Laboratorio

Configuración de una solución Xamarin multiplataforma

Versión: 1.0.0
Abril de 2017

Miguel Muñoz Serafín
@msmdotnet

CONTENIDO

Introducción

Un principio clave de la creación de aplicaciones multiplataforma es crear una arquitectura que maximice la compartición de código a través de las distintas plataformas. La adhesión a los siguientes principios de programación orientados a objetos ayuda a diseñar una buena arquitectura de una aplicación:

- **Encapsulación**. Garantiza que las clases e incluso las capas de la arquitectura sólo exponen una API mínima que realiza las funciones requeridas y oculta los detalles de la implementación. A nivel de clase, esto significa que los objetos se comportan como "cajas negras" y que el código que las consume no necesita saber cómo realizan sus tareas. A nivel de arquitectura, esto sugiere implementar patrones como ***Facade*** que fomenta una API simplificada que orquesta interacciones más complejas en lugar del código ubicado en las capas más abstractas. Por ejemplo, esto significa que el código de la interfaz de usuario sólo debe ser responsable de mostrar pantallas y aceptar la entrada de usuario, pero nunca debería interactuar con la base de datos directamente. Del mismo modo, el código de acceso a datos sólo debería leer y escribir en la base de datos, pero nunca interactuar directamente con botones o etiquetas de la interfaz de usuario.
- **Separación de responsabilidades**. Debemos asegurarnos de que cada componente (tanto a nivel de arquitectura como de clase) tenga un propósito claro y bien definido. Cada componente debe realizar sólo sus tareas definidas y exponer esa funcionalidad a través de una API que sea accesible a las otras clases que necesitan utilizarlo.
- **Polimorfismo**. La programación para una interface (o clase abstracta) que soporte múltiples implementaciones permite que el código núcleo pueda ser escrito y compartido entre plataformas al mismo tiempo que también pueda interactuar con características específicas de la plataforma.

Separar el código en capas hace que las aplicaciones sean más fáciles de entender, probar y mantener. Se recomienda que el código de cada capa esté físicamente separado (ya sea en directorios o incluso proyectos independientes para aplicaciones muy grandes), así como lógicamente separado (usando espacios de nombres).

Patrones de software comunes para desarrollo móvil

Los patrones son una forma establecida para aplicar soluciones recurrentes a problemas comunes. Existen algunos patrones clave que son útiles para entender la creación de aplicaciones móviles que puedan ser de fácil mantenimiento y de fácil entendimiento.

- **Model, View, ViewModel (MVVM)**. El patrón Model-View-ViewModel es muy popular con los Frameworks que soportan la vinculación de datos (data-binding), tal como

Xamarin.Forms. Fue popularizado por SDKs habilitados para XAML como Windows Presentation Foundation (WPF) y Silverlight donde el ViewModel actúa como un intermediario entre los datos (Model) y la interfaz de usuario (View) a través de enlace de datos y comandos.

- **Model, View, Controller (MVC)**. Un patrón común y a menudo incomprendido, MVC se utiliza con mayor frecuencia cuando se crean interfaces de usuario y se requiere una separación entre la definición de una pantalla de interfaz de usuario (View), el motor detrás de él que maneja la interacción (Controller) y los datos que lo alimentan (Model). El modelo es una pieza opcional y, por lo tanto, el núcleo de la comprensión de este patrón se encuentra en la Vista y el Controlador. MVC es un enfoque popular para aplicaciones iOS.
- **Business Facade**. También conocido como **Manager Pattern**, proporciona un punto de entrada simplificado para trabajos complejos. Por ejemplo, en una aplicación de seguimiento de tareas, podríamos tener una clase *TaskManager* con métodos como *GetAllTasks()*, *GetTask(taskID*), SaveTask(task), etc. La clase TaskManager proporciona una fachada al funcionamiento interno de guardar/recuperar objetos que representan las tareas.
- **Singleton**. El patrón **Singleton** proporciona una forma en la que sólo una sola instancia de un objeto en particular puede existir. Por ejemplo, al utilizar SQLite en aplicaciones móviles, sólo desearíamos tener una única instancia de la base de datos. El uso del patrón **Singleton** es una forma sencilla de garantizar esto.
- **Provider**. Un patrón acuñado por Microsoft (posiblemente similar al patrón **Strategy**, o al patrón **Dependency Injection**) para fomentar la reutilización de código a través de aplicaciones Silverlight, WPF y WinForms. El código compartido se puede escribir implementando una interface o una clase abstracta, y las implementaciones concretas específicas de la plataforma se escriben y pasan cuando se usa el código.
- **Async**. No se debe confundir con la palabra clave *Async*. El patrón **Async** se utiliza cuando se necesita ejecutar un trabajo de larga ejecución sin congelar la interfaz de usuario o el procesamiento actual. En su forma más simple, el patrón Async simplemente describe que las tareas de larga duración deberían iniciarse en otro subproceso (Thread) mientras el subproceso actual sigue procesando y esperando una respuesta del proceso en segundo plano para posteriormente actualizar la interfaz de usuario cuando se devuelven datos y/o estado.

En este laboratorio crearás la arquitectura típica de una solución Xamarin con Visual Studio. La arquitectura que crearás es solo un ejemplo de la forma en que puede ser estructurada una aplicación Xamarin multiplataforma, sin embargo, la arquitectura en una aplicación real puede variar dependiendo de los requerimientos de la aplicación.

Objetivos

Al finalizar este laboratorio, los participantes serán capaces de:

- Describir las capas típicas de una aplicación Xamarin multiplataforma.
- Describir la configuración típica de una solución Xamarin multiplataforma.

Requisitos

Para la realización de este laboratorio es necesario contar con lo siguiente:

- Un equipo de desarrollo con Visual Studio. Los pasos descritos en este laboratorio fueron diseñados con Visual Studio Enterprise 2017 y Windows 10 Professional.
- Xamarin para Visual Studio.

Tiempo estimado para completar este laboratorio: **60 minutos**.

Ejercicio 1: Configurando una solución Xamarin multiplataforma

Independientemente de qué plataformas se utilicen, los proyectos Xamarin utilizan el mismo formato de archivo de solución (el formato de archivo .sln de Visual Studio). Las soluciones se pueden compartir en entornos de desarrollo multiplataforma, incluso cuando no se puedan cargar proyectos individuales (como un proyecto de Windows en Xamarin Studio).

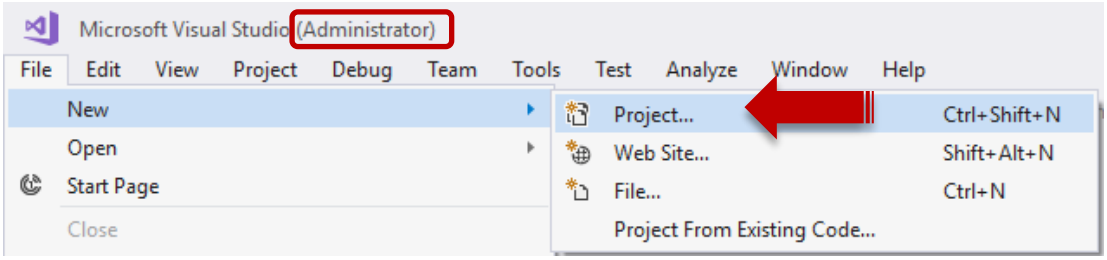
En este ejercicio realizarás las tareas comunes de configuración de una solución Xamarin multiplataforma.

Tarea 1. Crear una solución Visual Studio vacía.

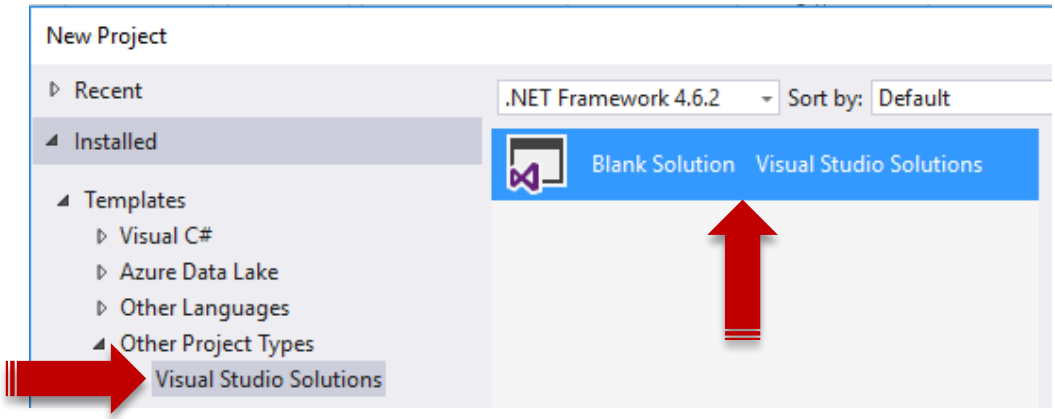
Al crear una nueva aplicación multiplataforma, el primer paso podría ser la creación de una solución en blanco y después la configuración de los proyectos para la aplicación.

Realiza los siguientes pasos para crear una solución en blanco desde Visual Studio.

1. Abre Visual Studio en el contexto del Administrador.
2. Selecciona la opción **File** > **New** > **Project**.

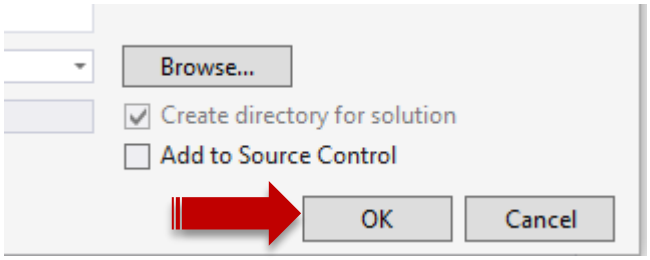


3. En la ventana **New Project** selecciona la plantilla **Blank Solution**.

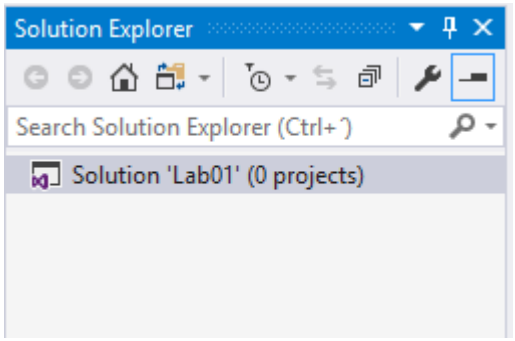


4. Proporciona el nombre y ubicación de la solución.

5. Haz clic en **OK** para crear la solución en blanco.



El explorador de soluciones de Visual Studio mostrará la solución vacía.



Tarea 2. Agregar proyectos a la solución.

Parte del proceso de configuración de la solución Xamarin multiplataforma es la estrategia a seguir para compartir código entre los distintos proyectos de la aplicación y para separar el código en capas para que sean más fáciles de entender, probar y mantener.

Compartir código

Con Xamarin, tenemos 3 métodos alternativos para compartir código en aplicaciones multiplataforma:

- **Proyectos Compartidos (Shared Projects).** El método más sencillo para compartir archivos de código es utilizar un proyecto compartido. Este método nos permite compartir el mismo

código fuente en diferentes proyectos de plataforma y utilizar directivas de compilación para incluir diferentes rutas de código específicas de plataforma.

- **Biblioteca de Clases Portable (Portable Class Libraries o PCL)**. Históricamente un archivo de proyecto .NET (y el ensamblado resultante) es compilado para una versión de Framework específica. Esto impide que el proyecto o el ensamblado se compartan en diferentes Frameworks.

  Una biblioteca de clases portáble (PCL) es un tipo especial de proyecto que puede utilizarse en diferentes plataformas CLI como Xamarin.iOS y Xamarin.Android, así como WPF, Universal Windows Platform y Xbox. La biblioteca sólo puede utilizar un subconjunto del .NET Framework completo, limitado por las plataformas a las que se estén dirigiendo.
- **.NET Standard**. Introducidos en 2016, los proyectos .NET Standard proporcionan una forma sencilla de compartir código entre plataformas, produciendo ensamblados que se pueden utilizar en Windows, plataformas Xamarin (iOS, Android, Mac) y Linux. Las bibliotecas .NET Standard se pueden crear y utilizar como PCLs.

Capas típicas en aplicaciones Xamarin

Separar el código en capas hace que las aplicaciones sean más fáciles de entender, probar y mantener. Se recomienda que el código de cada capa esté físicamente separado (ya sea en directorios o incluso proyectos independientes para aplicaciones muy grandes), así como lógicamente separado (usando espacios de nombres).

Las siguientes son las seis capas más comunes en que una aplicación Xamarin suele dividirse:

- **Capa de datos (Data Layer)**. Es la capa en la que se realiza la persistencia de datos no volátil. Puede ser por ejemplo una base de datos *SQLite*, o podría implementarse con archivos XML o cualquier otro mecanismo adecuado.
- **Capa de acceso a datos (Data Access Layer)**. Es la capa que envuelve a la capa de datos y que proporciona la funcionalidad de Crear, Leer, Actualizar y Eliminar datos (operaciones CRUD) sin exponer detalles de la implementación al invocador. Por ejemplo, esta capa puede contener instrucciones SQL para consultar o actualizar los datos, pero el código que lo utiliza no necesita saberlo.
- **Capa de negocio (Business Layer)**. Esta capa es a veces llamada Business Logic Layer o BLL. Contiene definiciones de entidad de negocio (el Modelo) y lógica de negocio. Esta es una capa candidata para implementar el patrón *Facade*.
- **Capa de acceso a servicio (Service Access Layer).** Esta capa se utiliza para acceder a servicios en la nube, desde servicios web complejos (REST, JSON, WCF) hasta la simple recuperación de datos e imágenes de servidores remotos. Esta capa encapsula el comportamiento de red y proporciona una API sencilla para ser consumida por las capas de Aplicación e Interfaz de usuario.

- **Capa de aplicación (Application Layer)**. Esta capa contiene típicamente código específico de la plataforma (no compartido generalmente entre plataformas) o código que es específico de la aplicación (generalmente no reutilizable). Una buena prueba para saber si debemos colocar código en la capa de aplicación o en la capa de interfaz de usuario es determinar si la clase contiene controles de pantalla o si podría compartirse entre varias pantallas o dispositivos (por ejemplo, iPhone y iPad).
- **Capa de interfaz de usuario (User Interface (UI) Layer).** Es la capa orientada al usuario, contiene pantallas, widgets y los controladores que los gestionan.

Es posible que una aplicación no contenga necesariamente todas las capas, por ejemplo, la capa de acceso a servicio no existiría en una aplicación que no tenga acceso a los recursos de la red. Una aplicación muy simple podría fusionar la Capa de datos y la Capa de acceso a datos, así como la Capa de aplicación y la Capa de interfaz de usuario porque las operaciones son extremadamente básicas.

Independientemente de qué método se utilice para compartir código, la estructura de la solución debe implementar una arquitectura en capas que fomente el uso compartido de código. El enfoque de Xamarin es agrupar el código en dos tipos de proyectos: **Proyecto núcleo** y **Proyectos específicos de plataforma**.

Proyecto núcleo

Es recomendable escribir código reutilizable en un solo lugar para ser compartido entre las diferentes plataformas. Siempre que sea posible, debemos aplicar los principios de encapsulación para ocultar los detalles de implementación.

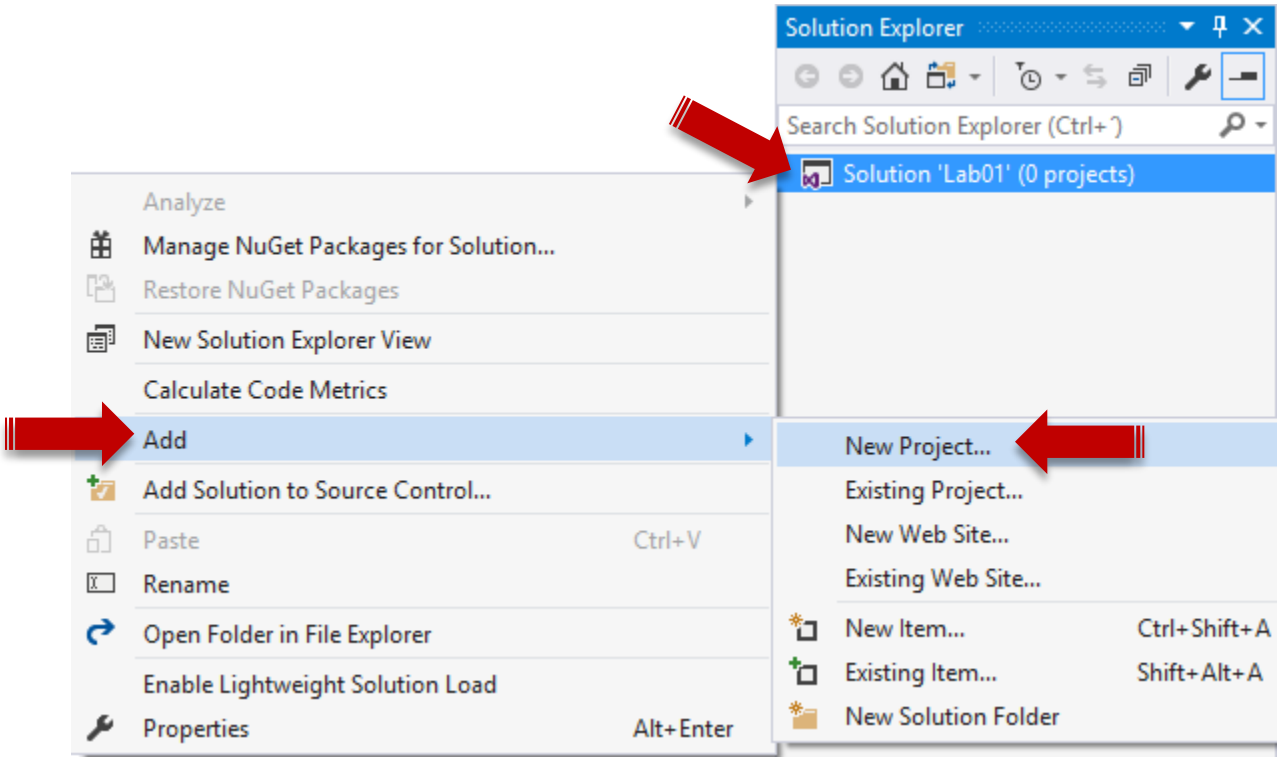
Los proyectos de código compartido sólo deben hacer referencia a ensamblados que estén disponibles en todas las plataformas, es decir, los espacios de nombres comunes del .NET Framework como *System*, *System.Core* y *System.Xml*.

Tanto como sea posible, los proyectos compartidos deben implementar la mayor parte de las funcionalidades que no sean de la interfaz de usuario. Esto podría incluir las siguientes capas:

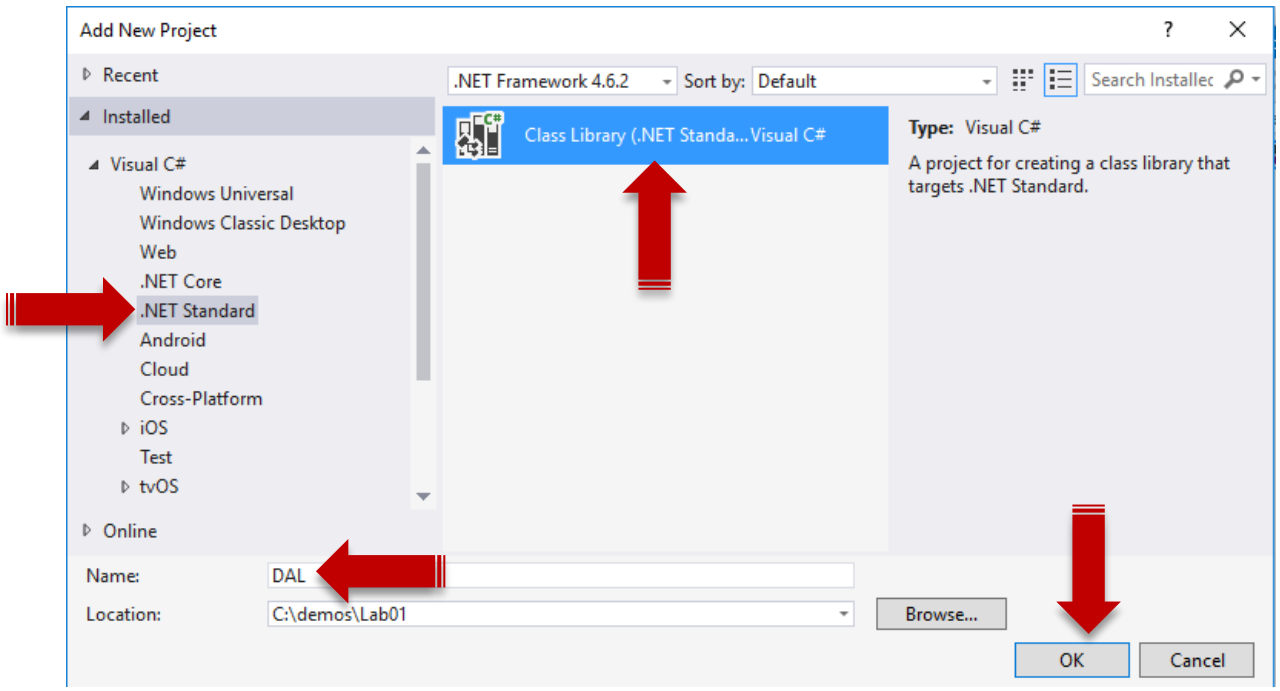
- Capa de datos.
- Capa de acceso a datos.
- Capa de acceso a servicio.
- Capa de negocio.

Realiza los siguientes pasos para agregar a la solución los proyectos núcleo que implementen las capas de Acceso a Datos, Negocio y Acceso a servicio.

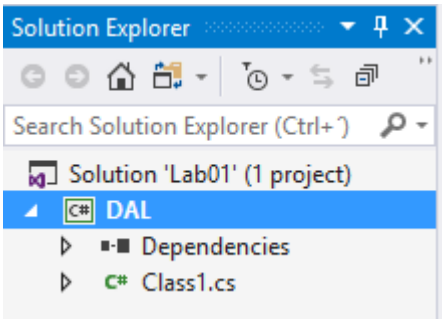
1. Selecciona la opción **Add** > **New Project…** del menú contextual de la solución.



2. En la ventana **Add New Project** selecciona la plantilla **Class Library (.NET Standard)**, asigna el nombre **DAL** y haz clic en **OK** para agregar el proyecto a la solución.



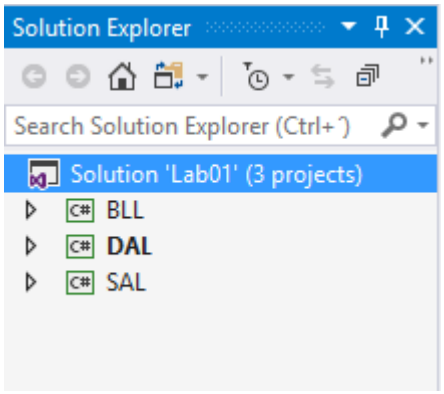
3. El explorador de soluciones mostrará el proyecto agregado.



4. Repite los pasos 1 y 2 para agregar los siguientes proyectos núcleo.

Nombre del proyecto	Descripción
SAL	Proyecto para implementar la capa de acceso a servicio.
BLL	Proyecto para implementar la capa de negocio.

Después de agregar los proyectos, el explorador de soluciones de Visual Studio tendrá la siguiente apariencia.



Proyectos específicos de plataforma

Es recomendable consumir el código reutilizable con el acoplamiento más pequeño posible. Las características específicas de la plataforma se añaden a este nivel, basadas en los componentes expuestos en el proyecto núcleo.

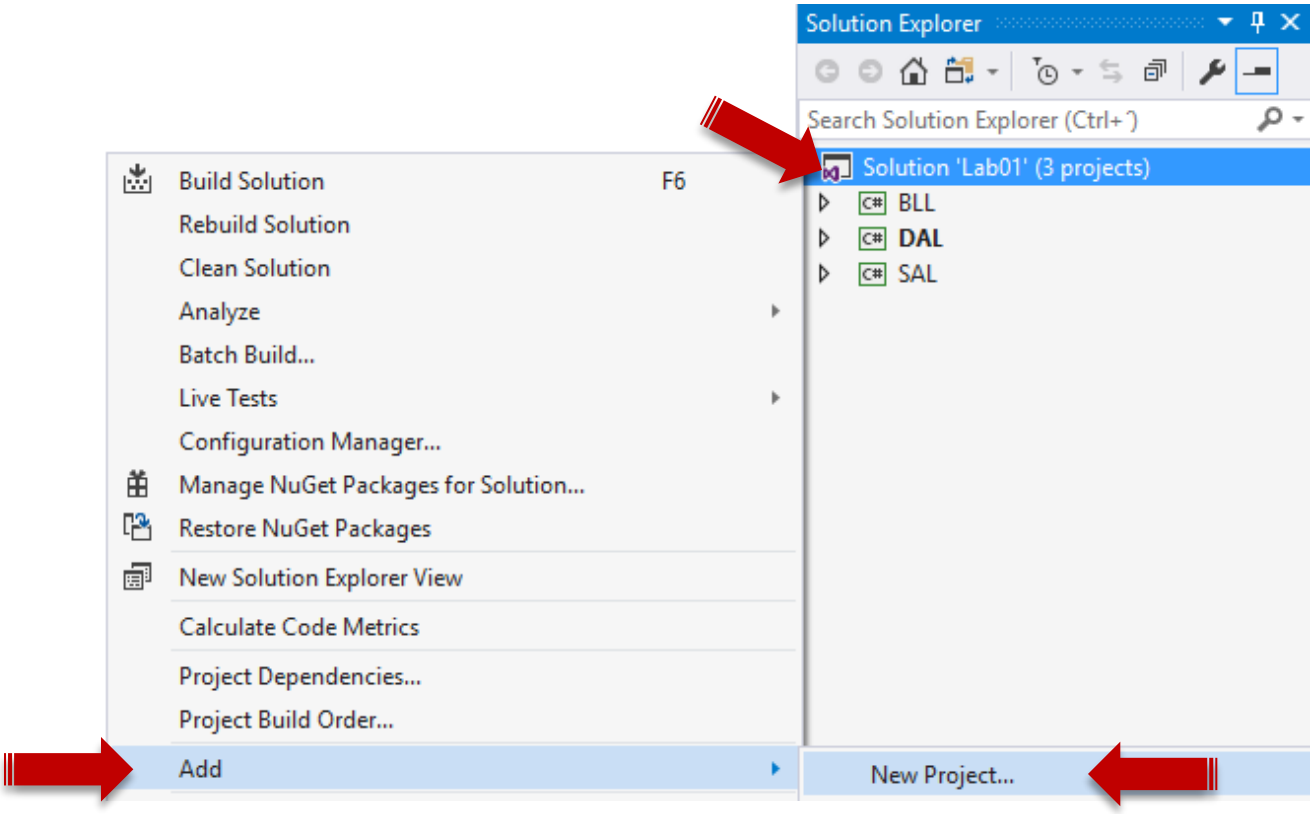
Los proyectos específicos de la plataforma deben hacer referencia a los ensamblados necesarios para enlazar a cada SDK de plataforma (Xamarin.iOS, Xamarin.Android, Xamarin.Mac o Windows), así como al proyecto de código núcleo compartido.

Los proyectos específicos de plataforma deben implementar las siguientes capas:

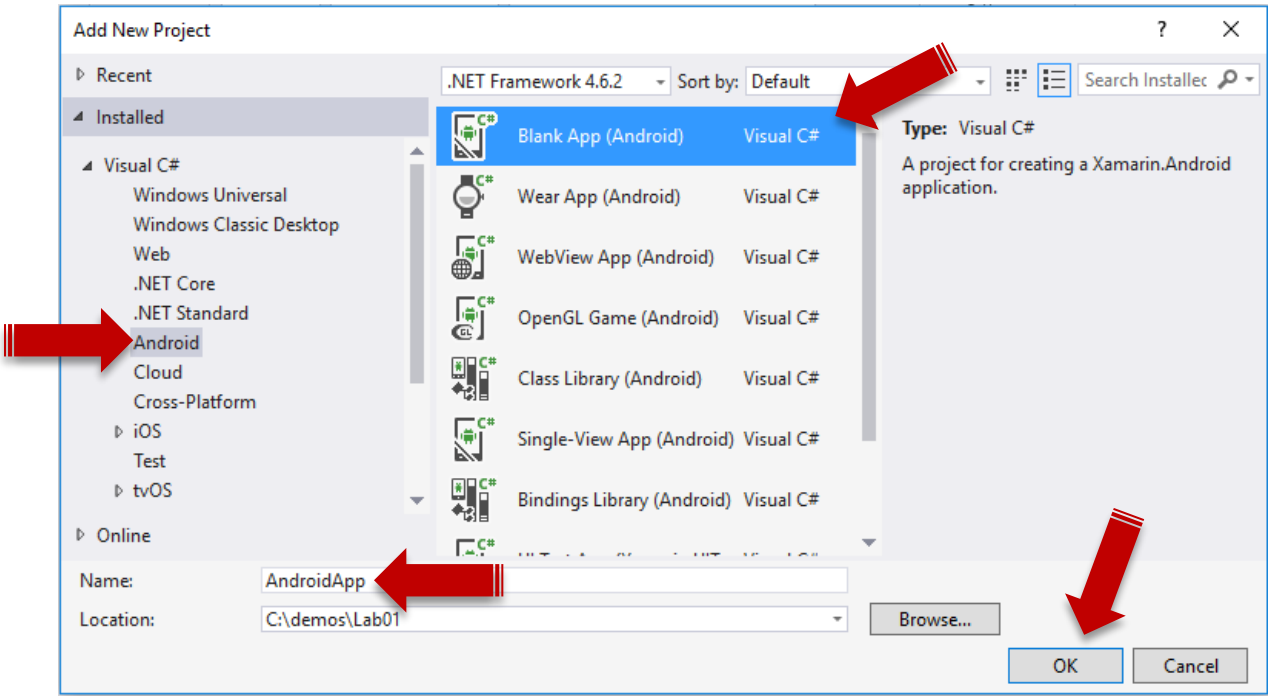
- Capa de aplicación. Para implementar funcionalidad específica de la plataforma y el enlace y conversión entre objetos BLL y la interfaz de usuario.
- Capa de interfaz de usuario. Para implementar pantallas, controles personalizados de interfaz de usuario y presentación de la lógica de validación.

Realiza los siguientes pasos para agregar a la solución los proyectos específicos para las plataformas Android y iOS.

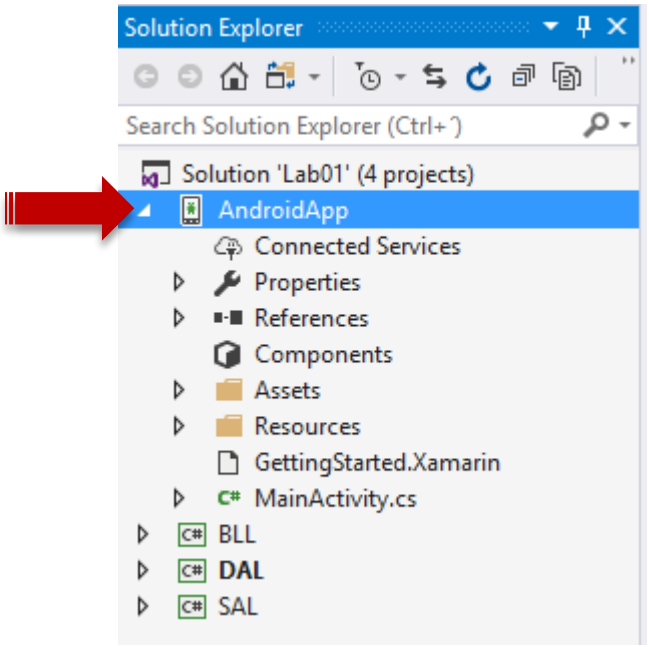
5. Selecciona la opción **Add** > **New Project…** del menú contextual de la solución.



6. En la ventana **Add New Project** selecciona la plantilla **Blank App (Android)**, asigna el nombre **AndroidApp** y haz clic en **OK** para agregar a la solución el proyecto que implementará la Capa de aplicación e Interfaz de usuario para la plataforma Android.

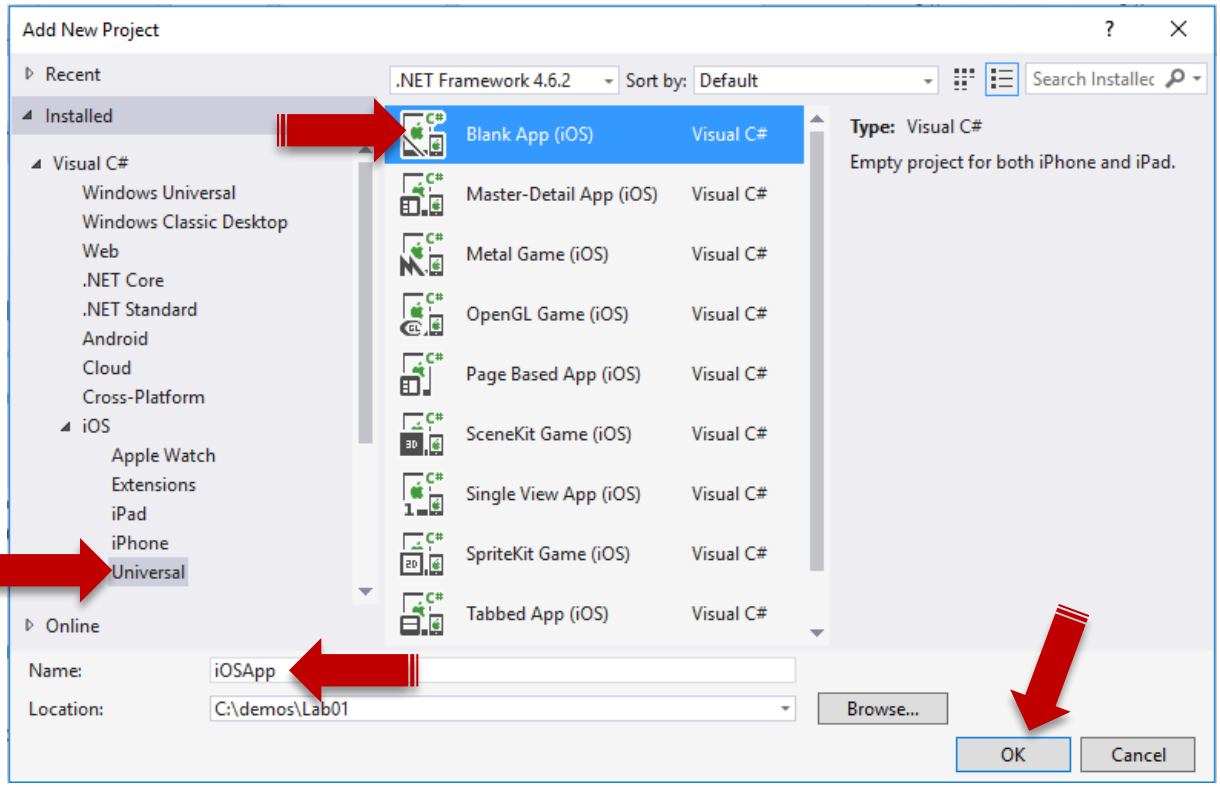


El explorador de soluciones de Visual Studio mostrará el nuevo proyecto agregado.

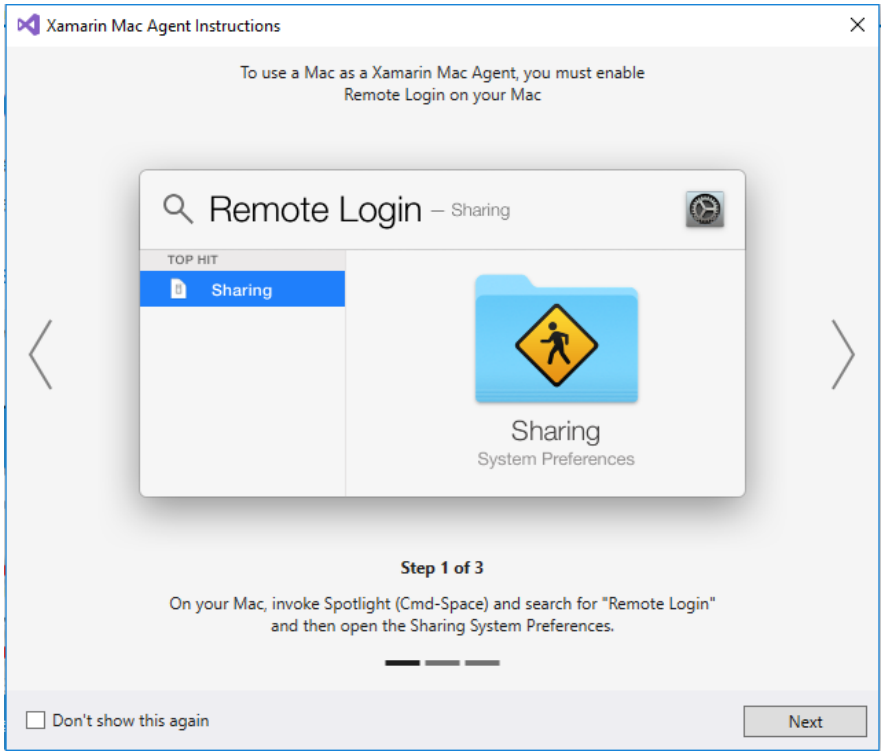


7. Selecciona la opción **Add** > **New Project…** del menú contextual de la solución.

8. En la ventana **Add New Project** selecciona la plantilla **Blank App (iOS)**, asigna el nombre **iOSApp** y haz clic en **OK** para agregar a la solución el proyecto que implementará la Capa de aplicación e Interfaz de usuario para la plataforma iOS.

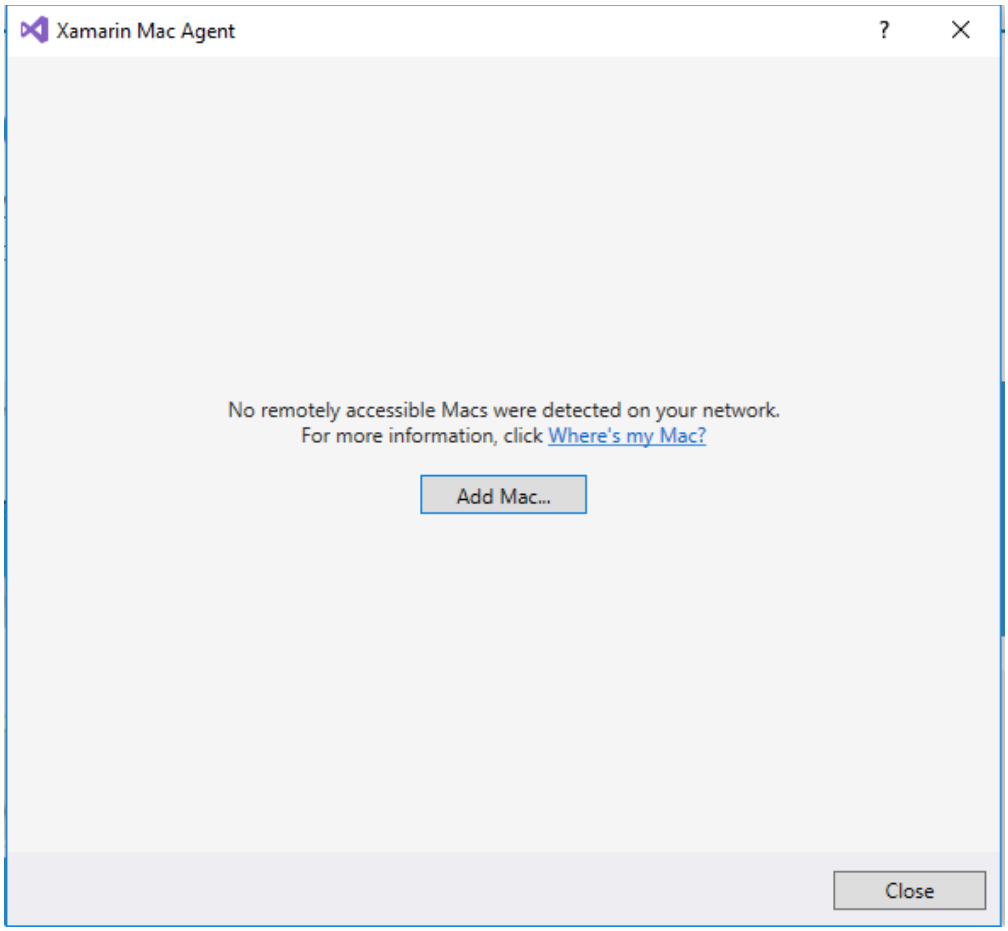


Visual Studio mostrará la ventana **Xamarin Mac Agent Instructions** con las instrucciones para poder utilizar la Mac necesaria para el desarrollo de aplicaciones para la plataforma iOS.



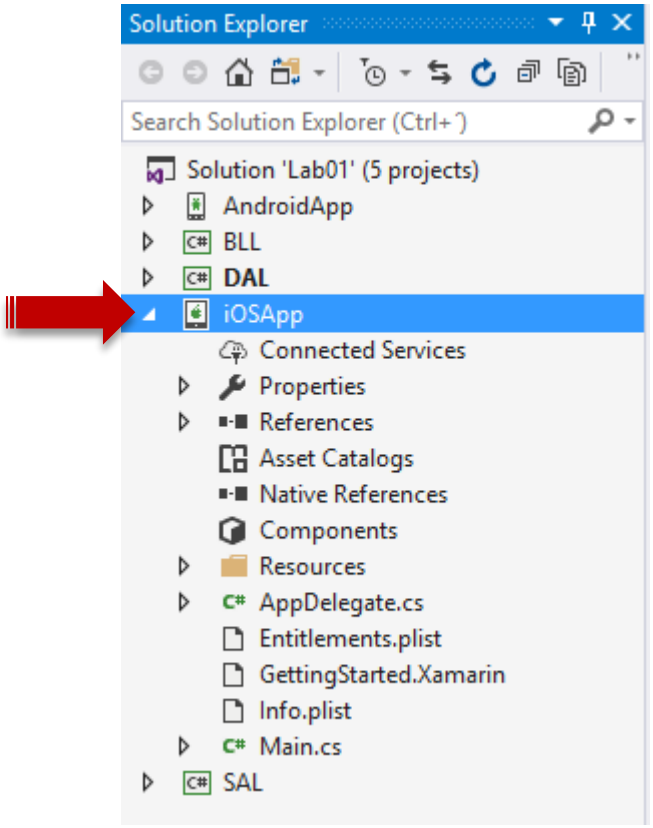
9. Cierra la ventana **Xamarin Mac Agent Instructions.**

   Visual Studio mostrará la ventana **Xamarin Mac Agent** para seleccionar la Mac.



10. Cierra la ventana **Xamarin Mac Agent**.

    El explorador de soluciones de Visual Studio mostrará el nuevo proyecto agregado.

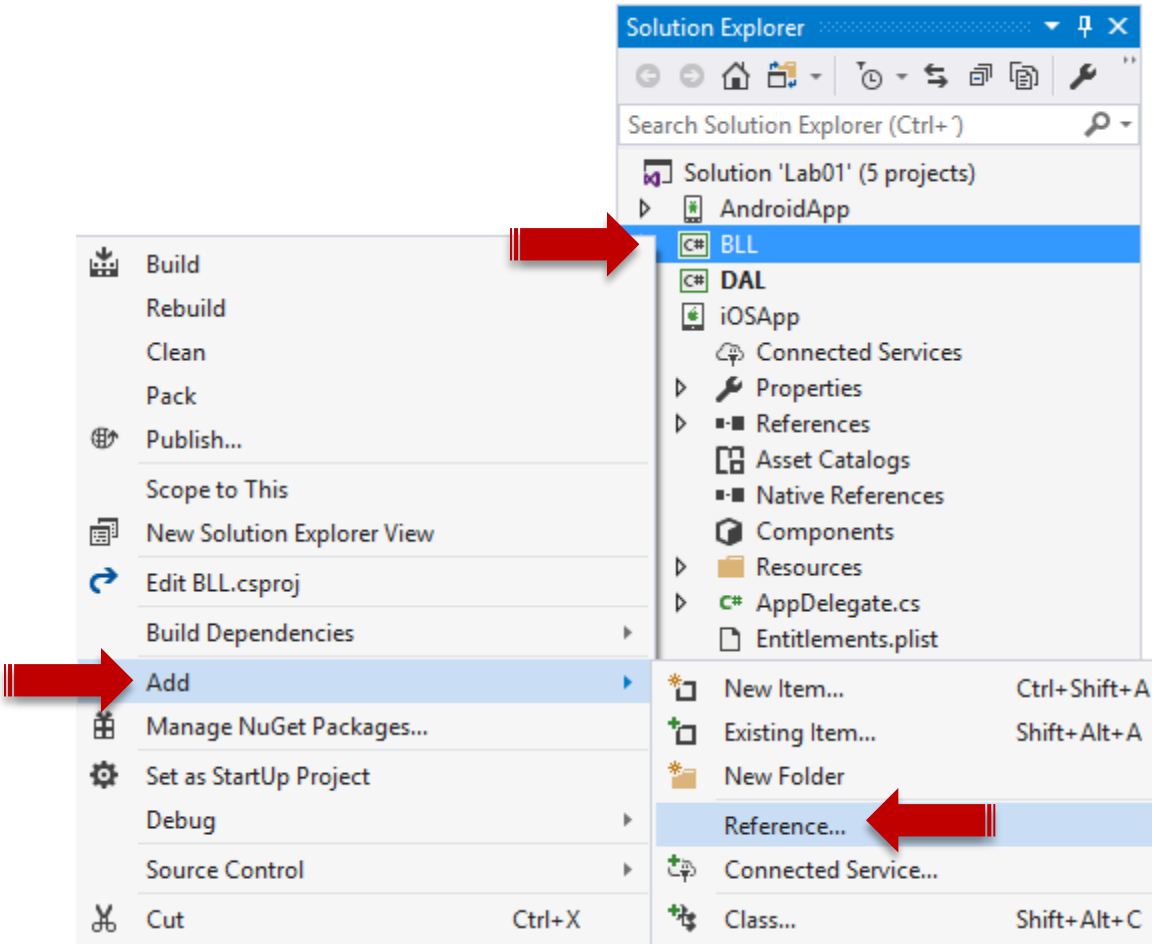


Tarea 3. Agregar referencias en los proyectos.

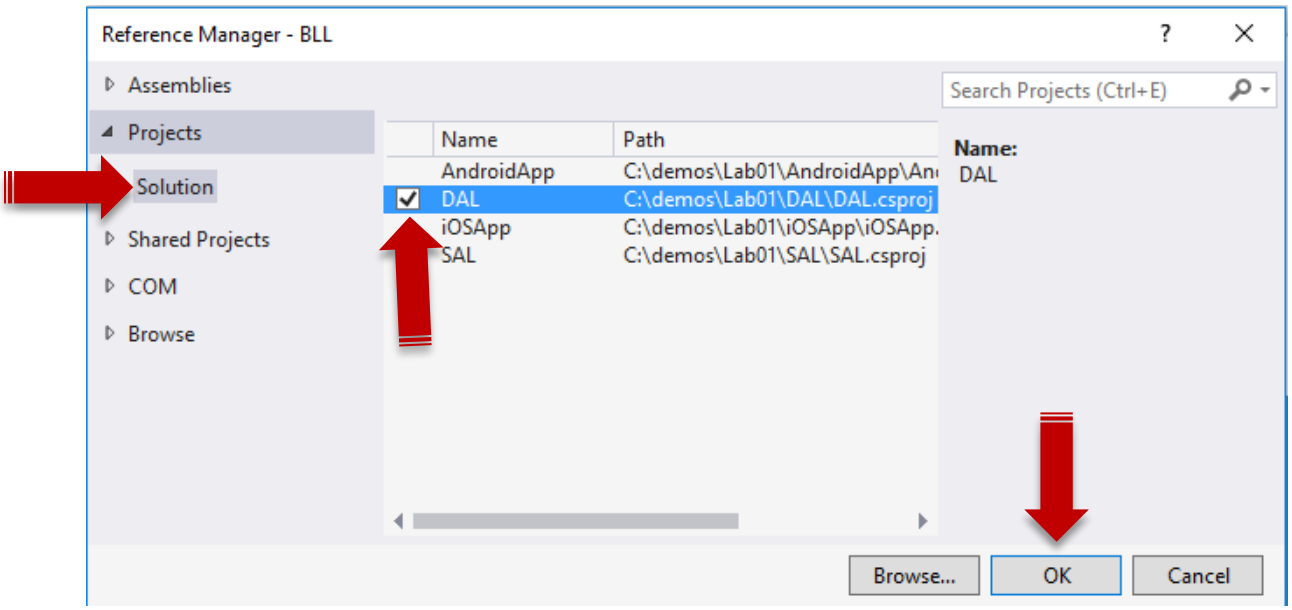
Las referencias de un proyecto reflejan sus dependencias. Los proyectos núcleo limitan sus referencias a ensamblados comunes para que el código sea fácil de compartir. Los proyectos específicos de plataforma hacen referencia al código compartido, además de cualquier otro ensamblado específico de la plataforma que necesiten para aprovechar la plataforma destino.

El proyecto **BLL** depende del proyecto **DAL** ya que este le proporciona los datos requeridos por la aplicación, por lo tanto, es necesario agregar en el proyecto **BLL** una referencia al proyecto **DAL**.

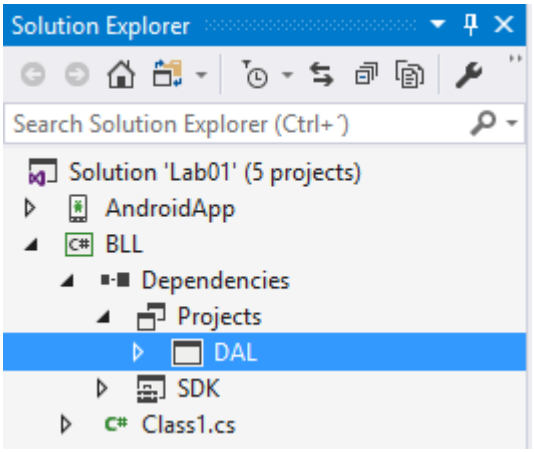
1. Selecciona la opción **Add** > **Reference…** del menú contextual del proyecto **BLL**.



2. En la ventana **Reference Manager – BLL**, selecciona el proyecto **DAL** y haz clic en **OK** para agregar la referencia.



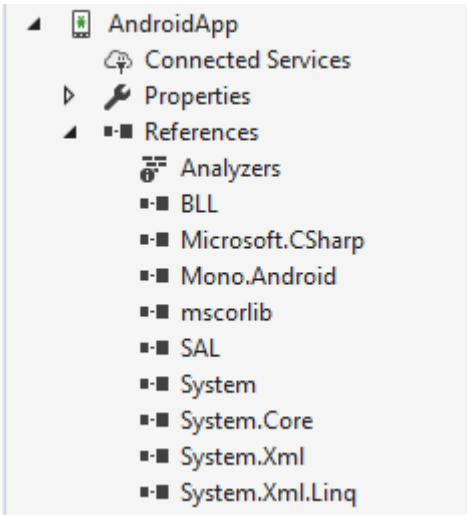
En la sección **Dependencies** > **Projects** del proyecto **BLL** puedes ver la referencia agregada.

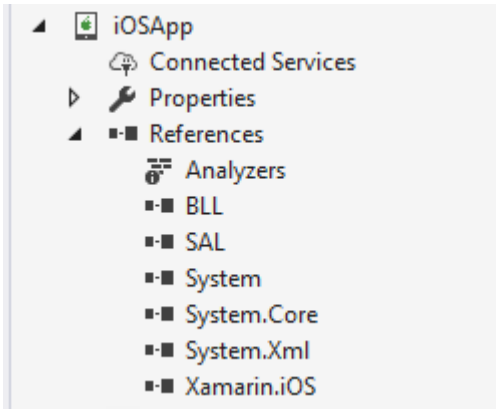


3. Realiza los pasos anteriores para agregar las siguientes referencias.

Proyecto	Dependencia
AndroidApp	BLL
AndroidApp	SAL
iOSApp	BLL
iOSApp	SAL

La siguiente imagen muestra las dependencias agregadas a los proyectos específicos de plataforma.



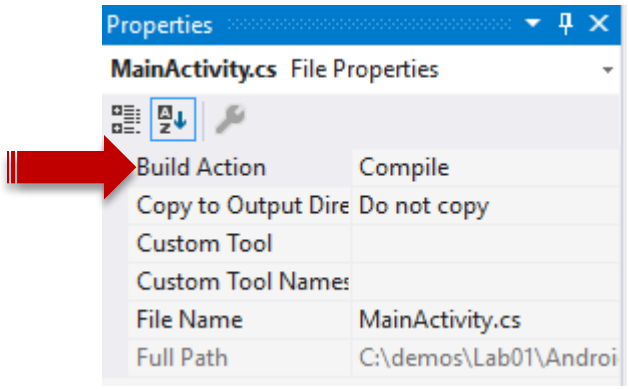


Tarea 4. Examinar la acción de compilación en los archivos agregados.

Es importante establecer la acción de compilación correcta para ciertos tipos de archivos. Veamos la acción de compilación para algunos tipos de archivo comunes.

1. Selecciona el archivo **MainActivity.cs** dentro del proyecto **AndroidApp**.

2. Presiona **F4** para mostrar la ventana de propiedades del archivo seleccionado.



Puedes notar que la acción de compilación para todos los archivos C# es **Compile**.

La siguiente tabla muestra la acción de compilación para algunos tipos de archivo comunes:

Tipo de archivo	Acción de compilación
Todos los archivos C#	Compile
Imágenes en Xamarin.iOS y Windows	Content
Archivos XIB y Storyboard en Xamarin.iOS	InterfaceDefinition
Imágenes y diseños AXML en Android	AndroidResource
Archivos XAML en proyectos Windows	Page
Archivos XAML de Xamarin.Forms	EmbeddedResource

Generalmente, el IDE detectará el tipo de archivo y sugerirá la acción de compilación correcta.

Sensibilidad de mayúsculas y minúsculas

Por último, debemos recordar que algunas plataformas tienen sistemas de archivos con distinción entre mayúsculas y minúsculas (por ejemplo, iOS y Android), por lo tanto, es importante asegurarnos de utilizar un estándar de nombres de archivos consistente y asegurarnos de que los nombres de archivo que utilizamos en código coincidan exactamente con el sistema de archivos. Esto es especialmente importante para las imágenes y otros recursos a los que se hace referencia en el código.

Ejercicio 2: Validando tu actividad

En este ejercicio agregarás funcionalidad a tu laboratorio con el único propósito de enviar una evidencia de la realización del mismo.

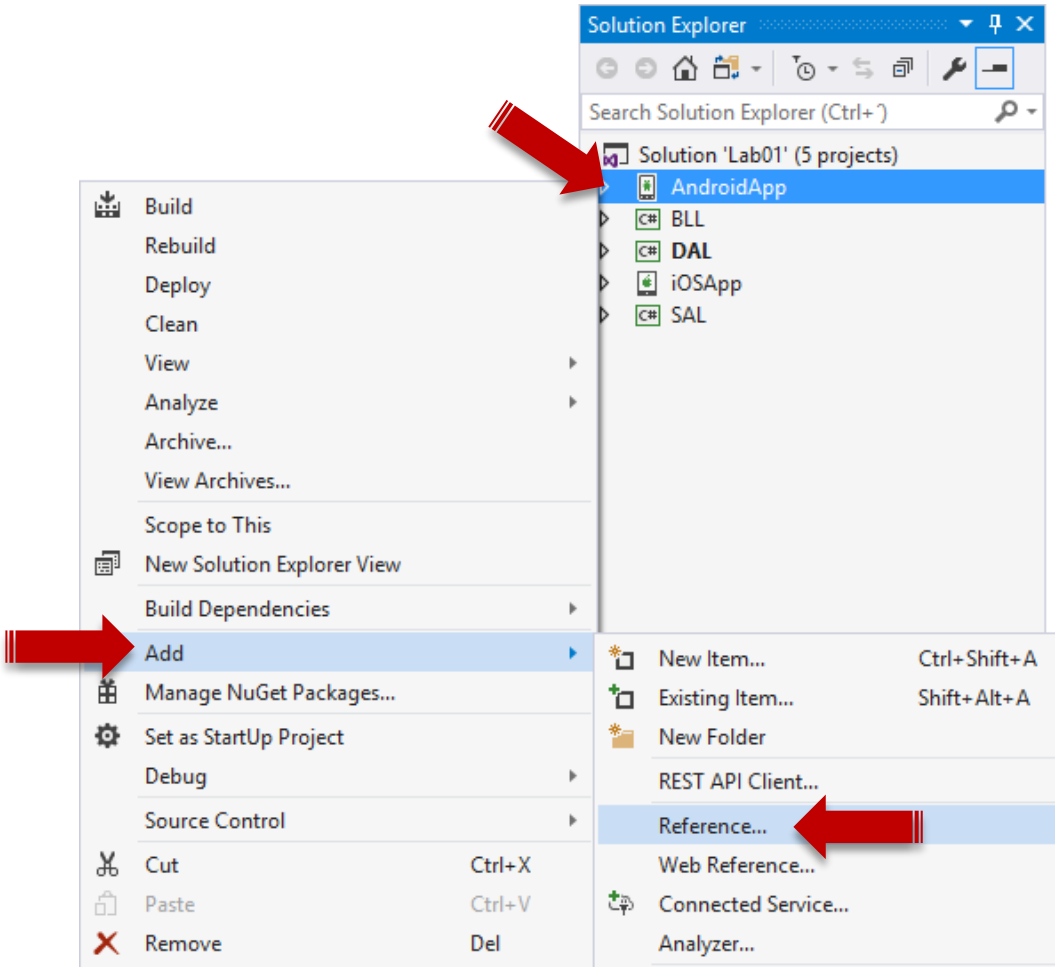
La funcionalidad que agregarás consumirá un ensamblado que representa una Capa de acceso a servicio (SAL) que será consumida por tu aplicación Android.

Es importante que realices cada laboratorio del diplomado ya que esto te dará derecho a obtener el diploma final del mismo.

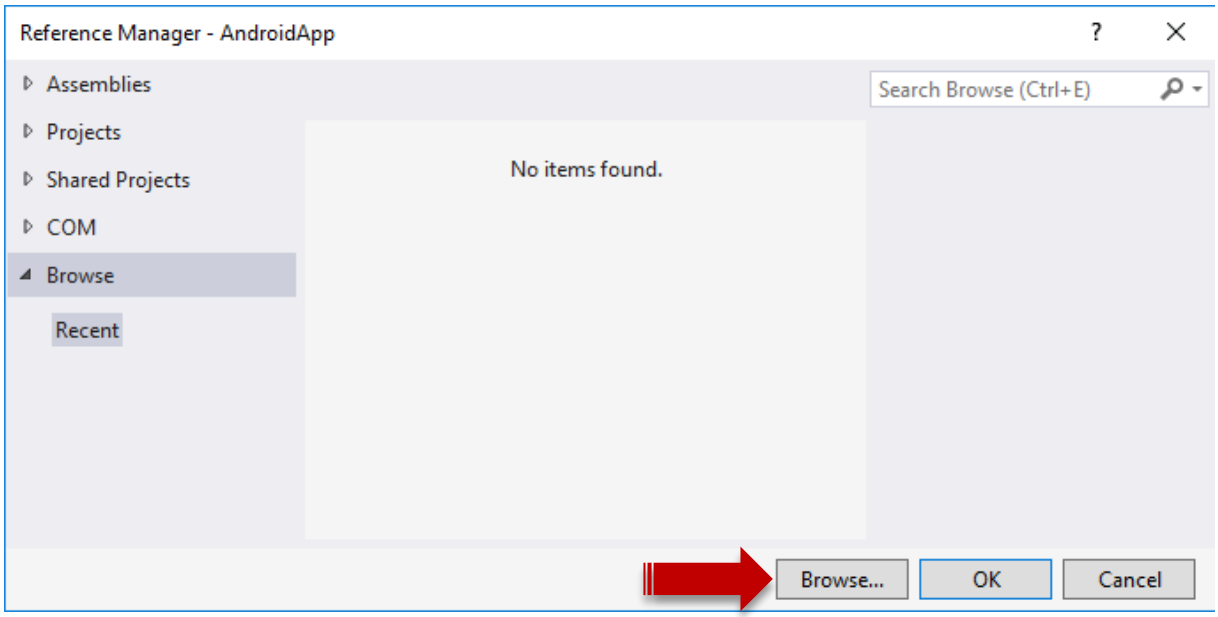
Tarea 1. Agregar los componentes de la Capa de acceso a Servicio.

En esta tarea agregarás una referencia al ensamblado **SALLab02.dll** que implementa la capa de acceso a servicio. El archivo **SALLab02.dll** se encuentra disponible junto con este documento.

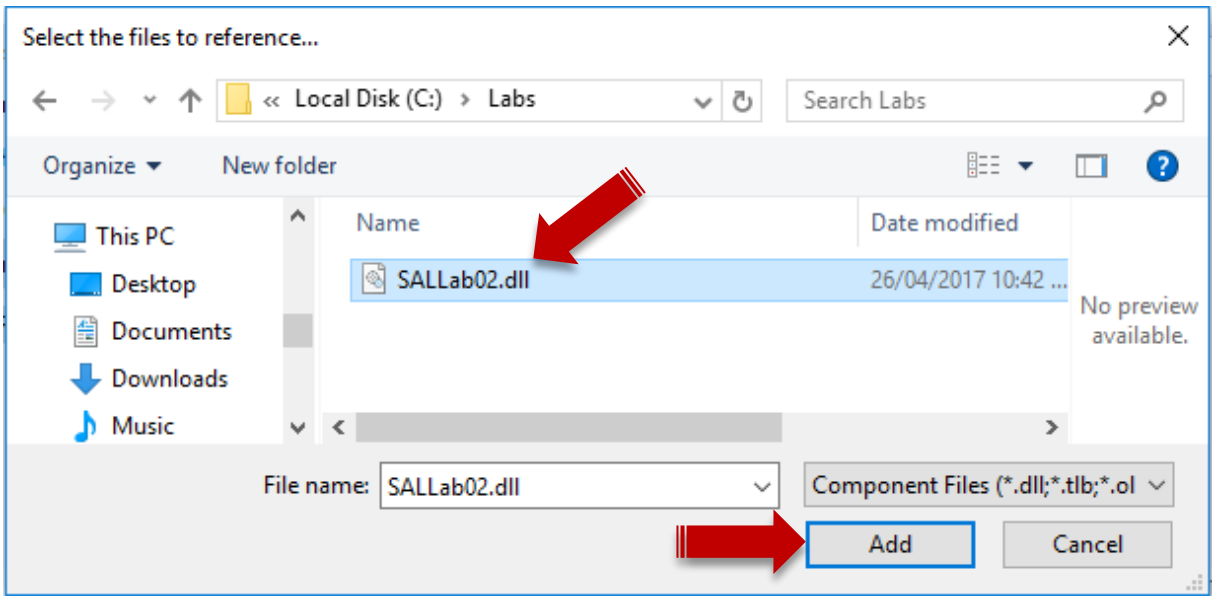
1. Accede a la opción **Add** > **Reference…** del menú contextual de la aplicación Android.



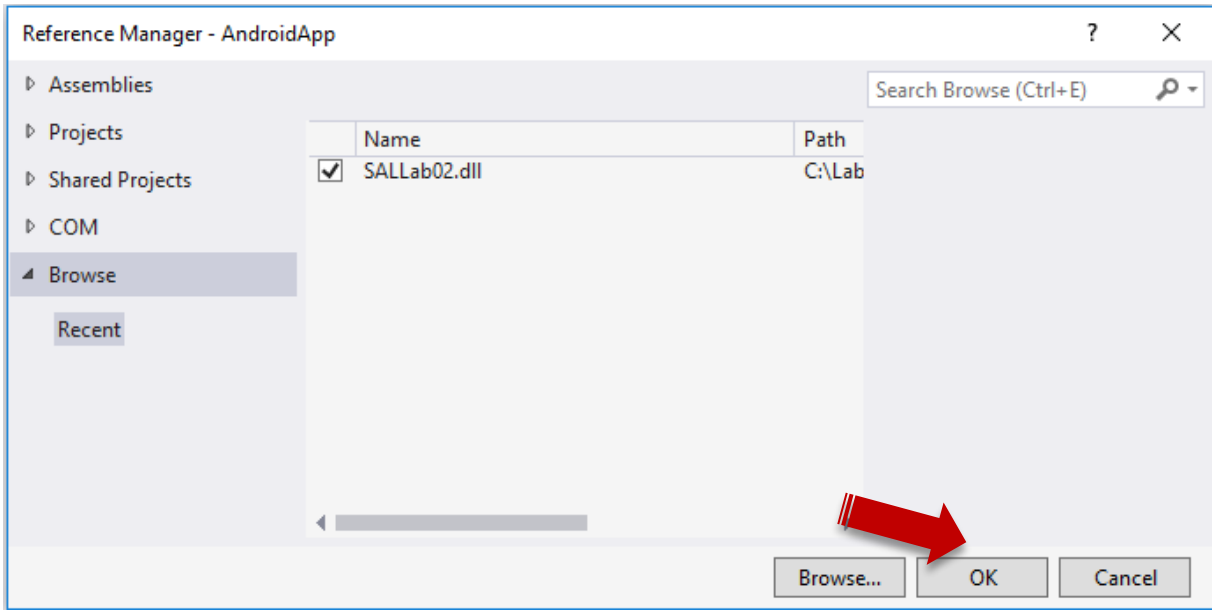
2. En la ventana **Reference Manager** haz clic en **Browse…** para buscar el ensamblado **SALLab02.dll**.



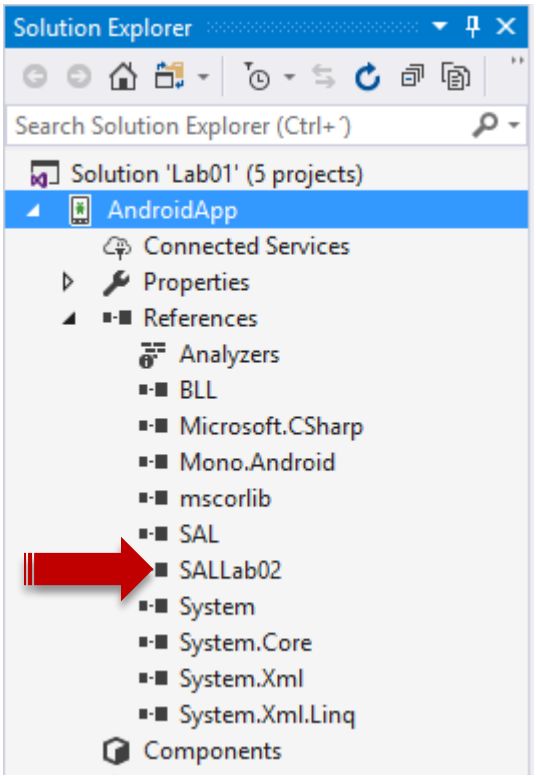
3. Selecciona el ensamblado **SALLab02.dll** y haz clic en **Add**.



4. En la ventana **Reference Manager** haz clic en **OK** para agregar la referencia.

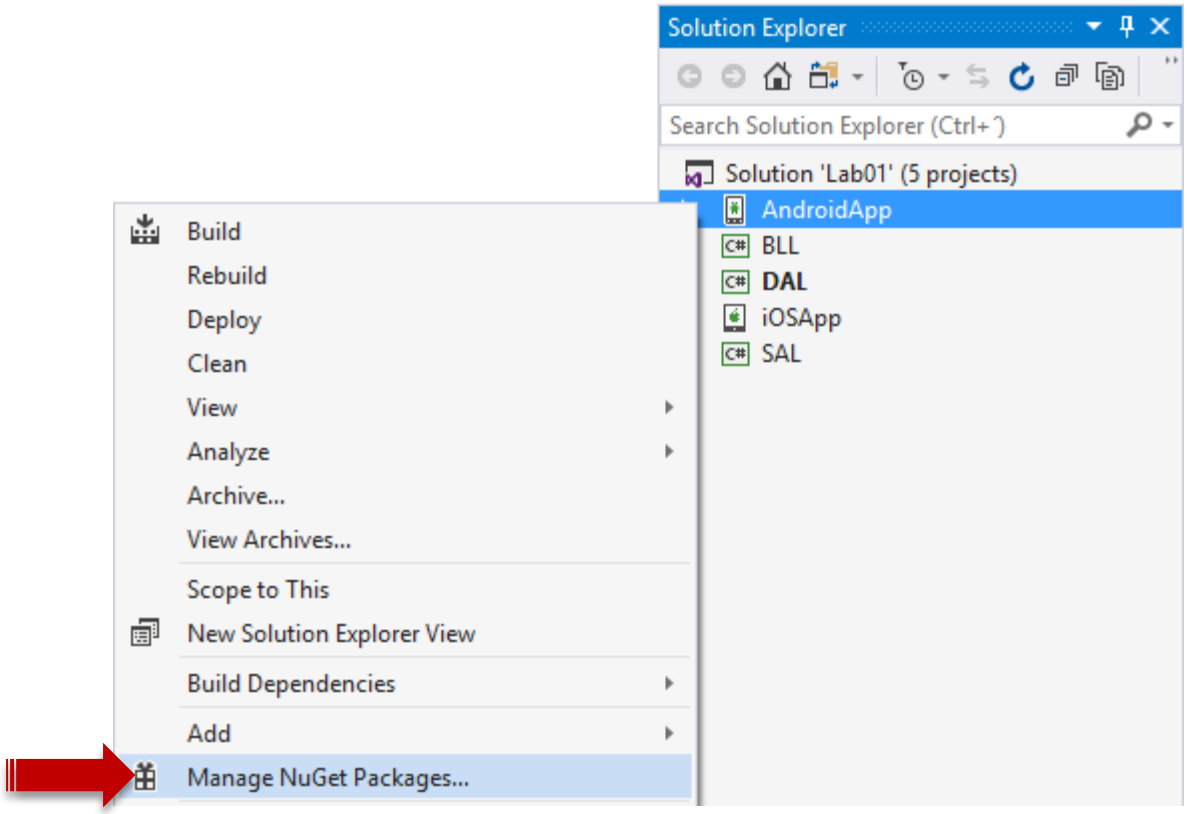


Puedes notar que la referencia ha sido agregada.

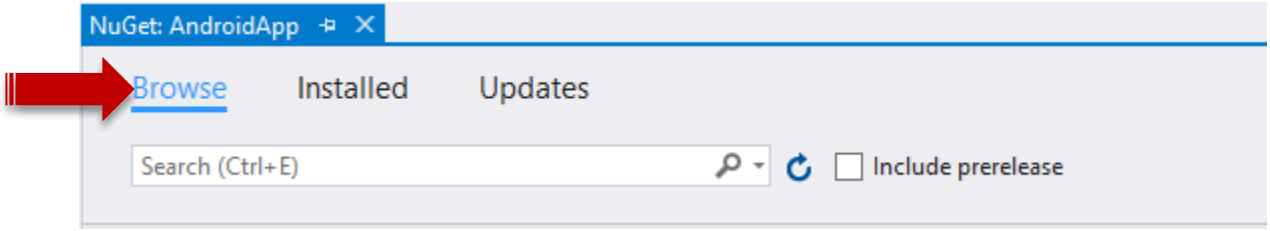


Este componente realiza una conexión a un servicio de Azure Mobile, por lo tanto, será necesario agregar el paquete NuGet **Microsoft.Azure.Mobile.Client**.

5. Accede a la opción **Manage NuGet Packages…** del menú contextual de la aplicación Android.

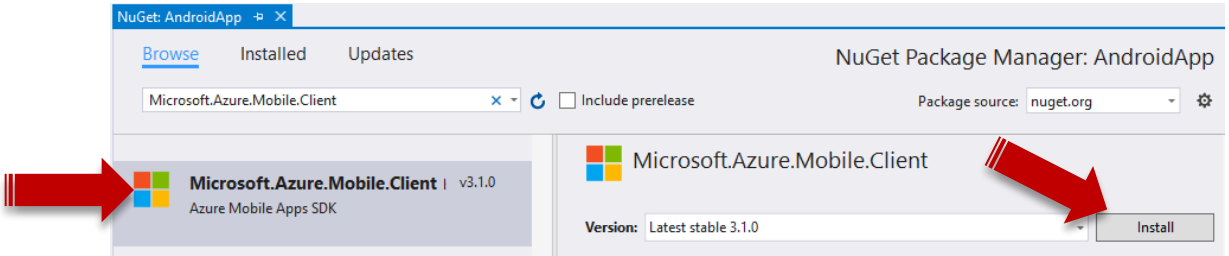


6. En la ventana **NuGet** haz clic en **Browse**.



7. En el cuadro de búsqueda escribe **Microsoft.Azure.Mobile.Client.** La lista de resultados se actualizará automaticamente conforme vas escribiendo.

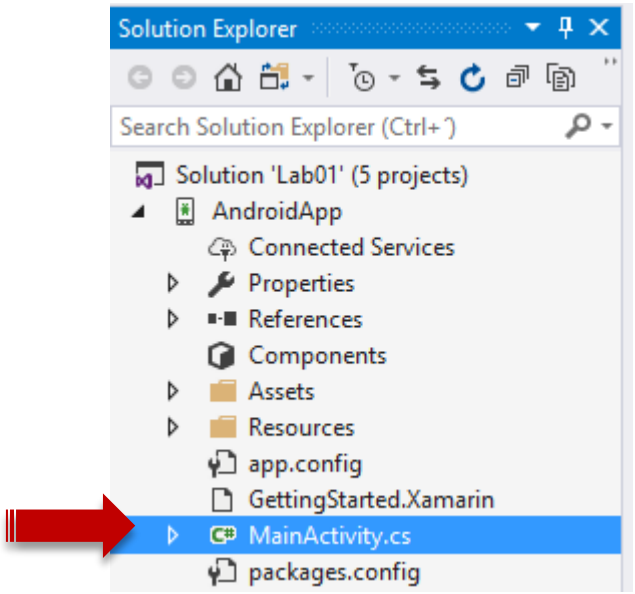
8. En la ventana de resultados haz clic en **Microsoft.Azure.Mobile.Client** y haz clic en **Install** para instalar el paquete. Acepta la instalación de los paquetes adicionales cuando te sea requerido.



Tarea 2. Agregar la funcionalidad para validar la actividad.

El componente DLL que agregaste te permite registrar tu actividad en la plataforma de TI Capacitación y Microsoft. El componente se comunica con la plataforma de TI Capacitación para autenticarte y posteriormente envía un registro a la plataforma Microsoft.

1. Abre el archivo **MainActivity.cs** ubicado dentro del proyecto Android.



2. Dentro de la clase **MainActivity** agrega la siguiente definición de método.

```
private async void Validate()
{
}
```

3. Dentro del método, agrega el siguiente código para crear una instancia del componente de acceso a servicio.

```
SALLab02.ServiceClient ServiceClient =
    new SALLab02.ServiceClient();
```

4. Agrega el siguiente código con los datos de tus credenciales de acceso a la plataforma del aula virtual de TI Capacitación que utilizas en este entrenamiento.

```
string StudentEmail = "TuCorreoElectrónico";
string Password = "TuContraseña";
```

NOTA: Después de terminar el laboratorio es importante que elimines estos datos para que no queden expuestos. En laboratorios posteriores aprenderás a solicitar esos datos en tiempo de ejecución para que no queden expuestos como código duro. La comunicación del

componente con la plataforma de TI Capacitación se realiza mediante HTTPS para seguridad de tu información.

5. Agrega el siguiente código para obtener el identificador Android que utilizas.

```
string myDevice =
    Android.Provider.Settings.Secure.GetString(
        ContentResolver,
        Android.Provider.Settings.Secure.AndroidId);
```

6. Agrega el siguiente código para invocar al método que validará tu actividad.

```
SALLab02.ResultInfo Result =
    await ServiceClient.ValidateAsync(
              StudentEmail, Password, myDevice);
```

7. Agrega el siguiente código que permitirá mostrar una alerta con el resultado de la validación.

```
Android.App.AlertDialog.Builder Builder =
    new AlertDialog.Builder(this);
AlertDialog Alert = Builder.Create();
Alert.SetTitle("Resultado de la verificación");
Alert.SetIcon(Resource.Drawable.Icon);
Alert.SetMessage(
    $"{Result.Status}\n{Result.Fullname}\n{Result.Token}");
Alert.SetButton("Ok", (s, ev) => { });
Alert.Show();
```

El siguiente paso será agregar el código que invoque a este método.

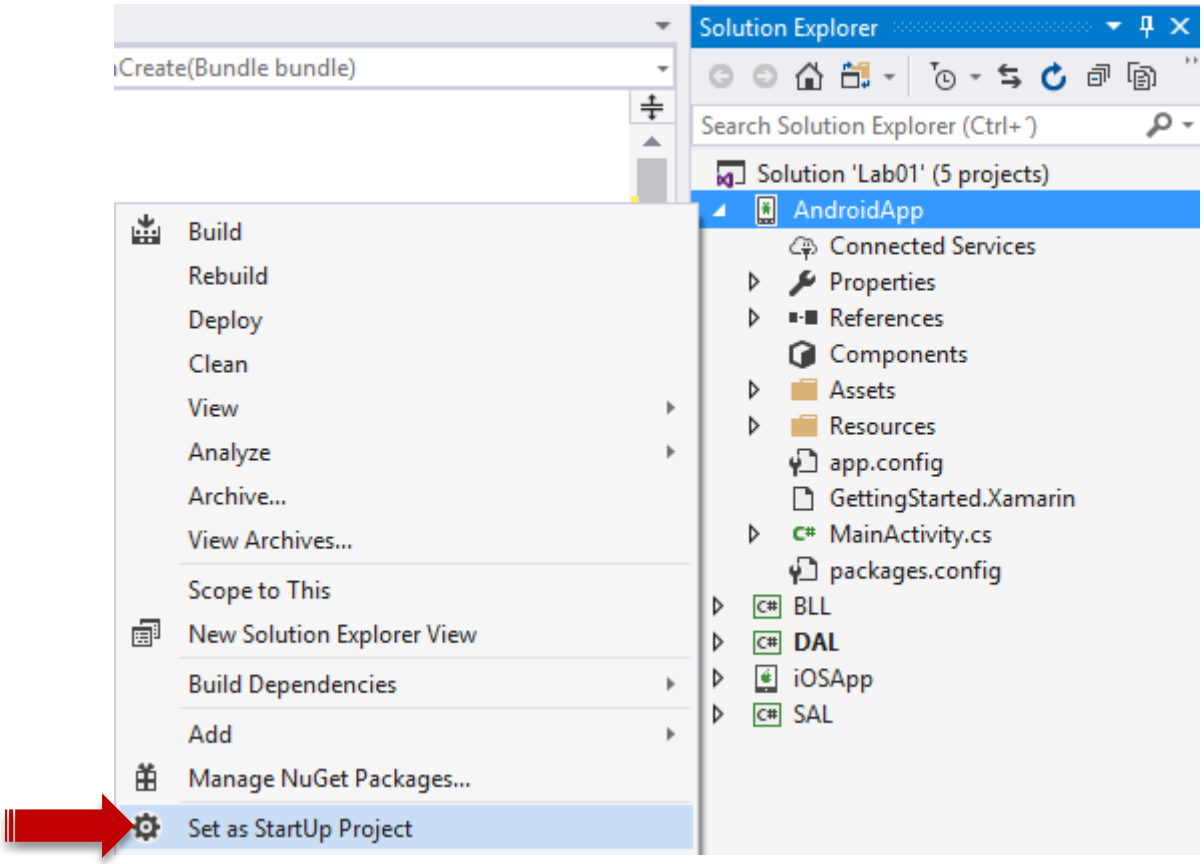
8. Agrega la siguiente instrucción en el método **OnCreate**.

```
protected override void OnCreate(Bundle bundle)
{
    base.OnCreate(bundle);
    Validate();
```

Tarea 3. Ejecutar la aplicación.

Finalmente, está todo listo para ejecutar la aplicación y realizar la validación de la actividad.

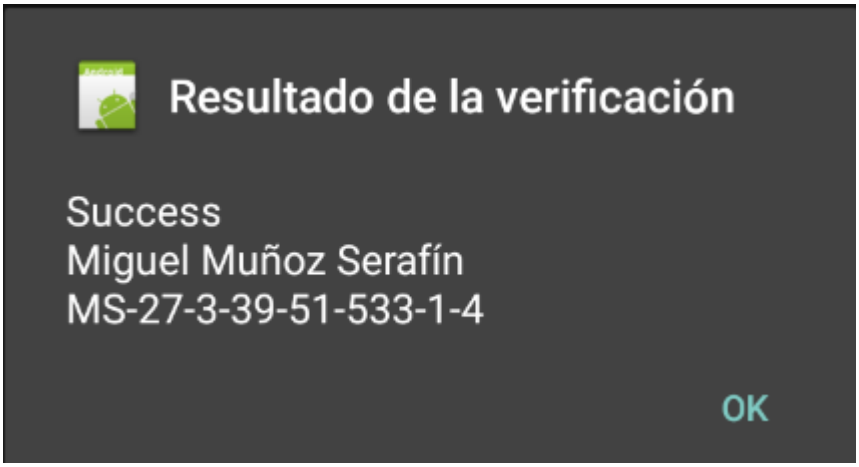
1. Selecciona la opción **Set as StartUp Project** del menú contextual del proyecto Android para establecerlo como proyecto de inicio.



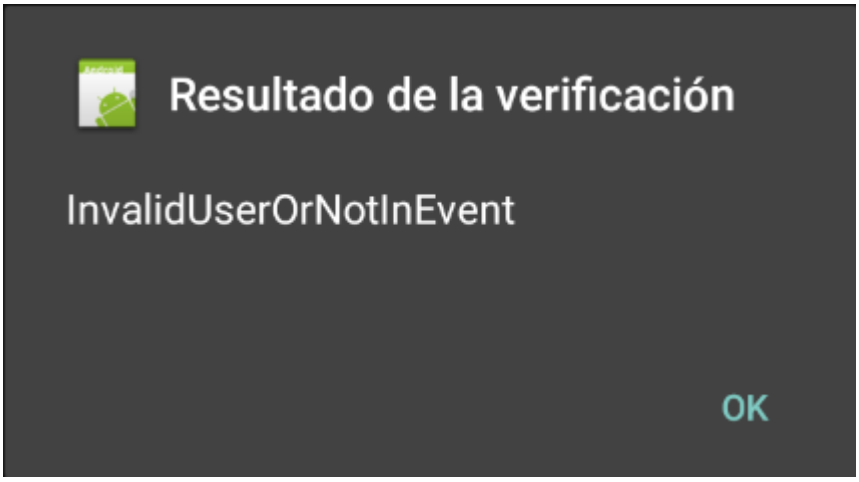
2. Selecciona el emulador Android de tu preferencia y ejecuta la aplicación.

3. La aplicación será ejecutada en el emulador e iniciará el proceso de validación de la actividad. Al finalizar la validación se mostrará un cuadro de dialogo mostrando tus datos y el estatus de la validación.

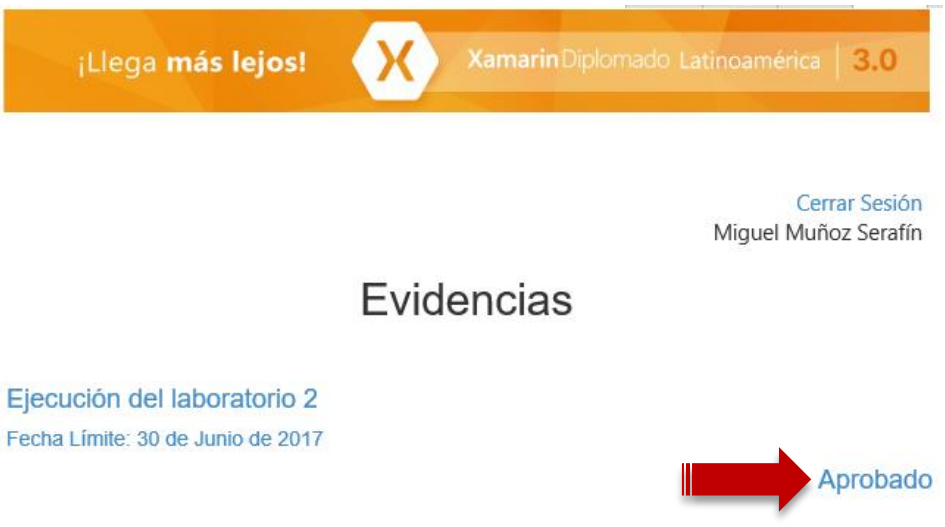
   Si la validación se realiza con éxito, te será mostrado tu nombre completo y el estatus **Success** como en la siguiente imagen.



Si proporcionaste tus credenciales incorrectas, te mostrará un mensaje similar al siguiente.



4. Cuando tu actividad se haya validado exitosamente puedes ver el estatus en el siguiente enlace: https://ticapacitacion.com/evidencias/xamarin30.



5. Regresa a Visual Studio y detén la ejecución de la aplicación.

6. **Elimina los datos de tus credenciales de acceso a la plataforma de TI Capacitación.**

   Nota: Es probable que recibas un correo similar al siguiente.

   Tu código de lab no es válido, revisa que estés utilizando un código de reto válido. Si tienes preguntas o dudas por favor contacta a dxaudmx@microsoft.com

   Puedes hacer caso omiso al mensaje.

   Si encuentras problemas durante la realización de este laboratorio, puedes solicitar apoyo en los grupos de Facebook siguientes:

   https://www.facebook.com/groups/iniciandoconxamarin/
   https://www.facebook.com/groups/xamarindiplomadoitc/

Resumen

En este laboratorio creaste la arquitectura típica de una solución Xamarin con Visual Studio. La arquitectura que creaste es solo un ejemplo de la forma en que puede ser estructurada una aplicación Xamarin multiplataforma, sin embargo, la arquitectura en una aplicación real puede variar dependiendo de los requerimientos de la aplicación o incluso de los patrones de diseño que implementes.

¿Qué te pareció este laboratorio?

Comparte tus comentarios en twitter y Facebook utilizando el hashtag ***#XamarinDiplomado.***