



# Introducción al desarrollo de aplicaciones N-Capas con tecnologías Microsoft

Manual de estudiante



Miguel Muñoz Serafín



# Introducción al desarrollo de aplicaciones N-Capas con tecnologías Microsoft

Manual de estudiante

Primera edición

Junio de 2018

Soporte técnico:  
[support@mail.ticapacitacion.com](mailto:support@mail.ticapacitacion.com)



## Contenido

Acerca del curso .....	4
Audencia.....	4
Objetivos .....	4
Requerimientos .....	5
Contenido del curso .....	5
<b>Módulo 1: La Arquitectura N-Capas.....</b>	<b>7</b>
Acerca del módulo .....	8
Objetivos .....	8
Lección 1 Definición de conceptos.....	9
Objetivos de la lección .....	9
Arquitectura basada en capas.....	10
Comunicación entre las capas de la arquitectura N-Capas.....	13
Capas Vs Niveles (Layers Vs Tiers).....	18
<b>Módulo 2: Desarrollando una aplicación N-Capas .....</b>	<b>23</b>
Acerca del módulo .....	24
Objetivos .....	24
Lección 1 Creando la capa de datos con SQL Server.....	26
Objetivos de la lección .....	26
Laboratorio Creando la base de datos con Microsoft SQL Server Management Studio.....	27
Laboratorio Creando la base de datos con la herramienta Server Explorer de Visual Studio ....	31
Laboratorio Creando el diagrama Entidad-Relación .....	37
Lección 2 Creando la capa de Entidades y Acceso a Datos con Entity Framework.....	41
Objetivos de la lección .....	41
Laboratorio Creando la capa de Acceso a Datos.....	42
Laboratorio Creando la capa de entidades .....	50
Laboratorio Definiendo la interface del patrón Repositorio.....	54
Laboratorio Implementando la interface del patrón Repositorio.....	57
Laboratorio Probando la Capa de Acceso a Datos .....	61
Lección 3 Creando la capa de Lógica de negocios.....	67



---

Objetivos de la lección .....	67
Laboratorio Creando la Capa de Lógica de Negocio .....	68
Lección 4 Creando la capa de Servicio utilizando Web API.....	72
Objetivos de la lección .....	72
Laboratorio Creando el contrato del servicio.....	73
Laboratorio Implementando el servicio .....	75
Laboratorio Probando el servicio .....	79
Lección 5 Creando la capa de Presentación.....	85
Objetivos de la lección .....	85
Laboratorio Creando una biblioteca Proxy .....	86
Laboratorio Creando una aplicación MVC.....	90
Laboratorio Creando una aplicación WPF.....	106
Laboratorio Creando el elemento ViewModel.....	109
Laboratorio Probando la aplicación WPF.....	116
Laboratorio Creando una aplicación Xamarin.Forms.....	121
Laboratorio Configurando el servicio .....	128
Laboratorio Probando la Aplicación Xamarin.Forms .....	131
<b>Resumen .....</b>	<b>134</b>



# Acerca del curso

Este entrenamiento tiene como objetivo mostrar al participante una forma sencilla de implementar la arquitectura N-Capas (N-Layers) y N-Niveles (N-Tiers) utilizando herramientas y tecnologías Microsoft. No se pretende enseñar al participante cada uno de los patrones existentes alrededor de esta arquitectura ya que se necesitaría más de un curso para lograrlo.

Este entrenamiento, se enfoca en ejemplificar la forma de utilizar Visual Studio con productos y tecnologías como SQL Server, Entity Framework, ASP.NET Web API, ASP.NET MVC, UWP y Xamarin para crear aplicaciones N-Capas y N-Niveles.

El entrenamiento inicia describiendo los conceptos N-Capas y N-Niveles para posteriormente describir sus características y razones para utilizar N-Capas o N-Niveles en el desarrollo de aplicaciones.

Al finalizar este entrenamiento, los participantes tendrán las bases para empezar a desarrollar aplicaciones N-Capas y N-Niveles.

## Audiencia

Este entrenamiento está dirigido principalmente a desarrolladores con experiencia en el desarrollo de aplicaciones .NET con acceso a base de datos.

Para un mejor aprovechamiento de este entrenamiento, se recomienda que los participantes cuenten con conocimientos básicos sobre las herramientas y tecnologías que serán utilizadas durante este entrenamiento, tales como:

- Visual Studio
- Visual C#
- SQL Server
- Entity Framework
- ASP.NET MVC
- ASP.NET Web API
- WPF
- UWP
- Xamarin

## Objetivos

Al finalizar este entrenamiento, los participantes contarán con las habilidades y conocimientos para:

- Definir los conceptos N-Capas (N-Layers) y N-Niveles (N-Tiers).
- Describir las características de la Arquitectura N-Capas.
- Describir los escenarios en los cuales es conveniente utilizar el modelo N-Capas.



- Aplicar el modelo N-Capas para el desarrollo de diferentes tipos de aplicaciones.
- Utilizar Visual Studio con productos y tecnologías como SQL Server, Entity Framework, ASP.NET Web API, ASP.NET MVC, Windows Presentation Foundation, UWP y Xamarin para crear aplicaciones N-Capas y N-Niveles.

## Requerimientos

Para poder realizar las prácticas de este entrenamiento, se recomienda contar con lo siguiente:

- Sistema Operativo Windows 10.
- Visual Studio 2017.
- SQL Server 2016 o posterior.

Los ejemplos de este entrenamiento fueron realizados sobre Windows 10 Pro, Visual Studio Enterprise 2017 y SQL Server 2017.

## Contenido del curso

El contenido de este entrenamiento consta de 2 módulos.

### Módulo 1. La Arquitectura N-Capas

En este módulo se describen los conceptos N-Capas y N-Niveles, así como la diferencia existente entre ellos. Se describen también las características y razones para utilizar N-Capas o N-Niveles en el desarrollo de aplicaciones.

Al finalizar este módulo, los participantes podrán:

- Describir los conceptos N-Capas y N-Niveles.
- Identificar aplicaciones N-Capas y aplicaciones N-Niveles.
- Describir las características de la Arquitectura N-Capas.
- Identificar las capas típicas de la mayoría de las aplicaciones N-Capas.
- Describir los escenarios en los cuales es conveniente utilizar la Arquitectura N-Capas.

El tema que forma parte de este módulo es:

- Lección 1: Definición de conceptos.

### Módulo 2. Desarrollando una aplicación N-Capas

En este módulo se aplican los conceptos teóricos descritos en el módulo anterior para desarrollar una aplicación en la cual se implementa la Arquitectura N-Capas.

Al finalizar este módulo, los participantes podrán:

- Crear la capa de origen de datos utilizando Microsoft SQL Server.



- Crear la capa de Acceso a datos utilizando Entity Framework.
- Crear la capa de Entidades utilizando una biblioteca de clases .NET Standard.
- Crear la capa de Lógica de Negocios.
- Crear la capa de Servicio utilizando ASP .NET Web API.
- Crear la capa de Presentación con una Aplicación ASP.NET MVC.
- Crear la capa de Presentación con una Aplicación Windows Presentation Foundation.
- Crear la capa de Presentación con una Aplicación Xamarin.Forms.

Los temas que forman parte de este módulo son:

- Lección 1: Creando la capa de datos con SQL Server.
- Lección 2: Creando la capa de Entidades y Acceso a Datos con Entity Framework.
- Lección 3: Creando la capa de Lógica de Negocios.
- Lección 4: Creando la capa de Servicio utilizando ASP.NET Web API.
- Lección 5: Creando la capa de Presentación.



# **Introducción al desarrollo de aplicaciones N-Capas con tecnologías Microsoft**

---

Módulo 1: La Arquitectura N-Capas



## Acerca del módulo

En este módulo se describen los conceptos N-Capas y N-Niveles, así como la diferencia existente entre ellos. Se describen también las características y razones para utilizar N-Capas o N-Niveles en el desarrollo de aplicaciones.

## Objetivos

Al finalizar este módulo, los participantes contarán con las habilidades y conocimientos para:

- Describir los conceptos N-Capas y N-Niveles.
- Identificar aplicaciones N-Capas y aplicaciones N-Niveles.
- Describir las características de la Arquitectura N-Capas.
- Identificar las capas típicas de la mayoría de las aplicaciones N-Capas.
- Describir los escenarios en los cuales es conveniente utilizar la Arquitectura N-Capas.

El tema que se cubre en este módulo es:

- Lección 1: Definición de conceptos.



# Lección 1

## Definición de conceptos

Antes de empezar con la demostración del desarrollo de aplicaciones con la arquitectura basada en Capas, es importante definir algunos de los conceptos relacionados con dicha arquitectura.

En esta lección se describen los conceptos N-Capas y N-Niveles, así como la diferencia existente entre ellos. Se describen también las características y razones para utilizar N-Capas o N-Niveles en el desarrollo de aplicaciones.

### Objetivos de la lección

Al finalizar esta lección, los participantes contarán con las habilidades y conocimientos para:

- Describir los conceptos N-Capas y N-Niveles.
- Identificar aplicaciones N-Capas y aplicaciones N-Niveles.
- Describir las características de la Arquitectura N-Capas.
- Identificar las capas típicas de la mayoría de las aplicaciones N-Capas.
- Describir los escenarios en los cuales es conveniente utilizar la Arquitectura N-Capas.



## Arquitectura basada en capas

La arquitectura basada en capas se enfoca principalmente en el agrupamiento de funcionalidad relacionada dentro de una aplicación en distintas capas que son colocadas verticalmente una encima de otra, la funcionalidad dentro de cada capa se relaciona con un rol o responsabilidad específica.

El dividir en capas una aplicación, permite la separación de responsabilidades lo que proporciona una mayor flexibilidad y un mejor mantenimiento.

Por ejemplo, en una aplicación con una capa de presentación, una capa de lógica y una capa de acceso a datos, la responsabilidad de la capa de presentación es la de interactuar con el usuario, solicitando y proporcionando la información que el usuario requiere.

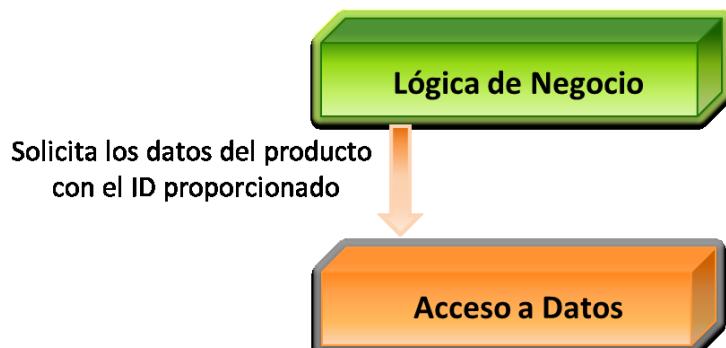
La responsabilidad de La capa de Lógica de negocio es la de hacer cumplir las reglas de negocio o requerimientos de la aplicación mientras que la responsabilidad de la Capa de acceso a datos es la de recuperar y modificar datos del origen de datos.

Supongamos que tenemos un sistema de ventas:

1. La capa de presentación solicita el ID del producto que el usuario desea comprar y se lo proporciona a la capa de lógica de negocio.

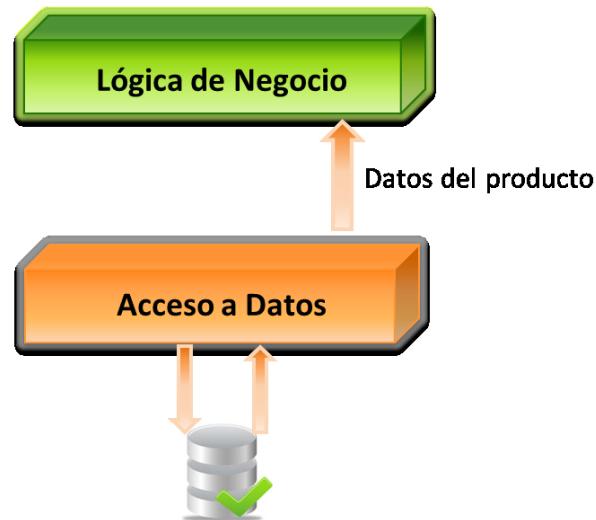


2. La capa de lógica de negocio debe verificar si aún hay existencias del producto requerido por el usuario y para esto solicita los datos del producto a la capa de acceso a datos pasándole el ID del producto a buscar.





3. La capa de acceso a datos busca la información del producto en la fuente de datos y lo devuelve a la capa de lógica de negocio.



4. Con los datos recibidos, la capa de lógica de negocio determina si el producto puede ser vendido o no y proporciona la respuesta de la consulta a la capa de presentación.



5. La capa de presentación muestra al usuario la respuesta que la capa de lógica le ha proporcionado.



En la siguiente imagen podemos ver el flujo completo entre las distintas capas de este ejemplo.

Páginas no disponibles en  
la versión demostrativa



## Laboratorio

### Creando una aplicación Xamarin.Forms

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.



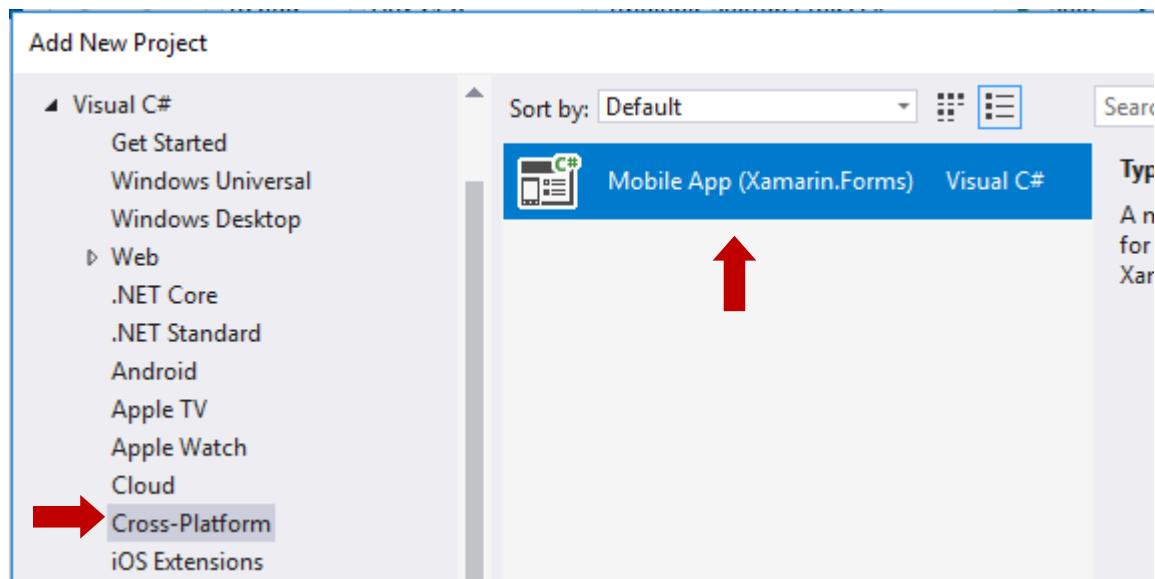
- **Model, View, ViewModel (MVVM).** El patrón Model-View-ViewModel es muy popular con los Frameworks que soportan la vinculación de datos (data-binding), tal como Xamarin.Forms. Fue popularizado por SDKs habilitados para XAML como Windows Presentation Foundation (WPF) y Silverlight donde el ViewModel 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.

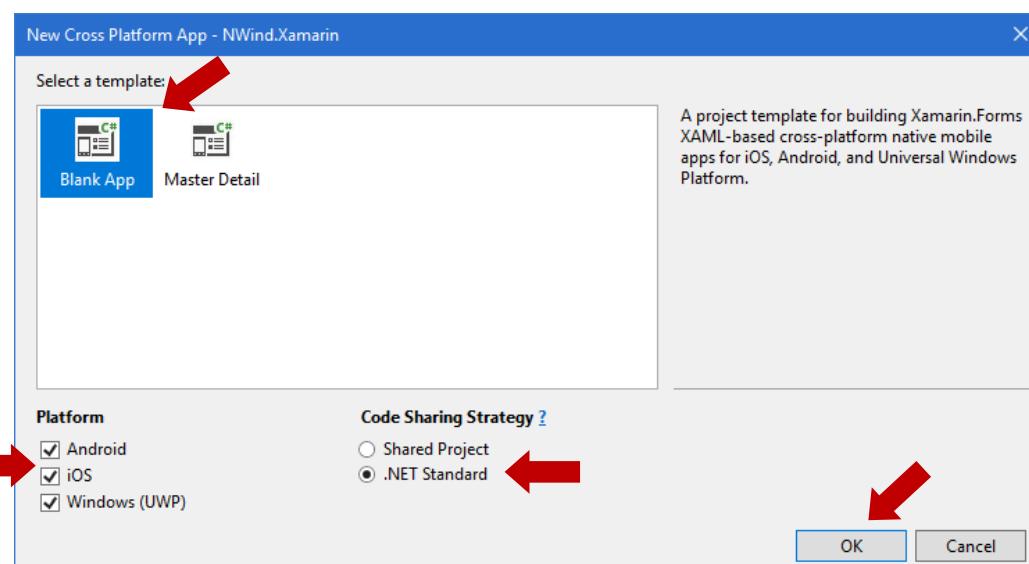


La interfaz de usuario para la aplicación Xamarin.Forms será similar a la interfaz de la aplicación WPF, proporcionándole al usuario la opción de buscar los productos correspondientes de una categoría, ver el detalle de un producto, así como crear, actualizar y eliminar un producto.

1. Agrega un nuevo proyecto a la solución de tipo **Mobile App (Xamarin.Forms)** con el nombre de **NWind.Xamarin**.

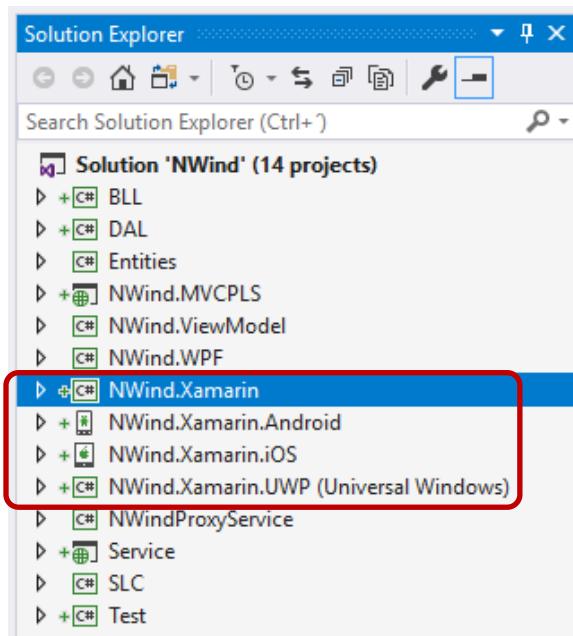


2. En la ventana **New Cross Platform App**, elige la plantilla **Blank App**, En la sección plataformas selecciona al menos una de las plataformas. Como estrategia para compartir código selecciona la opción **.NET Standard**.





Observa que en la solución se han agregados 4 proyectos (o menos dependiendo de las plataformas que hayas seleccionado), uno por cada plataforma seleccionada más uno que contiene el código compartido por las plataformas.



3. En el proyecto **NWInd.Xamarin** agrega la referencia al proyecto **NWInd.ViewModel**.
4. Agrega el paquete NuGet **Newtonsoft.Json** al proyecto **NWInd.Xamarin**.

Diseñaremos ahora la vista que permitirá listar los productos de una categoría.

5. Abre el archivo **MainPage.xaml** y agrega el espacio de nombres del *ViewModel* al elemento *ContentPage*.

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:local="clr-namespace:NWInd.Xamarin"
    x:Class="NWInd.Xamarin.MainPage"
    xmlns:vm="clr-namespace:NWInd.ViewModel;assembly=NWInd.ViewModel">
```

6. Agrega la clase **Product** como contexto de datos para esta ventana.

```
<ContentPage.BindingContext>
    <vm:Product/>
</ContentPage.BindingContext>
```

7. Modifica el elemento **<StackLayout>** con el siguiente código.

```
<StackLayout>
    <StackLayout>
        <StackLayout Orientation="Horizontal">
```

Páginas no disponibles en  
la versión demostrativa