

# NuGet Firebase Auth

Explicación para la utilización del NuGet de Firebase



# FirebaseAuthentication.net

4.0.2

**.NET Standard 2.0****.NET CLI**

Package Manager



PackageReference

Paket CLI

Script &amp; Interactive

Cake

```
> dotnet add package FirebaseAuthentication.net --version 4.0.2
```

 **README** Frameworks Dependencies Used By Versions

## FirebaseAuthentication.net



build passing



nuget v4.0.2



feedz.io 4.0.0-preview-20230205170144

Enlace del NuGet:

<https://www.nuget.org/packages/FirebaseAuthentication.net>

# Instalación ✓

Primeramente se deberá instalar el NuGet de Firebase a nuestro proyecto en Visual Studio.

## ¿Cómo instalarlo?

Nos dirigimos en el menú superior al apartado “Herramientas”, posteriormente localizamos y presionamos el apartado de “Administrador de paquetes NuGet”, después a “Administrar paquetes NuGet para la solución...”.

## Continuación...

Se nos abrirá una ventana emergente donde tenemos múltiples opciones, como lo serían: “Examinar”, “Instalado”, “Actualizaciones” y “Consolidar”. En este caso nosotros nos iremos a “Examinar”. Después de haber presionado en la barra de búsqueda ingresamos lo siguiente: “**FirebaseAuthentication.Net**” verificar que sea desarrollado por **Step Up Labs, Inc.**



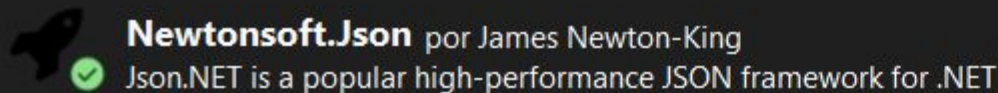
**FirebaseAuthentication.net** por Step Up Labs, Inc.  
Unofficial C# implementation of Firebase authentication targeting .NET Standard.

4.0.2

## Presionamos a Instalar la versión ‘latest’: 4.0.2

## Después de instalar...

Ya que se haya instalado, buscaremos un nuevo NuGet que sería “**Newtonsoft.Json**”, ya que este es dependencia para el funcionamiento del NuGet de Firebase.








13.0.3

De igual manera instalar la versión más reciente.

# ¡Listo! 😁

Si has seguido los pasos correctamente, deberás tener los NuGet instalados con éxito en tu proyecto. Para verificar que están instalados nos dirigimos a la pestaña de “Instalado” en nuestra ventana emergente de Administrar NuGets.

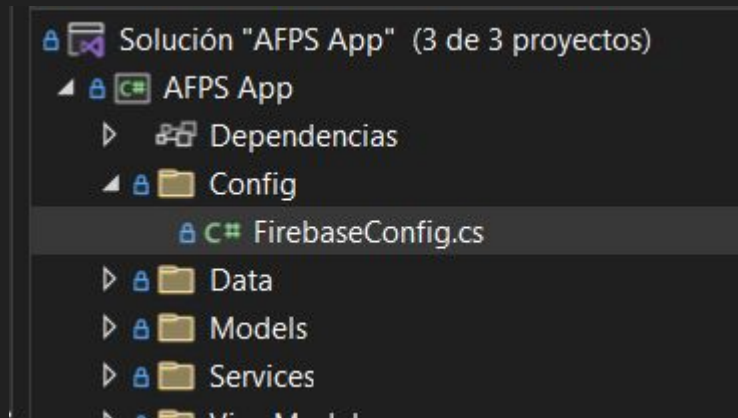
	<b>FirebaseAuthentication.net</b> por Step Up Labs, Inc. Unofficial C# implementation of Firebase authentication targeting .NET Standard.	4.0.2
	<b>NETStandard.Library</b> por Microsoft A set of standard .NET APIs that are prescribed to be used and supported together. 18a36291e48808fa7ef2d00a764ceb1ec95645a5	2.0.3
	<b>Newtonsoft.Json</b> por James Newton-King Json.NET is a popular high-performance JSON framework for .NET	13.0.3
	<b>Xamarin.Essentials</b> por Microsoft Xamarin.Essentials: a kit of essential API's for your apps	1.7.0 1.7.5
	<b>Xamarin.Forms</b> por Microsoft Build native UIs for iOS, Android, UWP, macOS, Tizen and many more from a single, shared C# codebase	5.0.0.2196 5.0.0.2578

# Hora del código

Empezaremos con el código y el consumo del NuGet

# Primeros pasos...

Debemos crear una carpeta en nuestra solución con el nombre de “**Config**” donde dentro crearemos una clase llamada “**FirestoreConfig.cs**”.



Una vez realizado lo anteriormente mencionado, debemos tenerlo de la manera presentada.



# Código de la clase

```
1  using Firebase.Auth;
2  using Firebase.Auth.Providers;
3
4  namespace AFPS_App.Config
5  {
6      public static class FirebaseConfig
7      {
8          public static string ApiKey = "AIzaSyCHiX[redacted]";
9          public static string AuthDomain = "afps-a[redacted]m";
10         public static FirebaseAuthProvider[] Providers = new FirebaseAuthProvider[]
11         {
12             new EmailProvider(),
13         };
14
15         public static FirebaseAuthConfig AuthConfig => new FirebaseAuthConfig
16         {
17             ApiKey = ApiKey,
18             AuthDomain = AuthDomain,
19             Providers = Providers
20         };
21     }
22 }
23
```

Importar librerías

Cadena que representa la clave API de Firebase

Dominio

Matriz de objetos 'FirebaseAuthProvider', que representan los proveedores disponibles

Proveedor

Miembro estático público para devolver el objeto que encapsula la configuración de FB

La clase es estática ('static') lo que significa que no puede ser instanciada, si no que se puede acceder a sus miembros directamente a través del nombre de la clase.

# ¿Cómo obtengo mis credenciales?

Muy fácil, nos dirigimos a nuestra consola de Firebase. En el menú lateral en el lado superior tenemos “Descripción del proyecto” y a lado tenemos un icono de “engranaje”, lo presionamos y nos dirigimos a “Configuración del proyecto”. Posteriormente hacemos scroll-down en el contenedor de “Tus apps”, visualizamos la configuración de SDK y en ese apartado estarían nuestras credenciales de nuestra aplicación.

```
const firebaseConfig = {
```

```
  apiKey: "AIza"
```

```
  authDomain: "
```

```
  databaseURL: "
```

```
  projectId: "a"
```

```
  storageBucket: "
```

```
  messagingSend
```

```
  appId: "1:164"
```

```
  measurementId
```

¡Listo! 😁

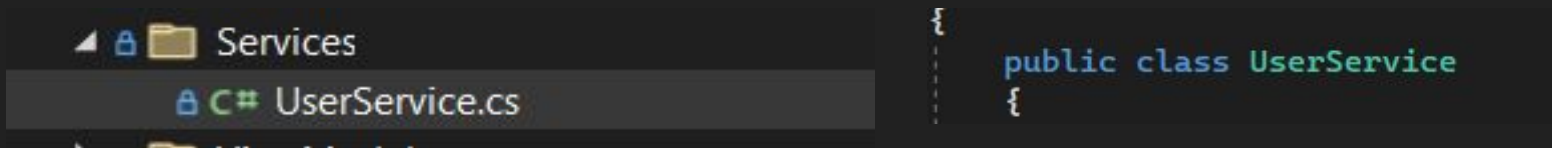
Si has seguido los pasos correctamente ya tendríamos listo nuestra clase de configuración con los datos de conexión con Firebase de nuestra aplicación.

# Creación del servicio

Ahora creamos un servicio especializado para el usuario, con este servicio podremos utilizarlo en todo nuestro proyecto.

## Pasos:

- Creamos una nueva carpeta en nuestra solución, llamada “Services”.
- Dentro de la carpeta “Services” creamos una clase llamada “UserService.cs”
- Declaramos nuestra clase como pública.



## Dentro del servicio...

Procederemos a declarar las siguientes variables en nuestra clase de “UserService.cs”:

```
private static FirebaseAuthClient _firebaseAuth;  
private UserCredential userCredential;  
private string token;
```

Tenemos tres variables privadas y una variable es estática que es “\_firebaseAuth”, que es una instancia de “FirebaseAuthClient”.

También tenemos UserCredential donde almacenaremos la credencial del usuario, lo mismo con el token.

## Continuación...

Procederemos a crear una propiedad pública que devuelva una instancia única de “FirebaseAuthClient” que se creará solo cuando se llama por primera vez.

```
public static FirebaseAuthClient FirebaseAuthInstance
{
    get
    {
        if (_firebaseAuth == null)
        {
            _firebaseAuth = new FirebaseAuthClient(FirebaseConfig.AuthConfig);
        }
        return _firebaseAuth;
    }
}
```

Annotations:

- Instance (points to `FirebaseAuthInstance`)
- getter (points to `get`)
- Comprobación (points to `if (_firebaseAuth == null)`)
- Si ya tiene valor, lo devuelve (points to `return _firebaseAuth;`)

## Método “Login”

Creamos un método “LoginAsync” que reciba dos parámetros: email y password. Utilizamos el objeto **FirebaseAuthInstance** para autenticar al usuario mediante el correo electrónico y la contraseña proporcionados para que devuelva un objeto **“UserCredential”** que contiene información sobre el usuario autenticado y un token de autenticación. Si la autenticación es exitosa, devuelve true. Si ocurre algún error, muestra una alerta con el mensaje de error y devuelve false.

A  
S  
I  
N  
C  
R  
O  
N  
O

```
public async Task<bool> LoginAsync(string email, string password)
{
    try
    {
        userCredential = await FirebaseAuthInstance.SignInWithEmailAndPasswordAsync(email, password);
        var user = userCredential.User;
        token = await user.GetIdTokenAsync();

        return true;
    }
    catch (Exception ex)
    {
        await Application.Current.MainPage.DisplayAlert("Alerta", ex.Message, "OK");
        return false;
    }
}
```

Parámetros

Utilizamos Try Catch para cachar el error en caso de que haya

Declara usuario

Método del NuGet

Almacena token

Retorna verdadero

Alerta de error

Prácticamente este método autentica al usuario con las credenciales proporcionadas utilizando **FirebaseAuthInstance**, obtiene información adicional del usuario a través del objeto **UserCredential**, almacena el token de autenticación en la variable token y devuelve un valor booleano que indica si la autenticación fue exitosa o no. Si se produce un error, se muestra una alerta en la página principal de la aplicación con el mensaje de error.



# Llamar a UserService y método

¡Perfecto! Una vez tenemos creado nuestro método procederemos a llamarlo en nuestro ViewModel para poder iniciar sesión.

## **Pasos:**

- En nuestro ViewModel de “Login” lo asignamos cómo asíncrono.
- Declaramos la variable userService llamando al servicio.
- Declaramos variable isLoggedIn llamando al método LoginAsync y asignamos los datos ingresados en la vista

```
#region PROCESOS
public async void OnLoginClicked(object obj)
{
    var userService = new UserService();
    var isLoggedIn = await userService.LoginAsync(Email, Contraseña);

    if (isLoggedIn)
    {
        await Shell.Current.GoToAsync($"///{nameof(PrincipalPage)}");

        var currentUser = await userService.GetCurrentUserAsync();
        await Application.Current.MainPage.DisplayAlert("Bienvenido", $"¡Hola {currentUser.Info.DisplayName}!", "OK");
    }
}
#endregion
```

Instancia

Llamada al método

Validación

Redireccionamiento

Este es un método se ejecuta cuando se hace clic en un botón de inicio de sesión. Crea una instancia de la clase `UserService` y llama al método `LoginAsync` para autenticar al usuario utilizando los valores de correo electrónico y contraseña almacenados en las variables `Email` y `Contraseña`.

Si la autenticación es exitosa, navega a la página principal de la aplicación utilizando `Shell.Current.GoToAsync`.

El método no devuelve ningún valor (utilizamos **async void**), lo que significa que no se puede esperar a que se complete la tarea de inicio de sesión antes de continuar con otras operaciones.

## Método “GetCurrentUser”

Crearemos un método que devuelve un objeto User que representa al usuario actualmente autenticado. Primero, intenta acceder a la información del usuario utilizando el objeto **UserCredential** almacenado en la variable userCredential. Luego, obtiene el identificador único del usuario y su nombre utilizando la propiedad User.Info.

Si se produce un error, se muestra una alerta en la página principal de la aplicación con el mensaje de error. El método devuelve un objeto User si la operación es exitosa y null si se produce un error.

***Es importante destacar que este método solo se puede llamar después de que el usuario se haya autenticado correctamente, ya que depende del objeto UserCredential almacenado en la variable userCredential.***

```
public async Task<User> GetCurrentUserAsync()
{
    try
    {
        var user = userCredential.User;
        var uid = user.Uid;
        var name = user.Info.DisplayName;

        return user;
    }
    catch (Exception ex)
    {
        await Application.Current.MainPage.DisplayAlert("Alerta", ex.Message, "OK");
        return null;
    }
}
```

El método devuelve una tarea que devuelve un objeto User, que representa al usuario autenticado en la aplicación. El método utiliza un objeto **UserCredential** para acceder a la información del usuario, como su UID y DisplayName. Utilizamos try-catch para manejar cualquier excepción que pueda ocurrir durante la operación. Es asíncrono, lo que significa que puede tardar un tiempo en obtener la información del usuario autenticado, por lo que devuelve un objeto Task<User>.

¡Listo! 😁

Creando este nuevo método, ya podemos concluir el inicio de sesión para que nos muestra el mensaje de bienvenida con el nombre del usuario. Recuerda que esto es solo un ejemplo para que veas como puedes llamar al método en tus otras clases donde requieras mostrar información del usuario, ya sea su nombre, identificador, correo electrónico y entre otras más.

```
var currentUser = await userService.GetCurrentUserAsync();  
await Application.Current.MainPage.DisplayAlert("Bienvenido", $"¡Hola {currentUser.Info.DisplayName}!", "OK");
```

# Repositorio para visualizar código

## Dirígete a mi repositorio:

- <https://www.github.com/thomasorzg/>



The screenshot shows a GitHub profile for a user named Thomas with the username thomasorzg. The profile includes a bio, a location of Sonora, México, and statistics showing 15 followers and 2 following. A prominent 'Edit profile' button is visible. To the right, a contribution graph is displayed, showing a single green square on Monday, May 1st, indicating one contribution. The graph covers the months of March, April, May, and June. Below the graph, a link to 'Learn how we count contributions' is provided.

**Thomas**  
thomasorzg

[Edit profile](#)

👤 15 followers · 2 following

📍 Sonora, México

Mar Apr May Jun

Mon ■

Wed

Fri

[Learn how we count contributions](#)

**¡Concluimos 😏!**

Con esto, tu aplicación tendrá éxito para autenticar usuarios correctamente.



firebase