Introducción XAML y MVVM

Sorey Bibiana García Zapata

Microsoft MVP Windows Phone Development
Arquitecto de Software en Globant

contacto@soreygarcia.com | @soreygarcia | blog.soreygarcia.me

Microsoft® Most Valuable Professional

Avanet www.avanet.org

IEEE Advancing Technology for Humanity

Globant we are ready

Tecnológico Pascual Bravo Institución Universitaria

Advertencia:

Esta presentación contiene mis propias opiniones y no necesariamente reflejan las de mi empleador o empresas con las que me encuentre vinculada.

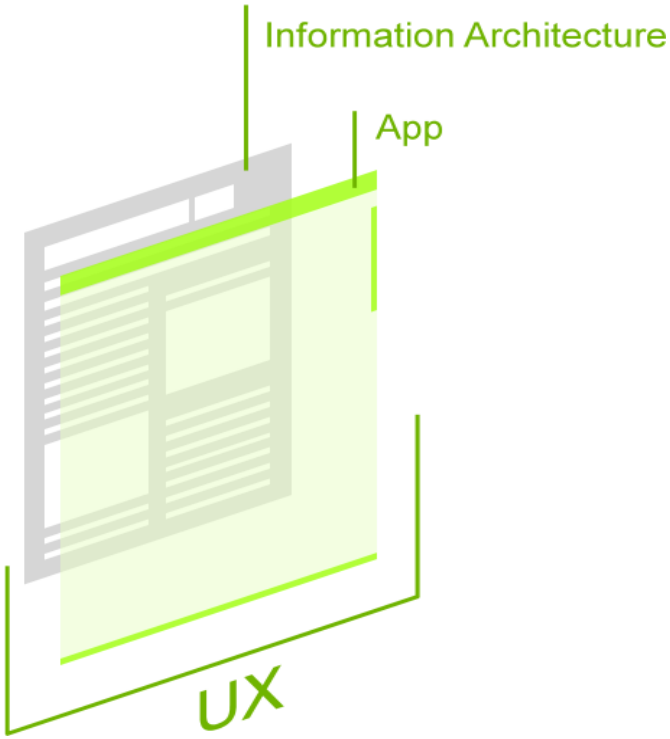
XAML

Conceptos básicos

Entendiendo las vistas

¿Qué es XAML?

- XAML es un lenguaje declarativo, basado en XML y pensado para escribir la interface gráfica de una aplicación de forma textual y ordenada.
- Una característica muy importante de XAML es que todos los objetos qué definamos en el, automáticamente son instanciados por el CLR y creados como objetos accesibles desde código, sin necesidad de realizar de nuevo la declaración de los mismos en **Code Behind**, todo gracias al mecanismo de las **Clases Parciales**.



XAML

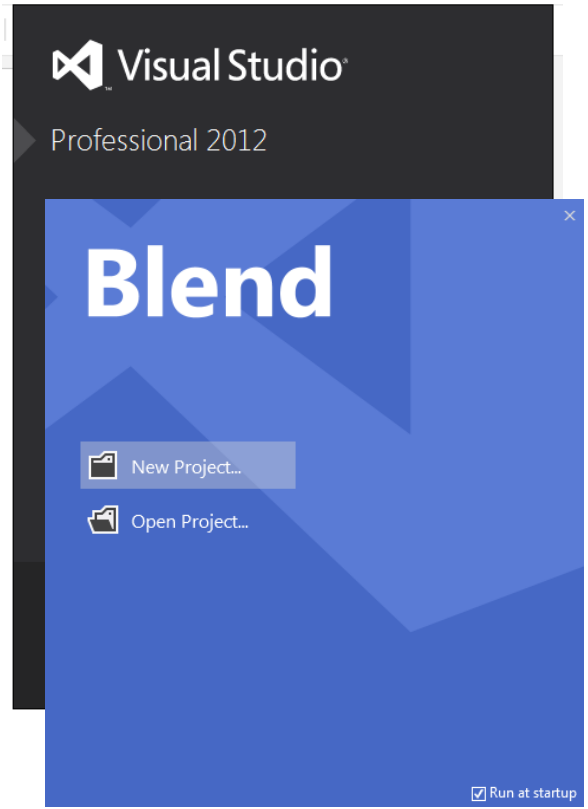
- La unidad básica de trabajo con XAML en Windows Phone y Windows 8 son las **páginas**.
- Cuando hablamos de **controles visuales** que componen una pantalla nos referimos a ellos como **elementos** de la misma.
- Internamente los controles que tienen una apariencia visual heredan de dos clases base: **UIElement y Framework Element**, que aportan capacidades extendidas de layout y otras características.
- Por último se construye la clase **Control**, que añade las características necesarias a nuestros elementos para que sean útiles al momento de construir el UI.

XAML

- Los conceptos que permiten esta separación de responsabilidades en XAML son en especial **los bindings y los comandos,** los cuales nos permiten respectivamente "enlazar" datos y acciones a nuestras vistas.
- Ambos dependen del otro concepto a manejar que se conoce como **el contexto de datos**, para el cual se requiere una adecuada comprensión de los conceptos de orientación a objetos.
- Los **convertidores** por su parte nos ayudan a **trasformar un dato** y mostrar en la vista el resultado de dicha transformación.

Separación de responsabilidades

- La posibilidad de crear la parte visual de la aplicación sin manipular el code behind, permite que diseñadores y programadores puedan trabajar en las mismas vistas sin alterar su parte correspondiente.
- Para cada actividad en particular Microsoft cuenta con herramientas que la facilitan. **Blend** está enfocado al diseño visual y **Visual Studio** está enfocado a la codificación, aunque desde ambos se puedan realizar ambas actividades con algunas limitaciones según corresponda.



Demo: Controles básicos en Visual Studio y en Blend

Bases y estructura

Implementando MVVM

Entendiendo una forma base de implementarlo

¿Qué es MVVM?

- Es un patrón de diseño creado en 2005 y es una evolución de MVC y MVP usado especialmente para aplicaciones basadas en XAML.
- La traducción de sus iniciales es **Modelo Vista – Vista Modelo**.
- Su principal propósito y beneficio está en la separación de responsabilidades a nivel de la presentación de nuestras aplicaciones.
- Las implementaciones de MVVM son diversas y dependen básicamente de las necesidades que tenga la aplicación o el desarrollo en sí mismo.
- Existen muchos frameworks disponibles para implementar MVVM en aplicaciones XAML, su elección es **personal y opcional**. El framework más más usado es MVVM light Toolkit.

MVVM

Los conceptos más sencillos usados en una implementación de MVVM básica son claramente las **Vistas, Vistas Modelos y Modelos**, soportados con base a los **bindings y comandos**.

Otros conceptos que nos proporcionan más desacoplamiento y capacidades de generación de pruebas automáticas son:

- Servicios
- Locator o Localizador de Vistas Modelo

MVVM Overview

View

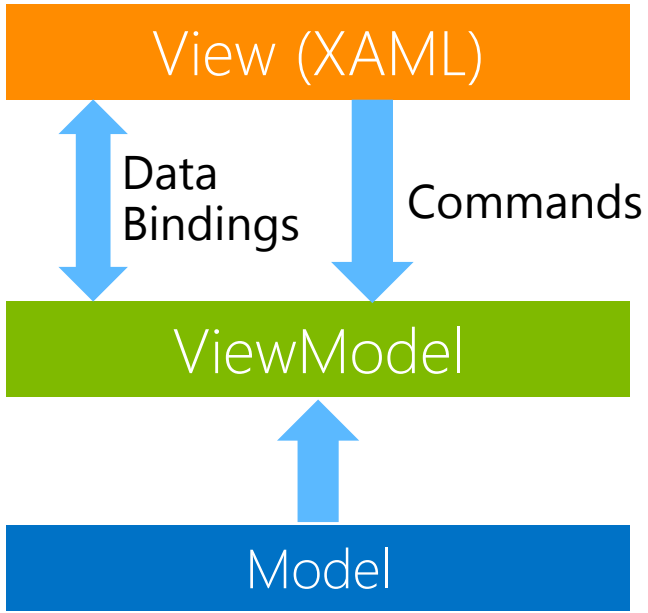
User Interface
Navigate to views
Interaction layer

ViewModel

Application logic
Service calls
Data management

Model

Simple representation of data
No logic or functionality



Singleton Pattern

- In App.xaml.cs

```
private static HueClientViewModel hueClientVM = null;
public static HueClientViewModel HueClientVM
{
    get{
        if (hueClientVM == null)
            hueClientVM = new HueClientViewModel();
        return hueClientVM;
    }
}
```

- In HueClientView.xaml.cs

```
public HueClientView()
{
    InitializeComponent();
    this.DataContext = App.HueClientVM;
}
```

Resource Pattern

- In App.xaml

```
<Application x:Class="Hue_Demo_Phone.App“ ...
         xmlns:vm="clr-namespace:Hue_Demo_Phone.ViewModels">
  <Application.Resources>
      <vm:HueClientViewModel x:Key="HueClientVM" />
  </Application.Resources>
```

- In HueClientView.xaml

```
<phone:PhoneApplicationPage ...
         "DataContext="{StaticResource HueClientVM}">
```

Demo MVVM

Algunas ideas finales

- Entender las **bases y conceptos** de la arquitectura propuesta por MVVM usando XAML es importante antes de iniciar el desarrollo.
- Antes de aventurarse a usar un framework para MVVM se recomienda saber manejar o implementar el patrón, entendiendo que muchos errores pueden ocurrir en el proceso de implementación de la aplicación al no entender correctamente los conceptos y lo que ocurre durante la ejecución de la aplicación.
- Si bien es una elección personal, usar Blend mejora altamente la productividad creando una aplicación, entendiendo que si se maneja y conoce XAML podrá tenerse más criterio para retirar las partes de código autogenerado que no consideremos apropiadas.

Consejos personales

Un ejercicio práctico con Windows Phone

Planifica la navegación y los contenedores de datos

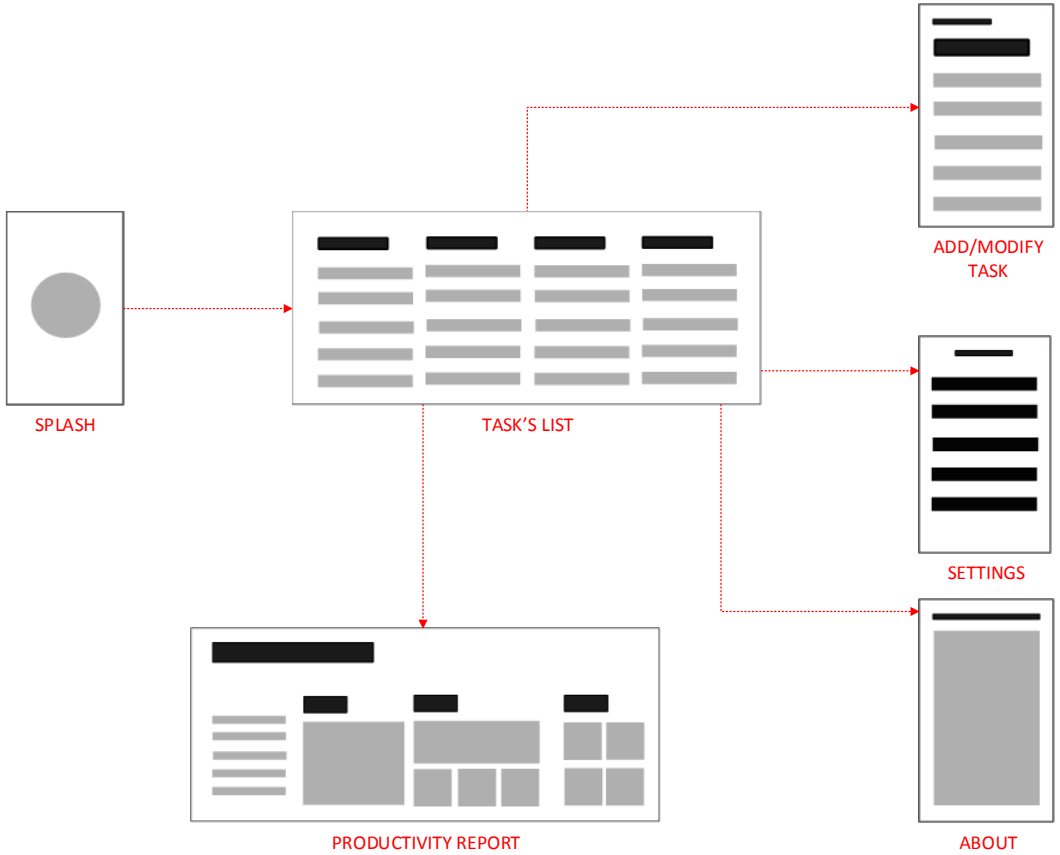
Ejemplo: Pomodori

¿Cómo debería ser la navegación y estructura de objetos de una aplicación como esta?

Fuente: http://www.slideshare.net/soreygarcia/planificando-las-bases-de-una-aplicacin-windows-phone

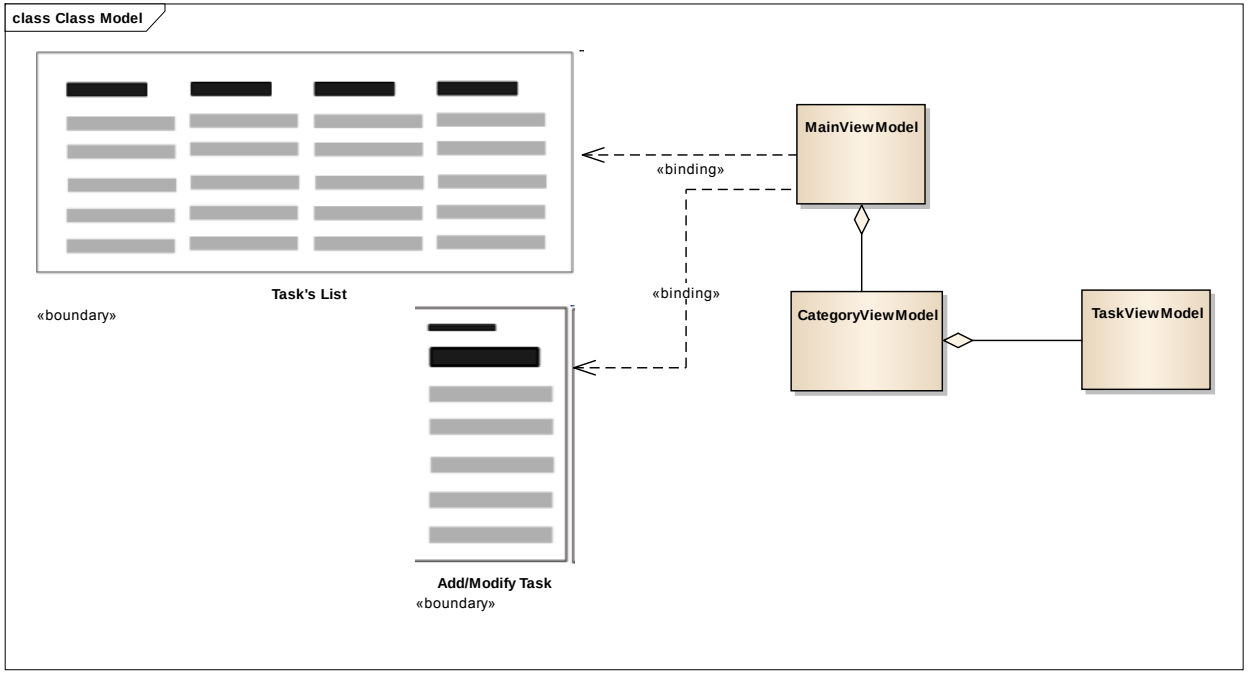
Ejemplo - Navegación

La practica más común consiste en definir una clase principal la cual es la contenedora **de todas las instancias de las demás clases contenedoras** que usa nuestra aplicación, como contenedoras no se debe entender más que las **Vista Modelo.**



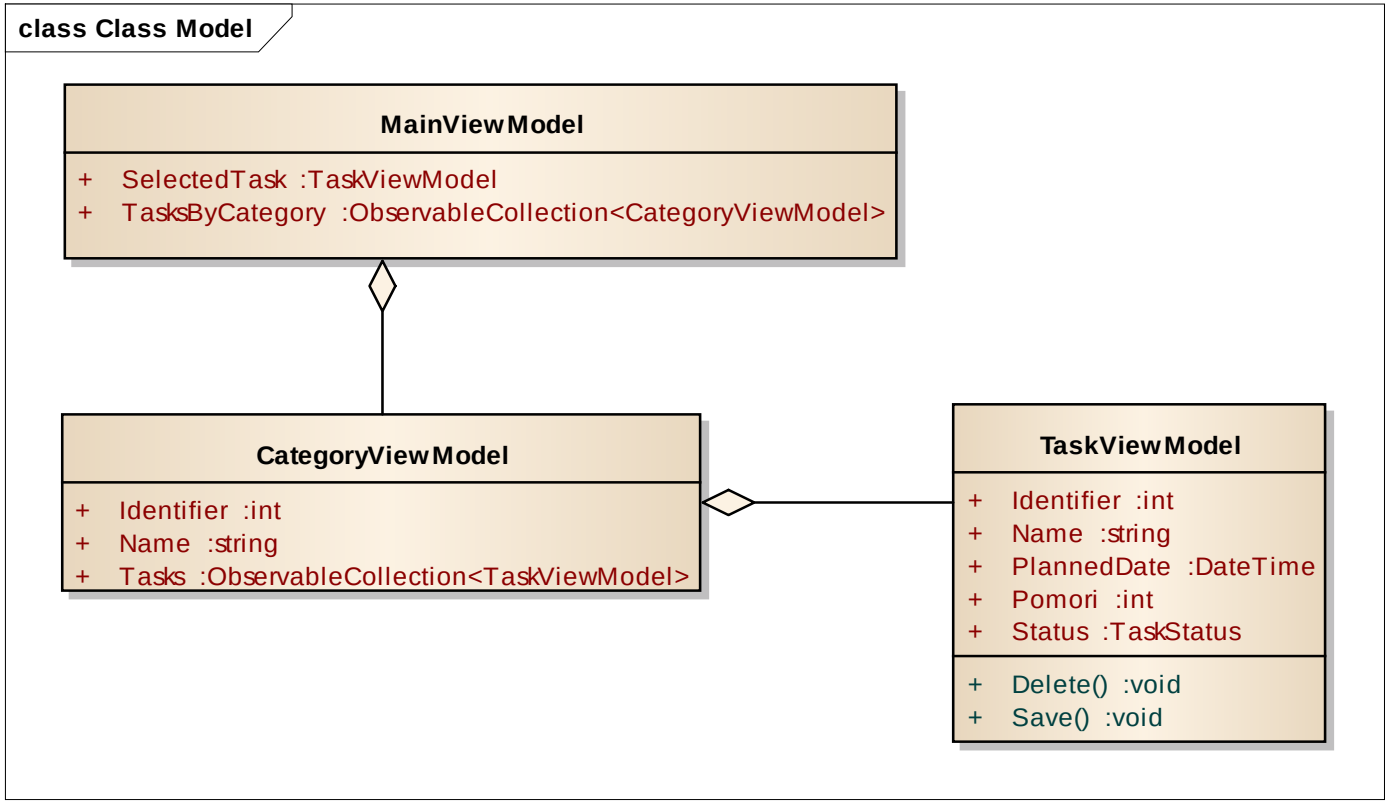
Ejemplo - Modelando

Entender **la conexión entre las vistas y las clases contenedoras** ayuda a durante el proceso de desarrollo de nuestras aplicaciones.



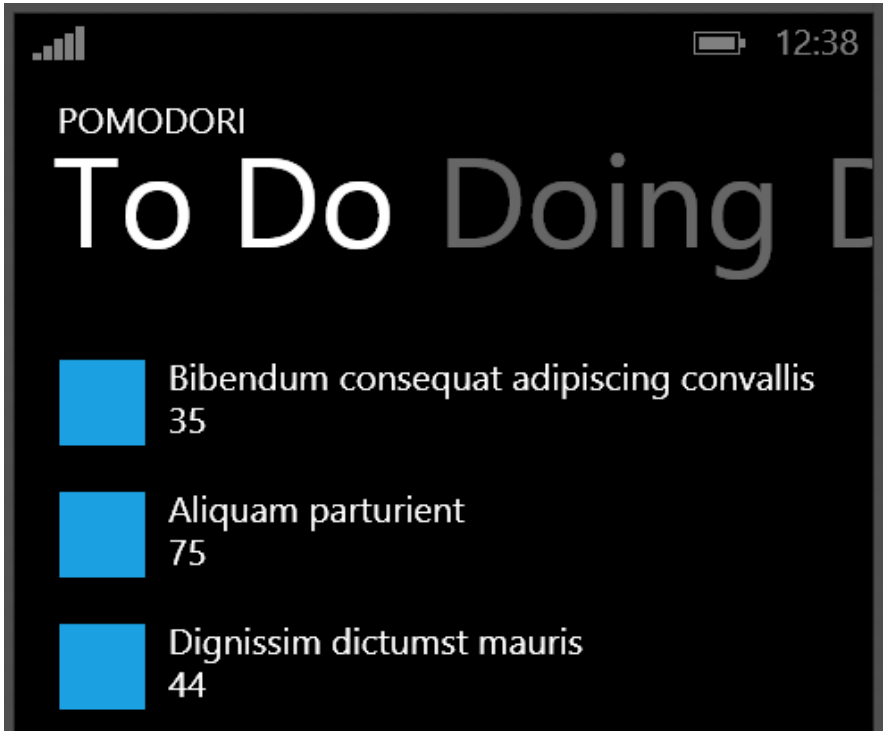
Fuente: http://www.slideshare.net/soreygarcia/planificando-las-bases-de-una-aplicacin-windows-phone

Ejemplo – Modelando Vistas Modelo

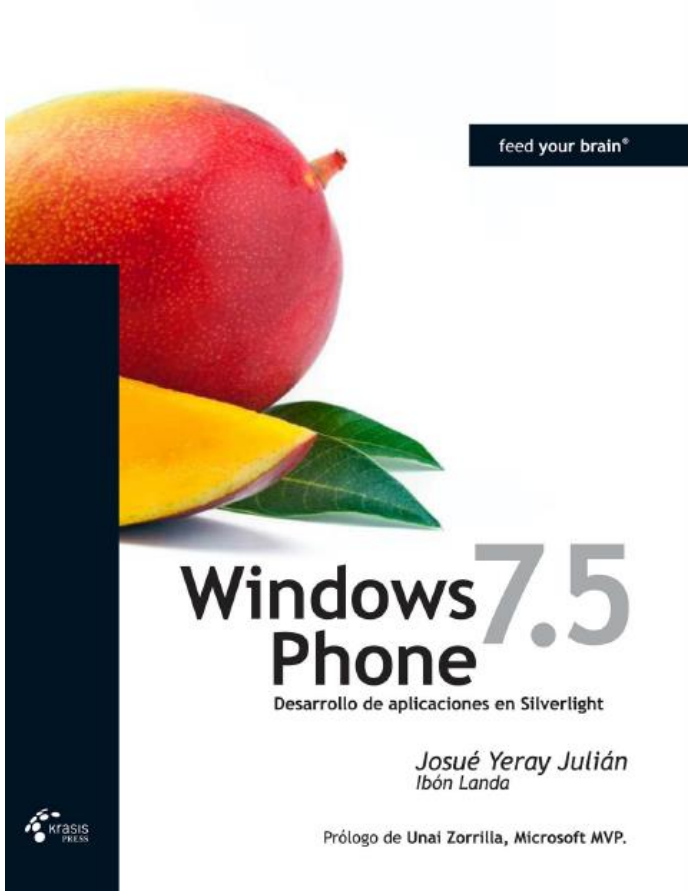


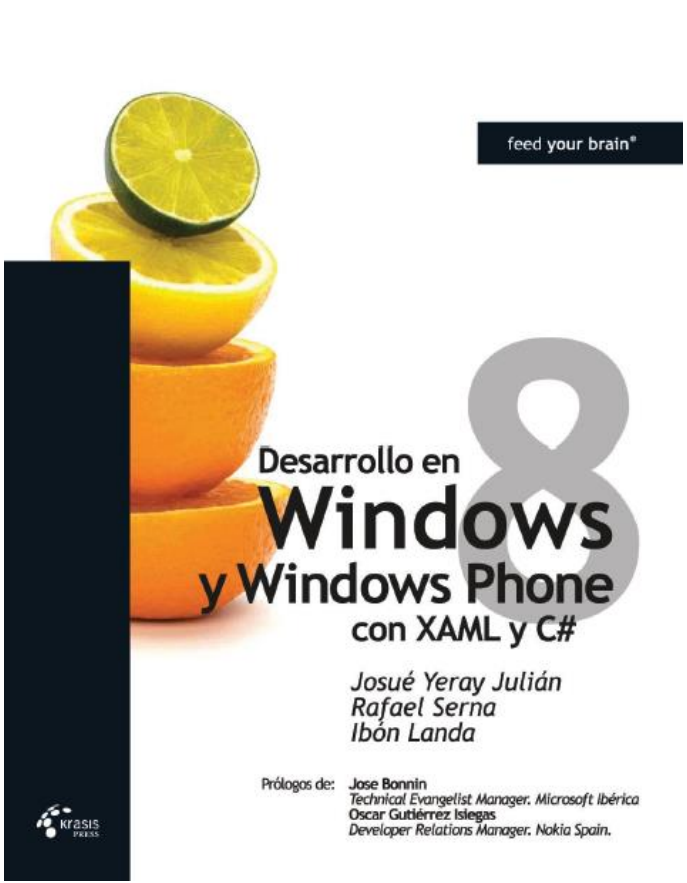
Usando datos de ejemplo

Tener el modelo de datos claro antes de empezar a construir nuestras **Vistas y Vistas Modelo**, ayudará a que el diseñador pueda trabajar de manera independiente usando datos de ejemplo, mientras que se implementa la lógica de negocio y luego se procede a la integración de ambas partes.

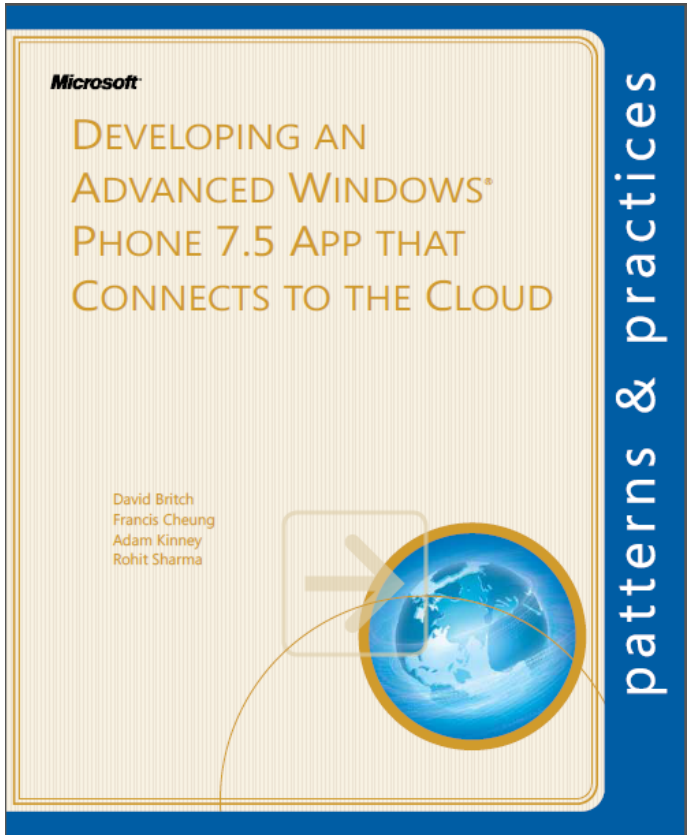


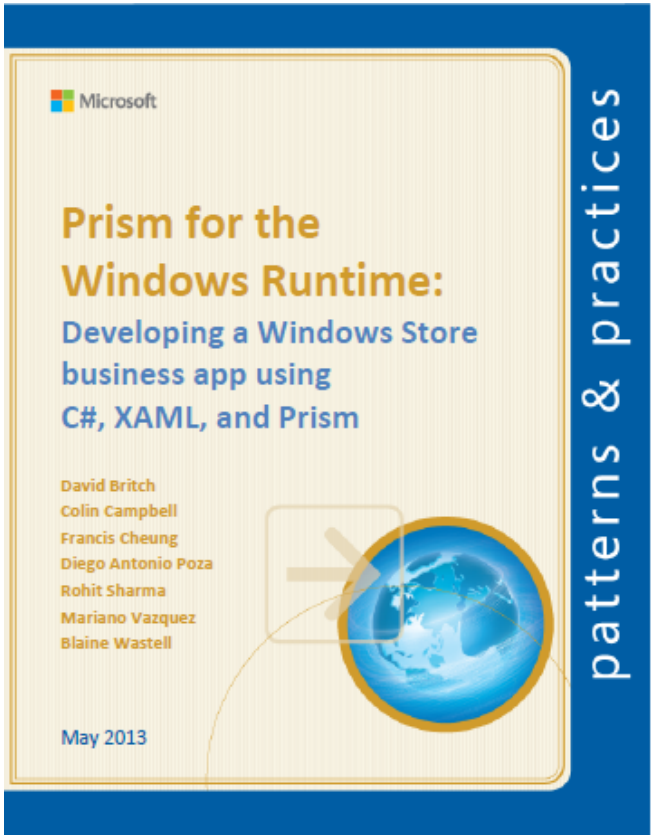
Referencias bibliográficas en español (Pagas)





Referencias en inglés (Gratuitas)





¿Preguntas?

Sorey García
contacto@soreygarcia.com | @soreygarcia | blog.soreygarcia.me