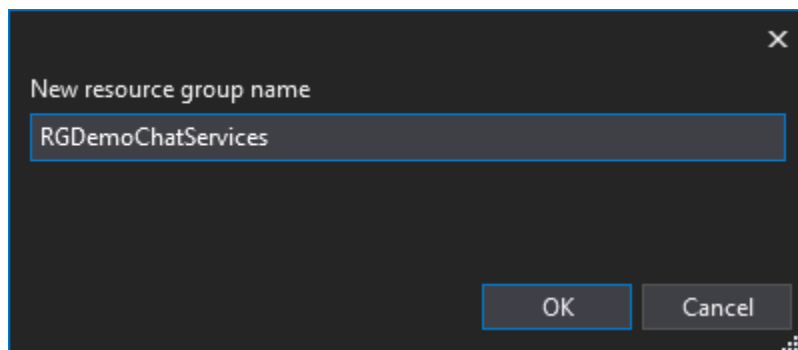
**Explore additional Azure services**

- [Create a SQL Database](#)
- [Create a storage account](#)

**Clicking the Create button will create the following Azure resources**

- Hosting Plan - mbdServerSignalRPlan
- App Service - mbdServerSignalR

[Export...](#) [Create](#) [Cancel](#)



**New resource group name**

RGDemoChatServices

[OK](#) [Cancel](#)

×

## Configure Hosting Plan

A hosting plan is the container for your app. The hosting plan settings will determine the location, features, cost and compute resources associated...

App Service Plan

mbdServerSignalRPlan

Location

Central US

Size

Free

OK

Cancel

Create App Service

Host your web and mobile applications, REST APIs, and more in Azure

Microsoft account  
mbd.97@hotmail.com

App Name

mbdServerSignalR

Subscription

Pago por uso

Resource Group

RGDemoChatServices\*

New...

Hosting Plan

mbdServerSignalRPlan\* (Central US, F1)

New...

Explore additional Azure services

Create a SQL Database

Create a storage account

Clicking the Create button will create the following Azure resources

Hosting Plan - mbdServerSignalRPlan

App Service - mbdServerSignalR

Deploying: Step 1...

Create

Cancel

Home Page

mbdserverignalr.azurewebsites.net/

Application name

Home

API

ASP.NET

ASP.NET is a free web framework for building great Web sites and Web applications using HTML, CSS, and JavaScript.

Learn more »

Getting started

ASP.NET Web API is a framework that makes it easy to build HTTP services that reach a broad range of clients, including browsers and mobile devices. ASP.NET Web API is an ideal platform for building RESTful applications on the .NET Framework.

Learn more »

Get more libraries

NuGet is a free Visual Studio extension that makes it easy to add, remove, and update libraries and tools in Visual Studio projects.

Learn more »

Web Hosting

You can easily find a web hosting company that offers the right mix of features and price for your applications.

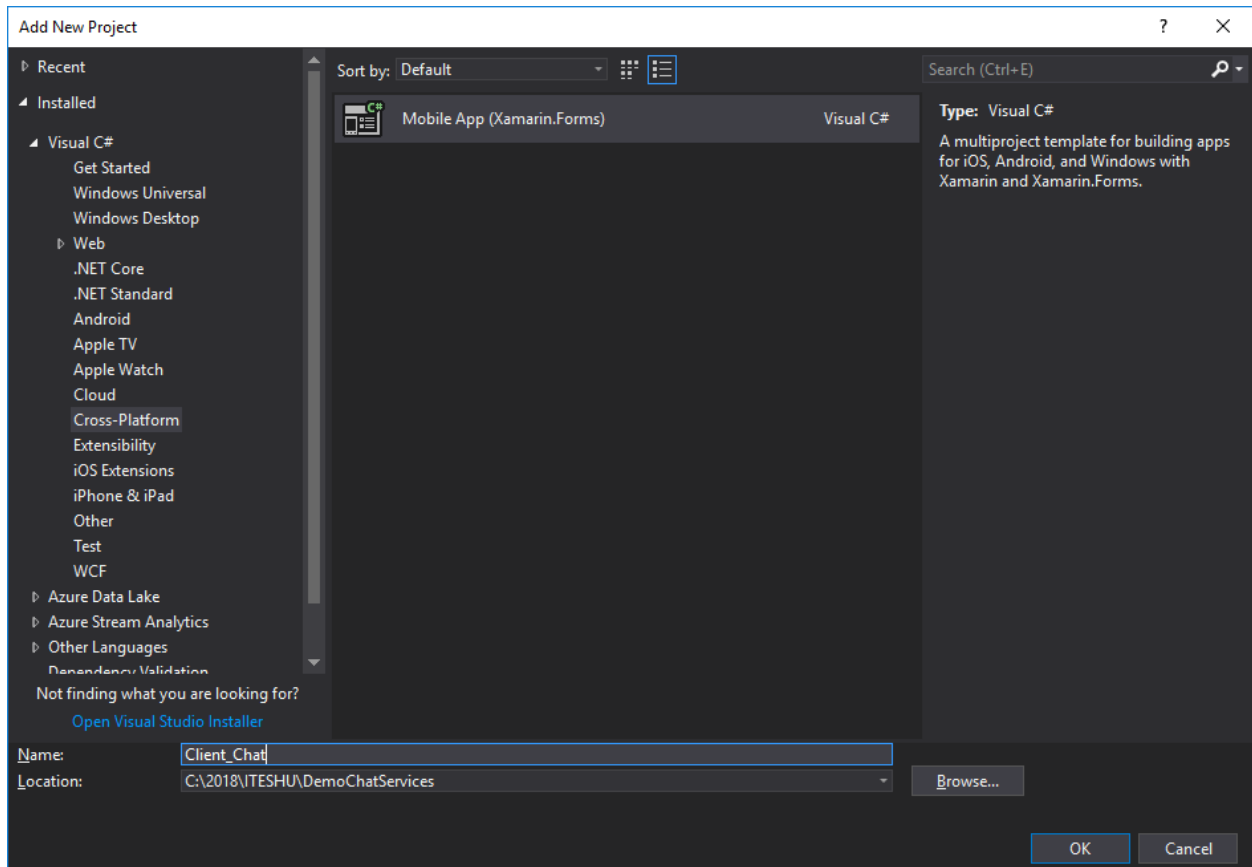
Learn more »

© 2018 - My ASP.NET Application

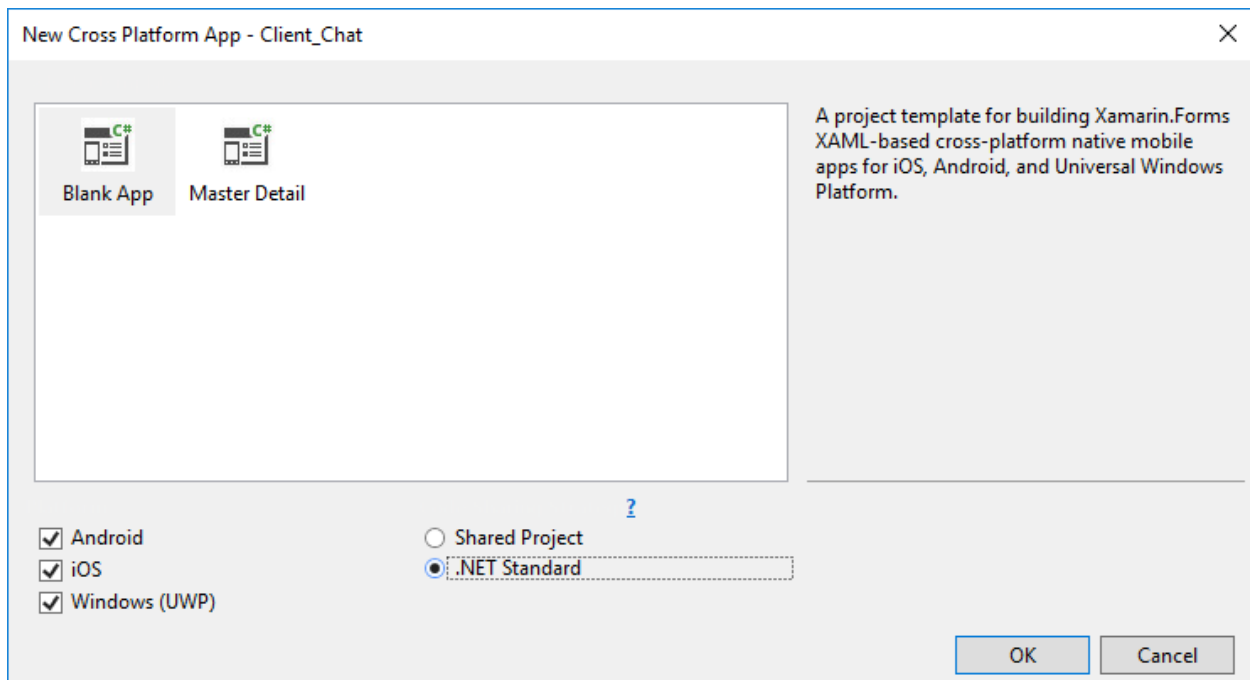
## Aplicación cliente

### Paso 1 Crear la aplicación cliente

Vamos a nuestro Explorador de soluciones -> Agregar -> Nuevo Proyecto -> Visual C# -> Cross-Platform -> Mobile App (Xamarin.Forms). Le asignamos un nombre, seleccionamos la ruta y damos Ok. Después en New Croos Platform App, Seleccionamos Blank App y .NET Statandar y damos Ok.

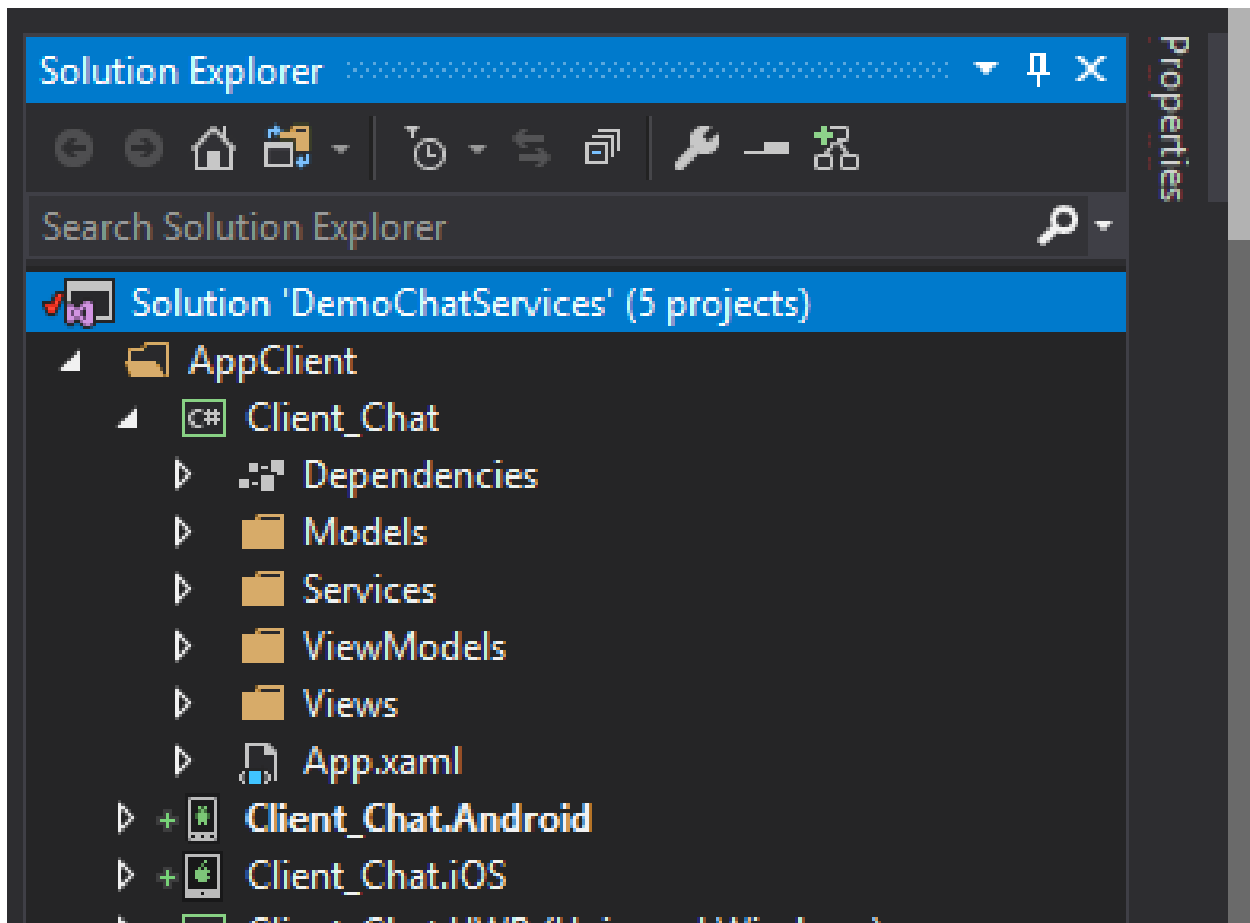






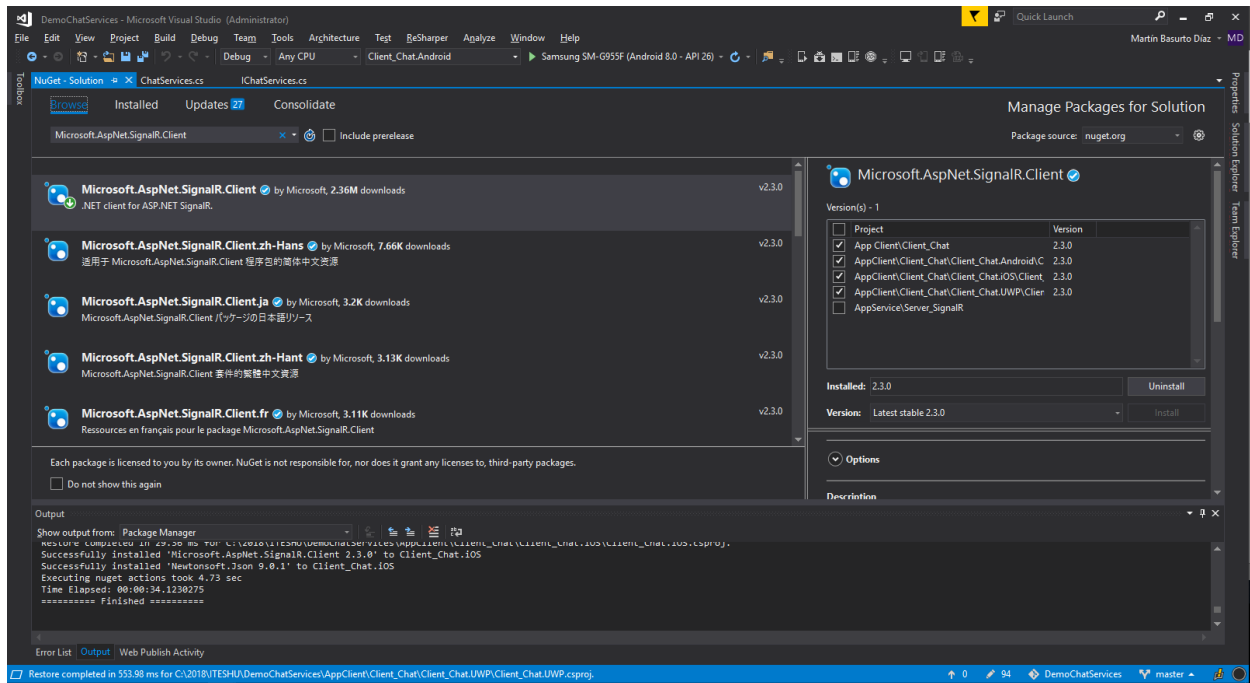
## Paso 2 Creación de la estructura del proyecto

Se realiza la creación de las siguientes carpetas en el proyecto croos, para darle una estructura al código. Las carpetas a creas son Models, Services, ViewModels, Views.



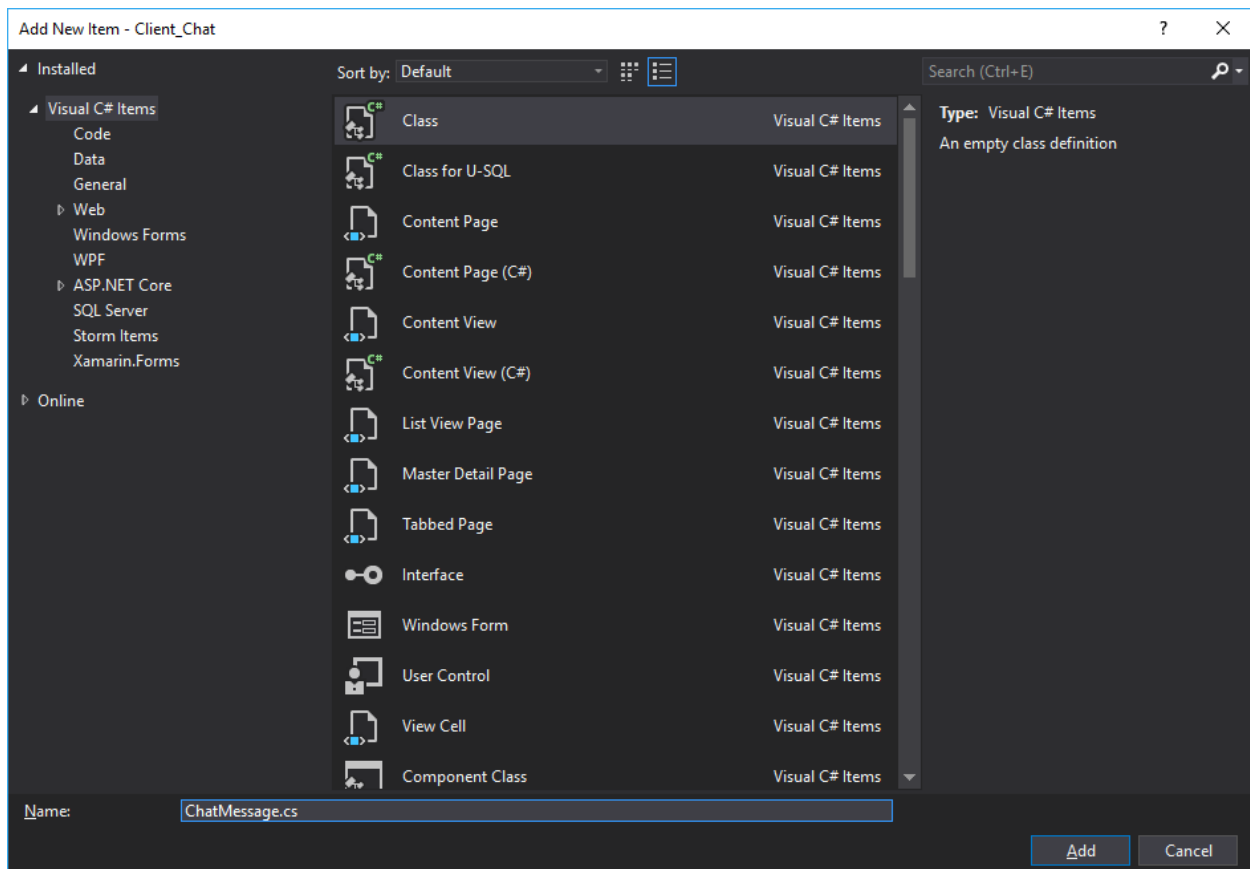
### Paso 3 Agregando paquetes Nugets

Vamos al Explorador de soluciones -> Nombre del proyecto -> Administrador de Nugets -> Explorar. Buscamos el paquete nuget Microsoft.AspNet.SignalR.Client, seleccionamos los proyectos cross (Nombre de proyecto) y clientes (Nombre de proyecto.Android, Nombre de proyecto.iOS y Nombre de proyecto.UWP), posteriormente le damos instalar y aceptamos los cambios y licencias.



## Paso 4 Creación del Modelo

Vamos al Explorador de soluciones -> Nombre del proyecto -> Models -> Agregar -> Nuevo elemento -> Visual C# -> Clase. Le asignamos el nombre de ChatMessage.cs.

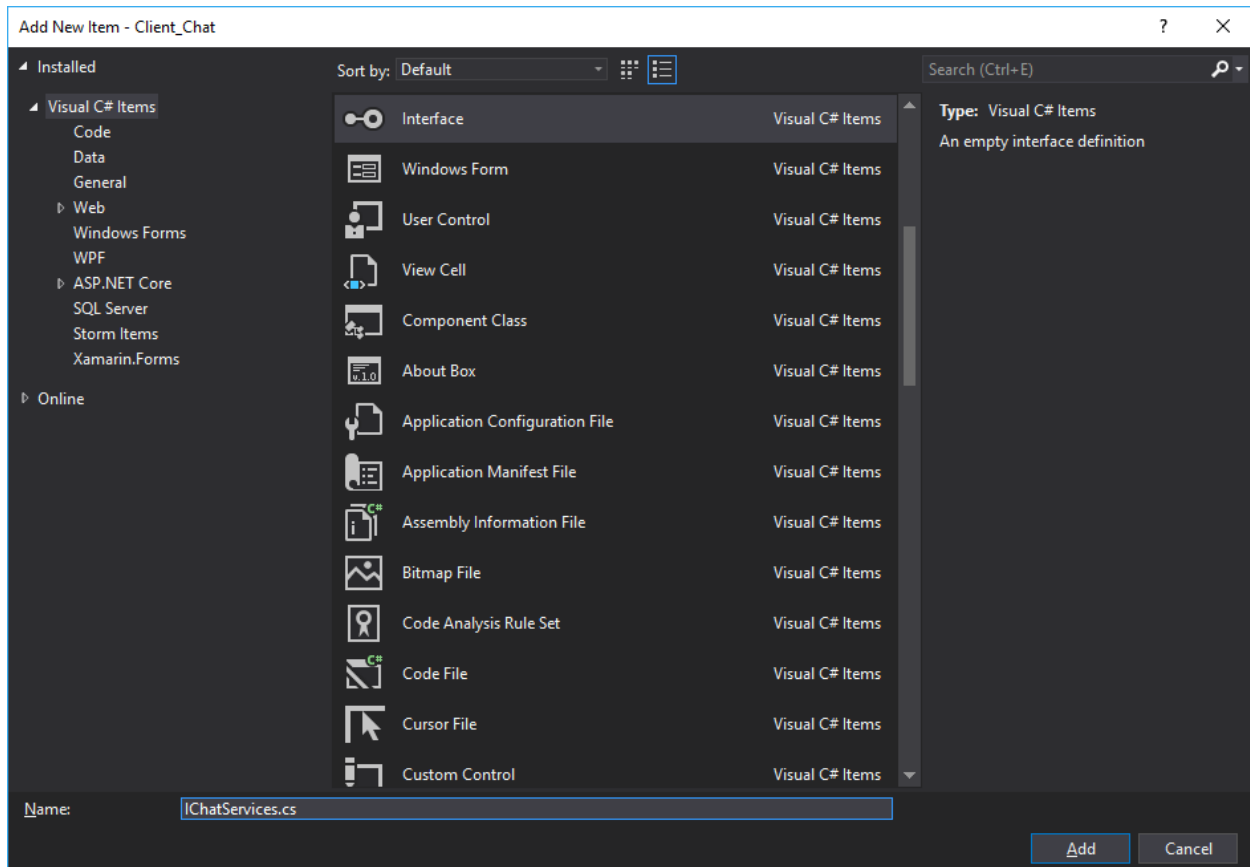


A la clase la hacemos publica y le asignamos el siguiente código.

```
public class ChatMessage
{
    public string Name { get; set; }
    public string Message { get; set; }
}
```

## Paso 5 Creación clases de servicios

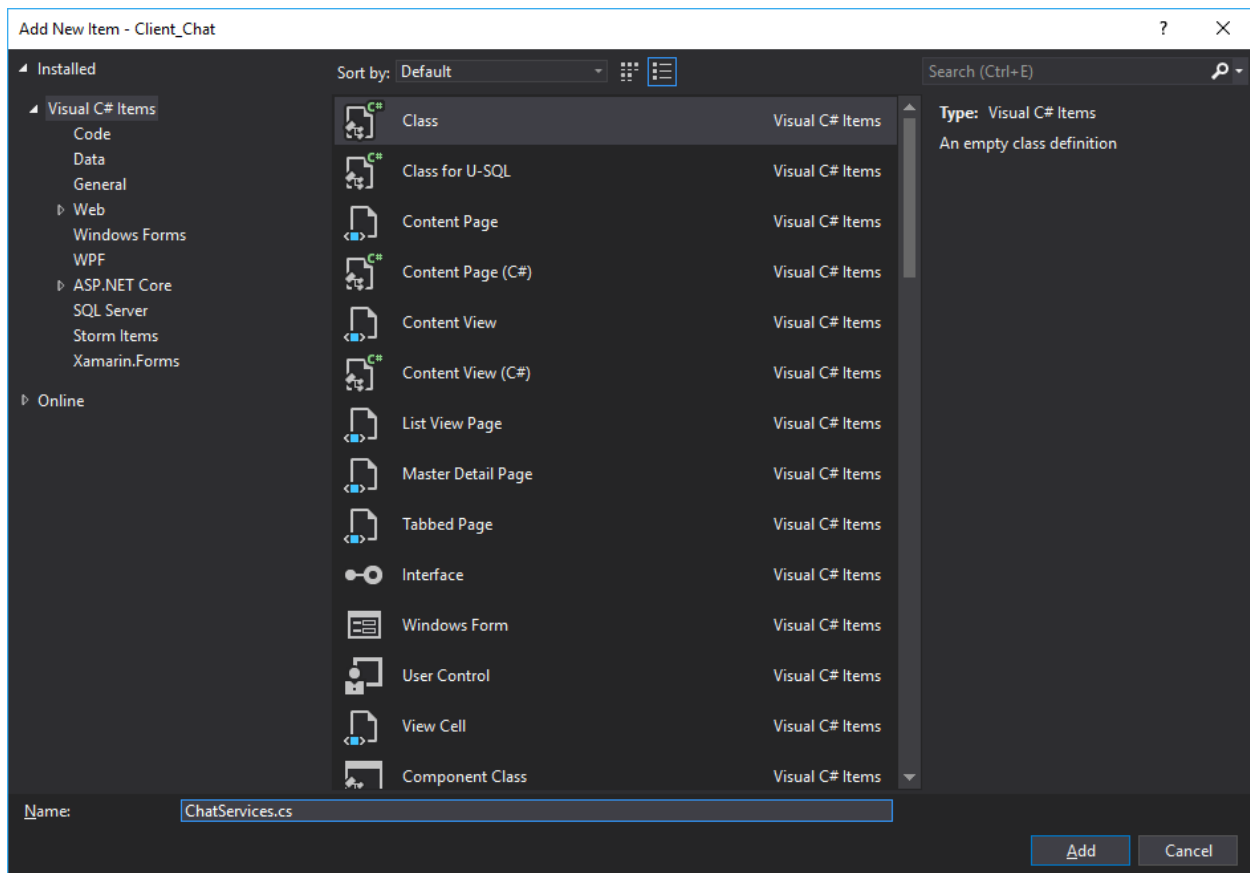
Vamos al Explorador de soluciones -> Nombre del proyecto -> Services -> Agregar -> Nuevo elementos -> Visual C# -> Interfaz. Le asignamos el nombre de IChatServices.cs.



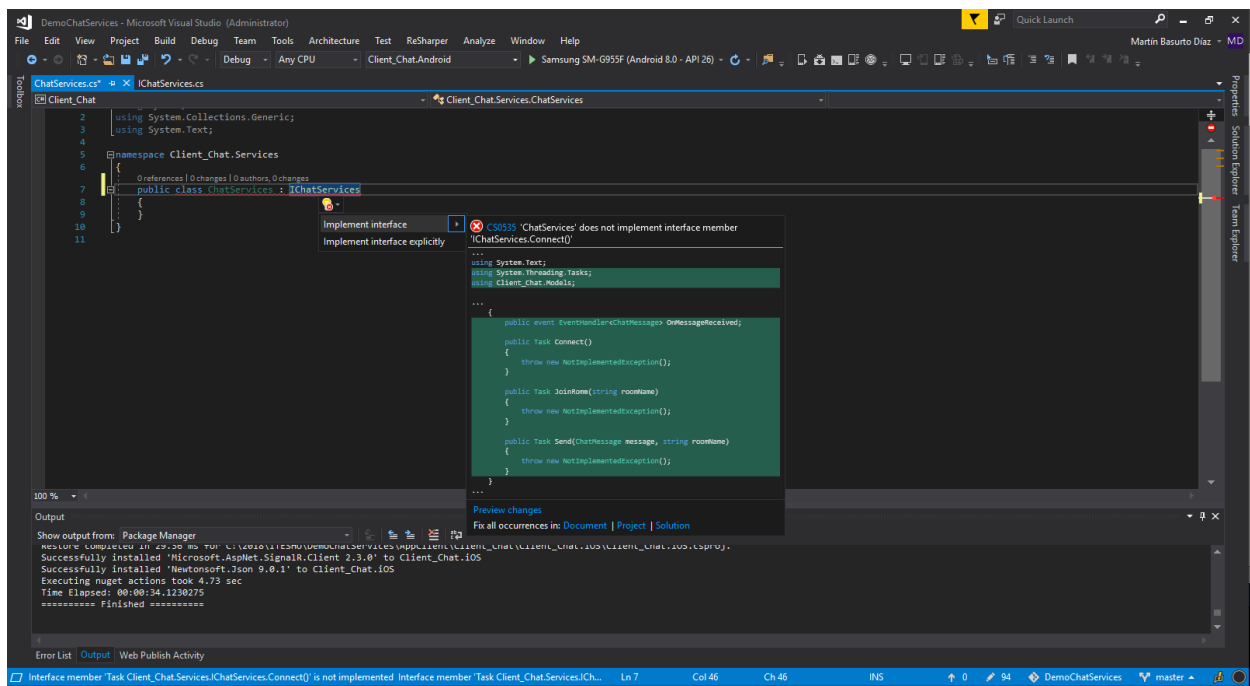
A la interfaz la hacemos publica y le escribimos el siguiente código.

```
public interface IChatServices
{
    Task Connect();
    Task Send(ChatMessage message, string roomName);
    Task JoinRoom(string roomName);
    event EventHandler<ChatMessage> OnMessageReceived;
}
```

Vamos al Explorador de soluciones -> Nombre del proyecto -> Services -> Agregar -> Nuevo elemento -> Visual C# -> Clase. Le asignamos el nombre de ChatServices.cs.



A la clase la hacemos publica, le heredamos la interfaz IChatServices y por ultimo hacemos la implementación de la interfaz.



En la clase le agregamos las siguientes variables.

```
private readonly HubConnection connection;  
private readonly IHubProxy proxy;
```

En el constructor de la clase le escribimos el siguiente código.

```
public ChatServices()  
{  
    connection = new HubConnection("http://mbdserver.signalr.azurewebsites.net/");  
    proxy = connection.CreateHubProxy("ChatHub");  
}
```

En el método Connect lo hacemos asíncrono y le escribimos el siguiente código.

```
public async Task Connect()  
{  
    connection.Start().Wait();  
  
    proxy.On("GetMessage",  
        (string name, string message) =>  
            OnMessageReceived(this, new ChatMessage {Message = message, Name =  
name}));  
}
```

En el método Send lo hacemos asíncrono y le escribimos el siguiente código.

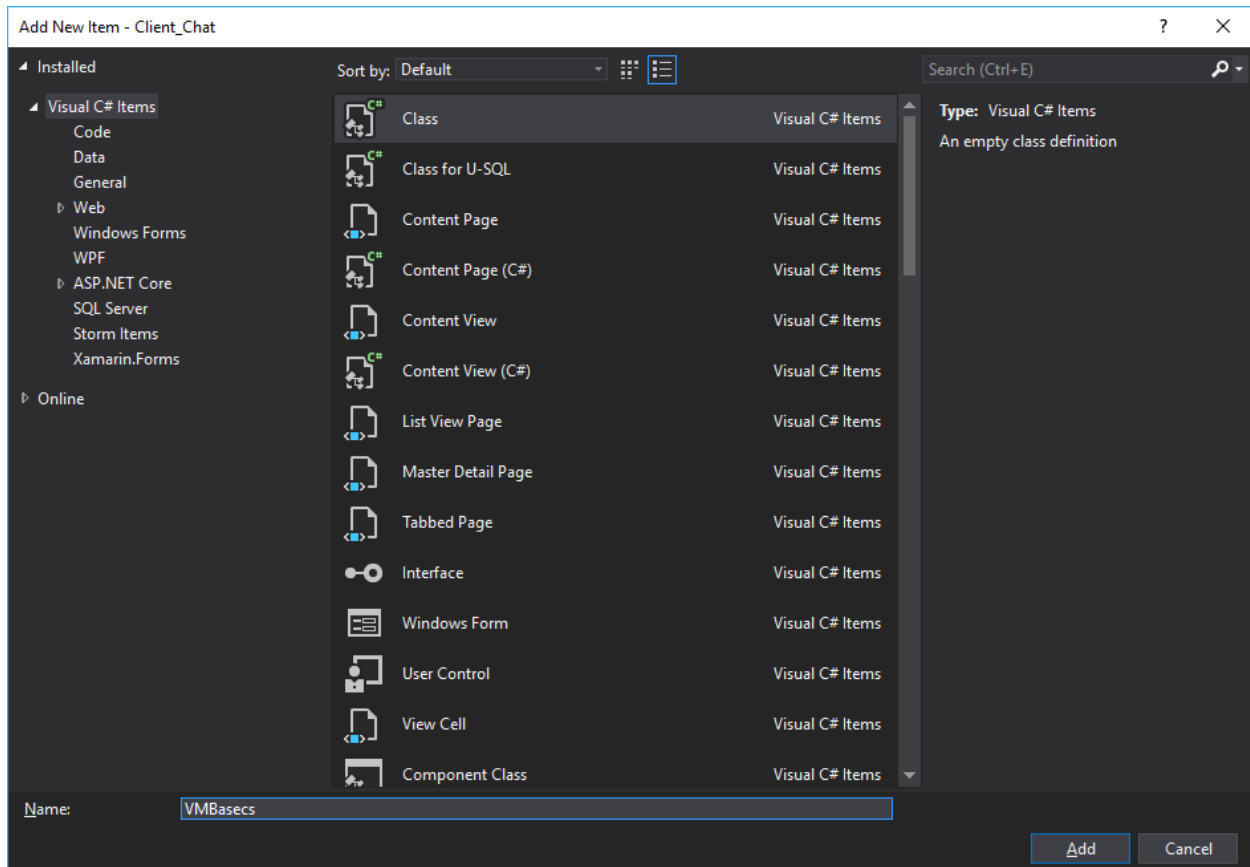
```
public async Task Send(ChatMessage message, string roomName)  
{  
    await proxy.Invoke("SendMessage", message.Name, message.Message, roomName);  
}
```

En el método JoinRoom lo hacemos asíncrono y le escribimos el siguiente código.

```
public async Task JoinRoom(string roomName)  
{  
    await proxy.Invoke("JoinRoom", roomName);  
}
```

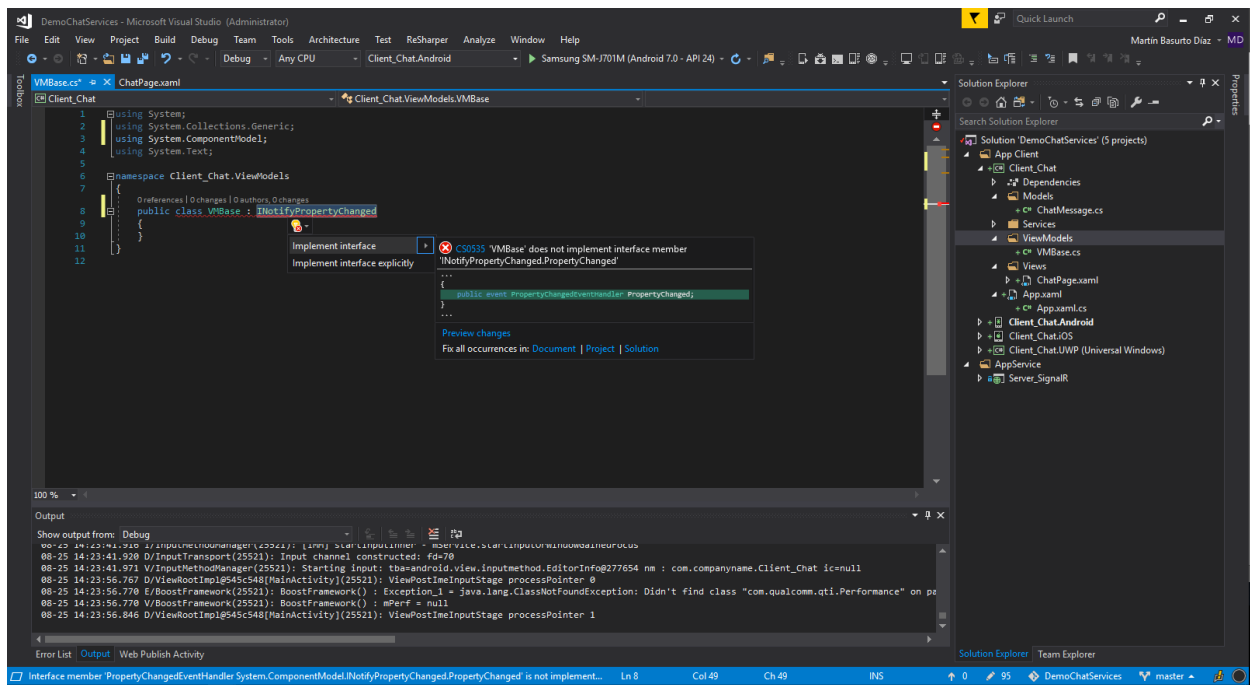
## Paso 6 Creación del ViewModel

Vamos al Explorador de soluciones -> Nombre de proyecto -> ViewModels -> Agregar -> Nuevo elemento -> Visual C# -> Clase. Le asignamos el nombre de VMBase.cs.



En la clase la hacemos publica, le heredamos INotifyPropertyChanged y hacemos la implementación.





En la misma clase le agregamos las siguientes propiedades.

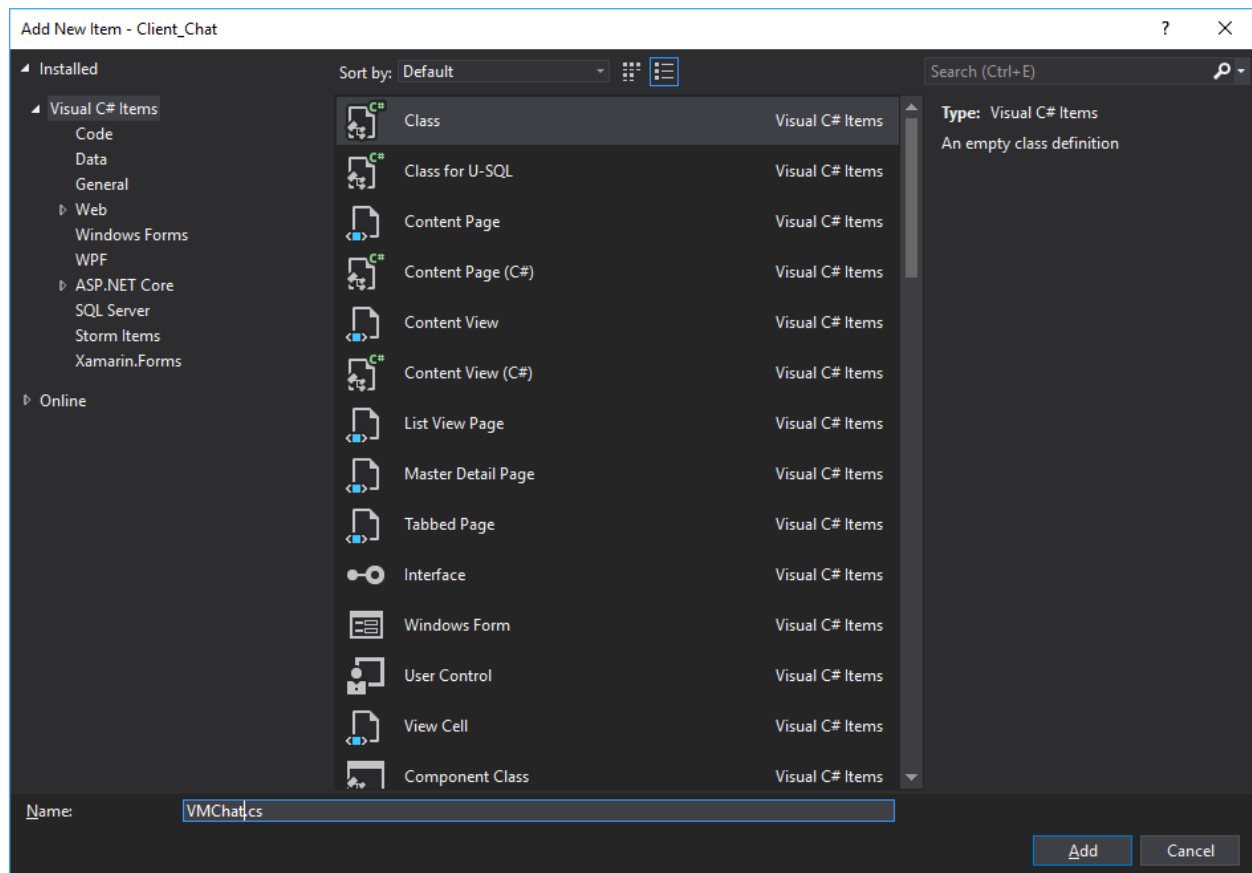
```
private bool isBusy;

public bool IsBusy
{
    get { return isBusy; }
    set
    {
        isBusy = value;
        OnPropertyChanged();
    }
}

private string title;

public string Title
{
    get { return title; }
    set
    {
        title = value;
        OnPropertyChanged();
    }
}
```

Vamos al Explorador de soluciones -> Nombre del proyecto -> ViewModels -> Agregar -> Nuevo elemento -> Visual C# -> Clase. Le asignamos el nombre de VMChat.cs. Hacemos publica la clase, y hacemos herencia de VMBase.



Dentro la clase se le va a agregar las siguientes variables y propiedades.

```
private IChatServices chatServices;
private string roomName = "SistemasRoom";

public Command SendMessageCommand { get; set; }

private ObservableCollection<ChatMessage> messages;

public ObservableCollection<ChatMessage> Messages
{
    get { return messages; }
    set
    {
        messages = value;
        OnPropertyChanged();
    }
}

private ChatMessage writeMessage;

public ChatMessage WriteMessage
{
    get { return writeMessage; }
    set
    {
        writeMessage = value;
        OnPropertyChanged();
    }
}
```

```
    }
}
```

Dentro del constructo de la clase se va a inicializar las propiedades y variables anteriormente definidas y se le agrega funcionalidad quedando de la siguiente forma.

```
public VMChat()
{
    Title = "ITESHU Chat";
    chatServices = DependencyService.Get<IChatServices>();
    chatServices = new ChatServices();

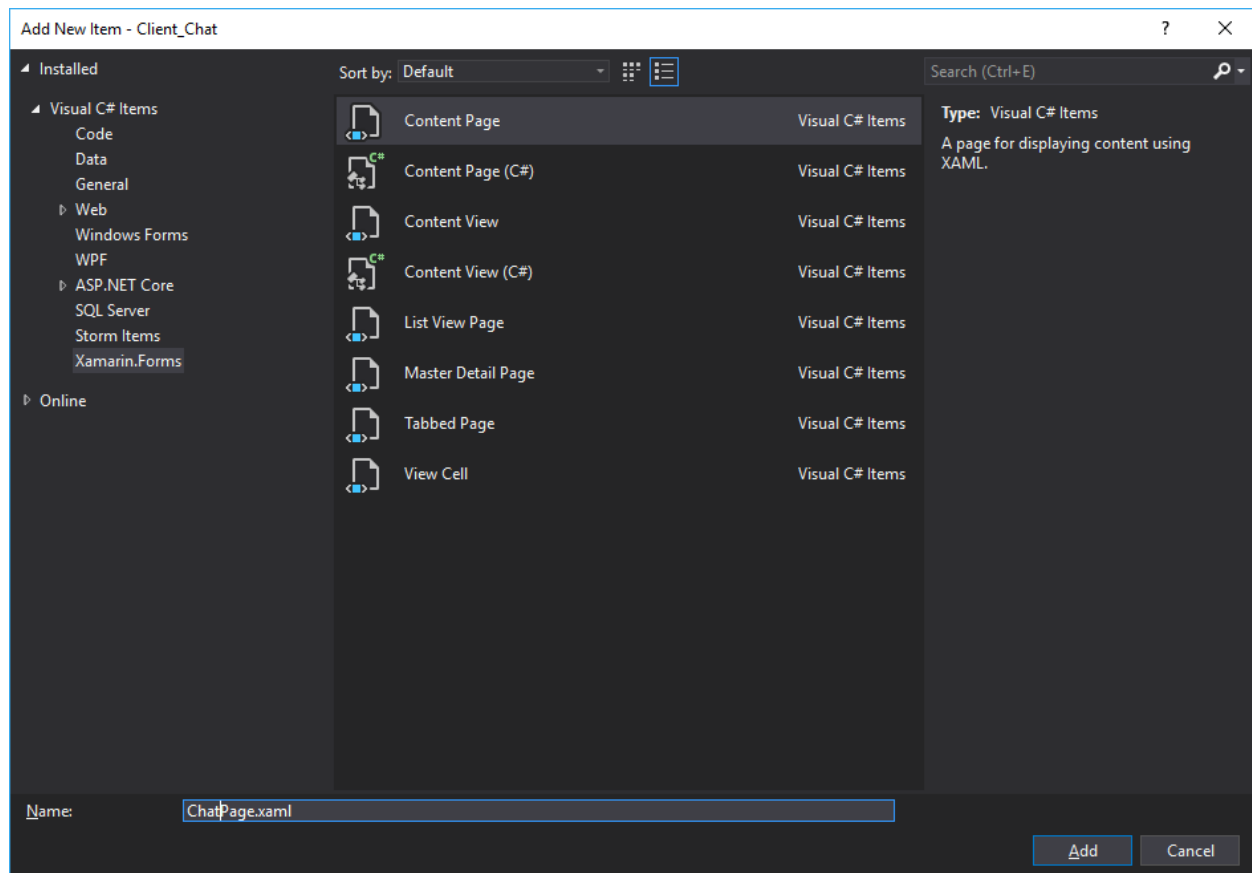
    Messages = new ObservableCollection<ChatMessage>();
    WriteMessage = new ChatMessage();

    chatServices.Connect();
    chatServices.JoinRoom(roomName);
    chatServices.OnMessageReceived += (sender, message) =>
    {
        Messages.Add(new ChatMessage {Name = message.Name, Message =
message.Message});
    };

    SendMessageCommand = new Command(async () =>
    {
        IsBusy = true;
        await chatServices.Send(new ChatMessage {Name = WriteMessage.Name,
Message = WriteMessage.Message},
            roomName);
        WriteMessage.Message = "";
        IsBusy = false;
    });
}
```

## Paso 7 Creación del View

Vamos a ir al Explorador de soluciones -> Nombre del proyecto -> Views -> Agregar -> Nuevo elemento -> Visual C# -> Xamarin.Forms -> Content Page. Le asignamos un nombre a la vista y damos Agregar.



Al agregar la página se nos genera dos archivos uno con terminación Page.xaml y otro Page.xaml.cs. Iniciamos a modificar el de la terminación Page.xaml y le agregamos el siguiente código dentro del ContentPage.Content.

```
<ContentPage.Content>
    <StackLayout>
        <Entry Text="{Binding WriteMessage.Name, Mode=TwoWay}" Placeholder="Nombre:"
VerticalOptions="Start" Keyboard="Text"/>
        <ScrollView VerticalOptions="FillAndExpand">
            <ListView ItemsSource="{Binding Messages, Mode=TwoWay}"
HorizontalOptions="FillAndExpand"
                HasUnevenRows="True" IsPullToRefreshEnabled="False"
SeparatorColor="Blue"
                HeightRequest="300" Margin="10,5,5,10" IsRefreshing="{Binding
IsBusy, Mode=TwoWay}">
                <ListView.ItemTemplate>
                    <DataTemplate>
                        <ViewCell>
                            <Grid>
                                <Grid.RowDefinitions>
                                    <RowDefinition Height="Auto"/>
                                    <RowDefinition Height="Auto"/>
                                </Grid.RowDefinitions>
                                <Label Grid.Row="0" Text="{Binding Name}"
FontAttributes="Bold" FontSize="Medium"/>
                                <Label Grid.Row="1" Text="{Binding Message}"
FontSize="Small"/>
                            </Grid>
                        </ViewCell>
                    </DataTemplate>
                </ListView.ItemTemplate>
            </ScrollView>
        </StackLayout>
    </ContentPage.Content>
```

```

        </Grid>
    </ViewCell>
</DataTemplate>
</ListView.ItemTemplate>
</ListView>
</ScrollView>
<StackLayout Orientation="Horizontal" VerticalOptions="End">
    <Entry Text="{Binding WriteMessage.Message, Mode=TwoWay}"
Placeholder="Mensaje:" HorizontalOptions="FillAndExpand" Keyboard="Chat"/>
    <Button Text="Send" Command="{Binding SendMessageCommand, Mode=TwoWay}"
HorizontalOptions="End"/>
</StackLayout>
</StackLayout>
</ContentPage.Content>

```

Dentro de la propiedad content le asignamos el título de la siguiente forma y le agregamos un evento Appearing quedando de la siguiente forma el ContentPage.

```

<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
x:Class="Client_Chat.Views.ChatPage"
Title="{Binding Title}" Appearing="ChatPage_OnAppearing">

```

Después de haber agregado la vista nos vamos al archivo con terminación Page.xaml.cs, en el evento agregado le asignamos el siguiente código.

```

private void ChatPage_OnAppearing(object sender, EventArgs e)
{
    BindingContext = new VMChat();
}

```

## Paso 8 Últimos detalles

Vamos al Explorador de soluciones -> Nombre del proyecto -> App.xaml.cs. Modificamos la variable MainPage del constructor quedando de la siguiente forma.

```

public App ()
{
    InitializeComponent();

    MainPage = new NavigationPage(new ChatPage());
}

```

Ya una vez hecho todo esto el cliente queda terminado.

