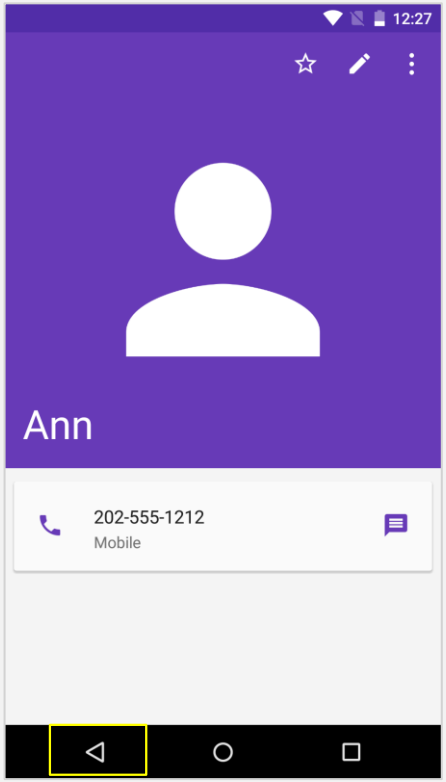
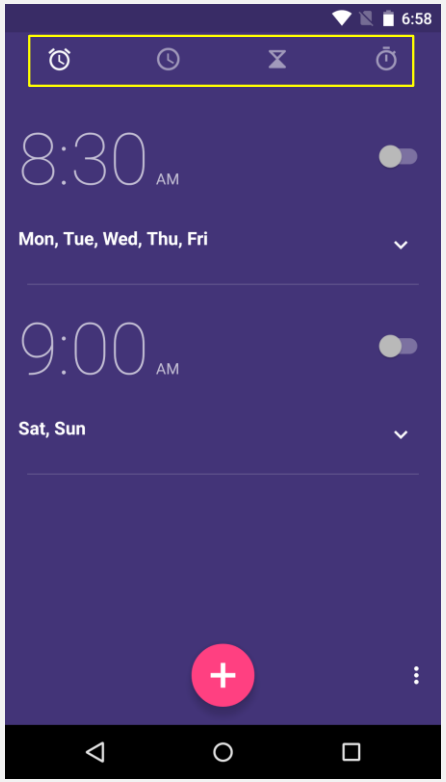
52. Navegación entre páginas

Patrones de navegación

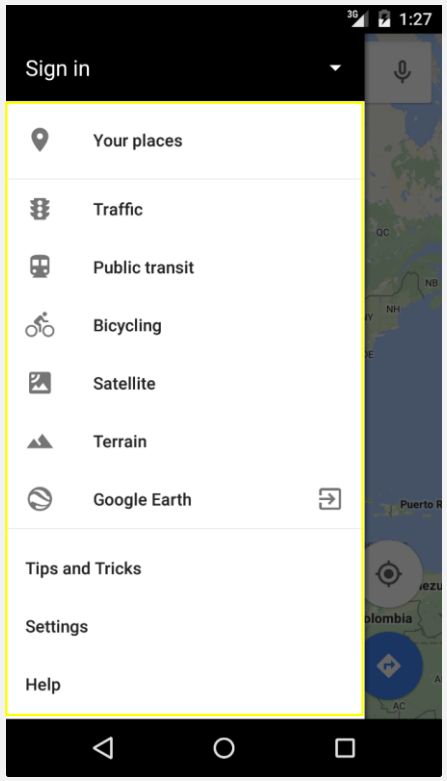
En aplicaciones móviles existen diferentes patrones de navegación.



Stack



Tab

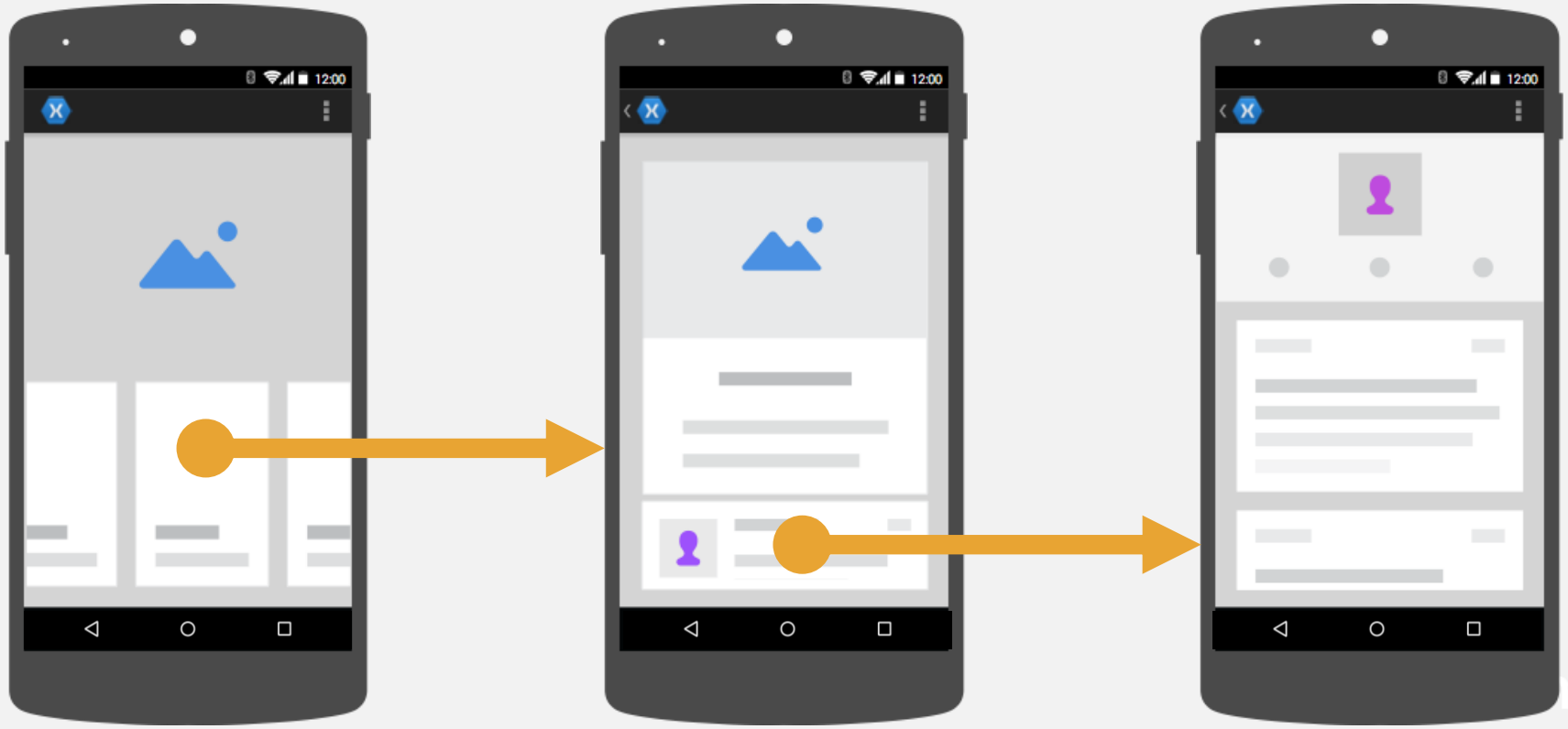


Drawer

Stack

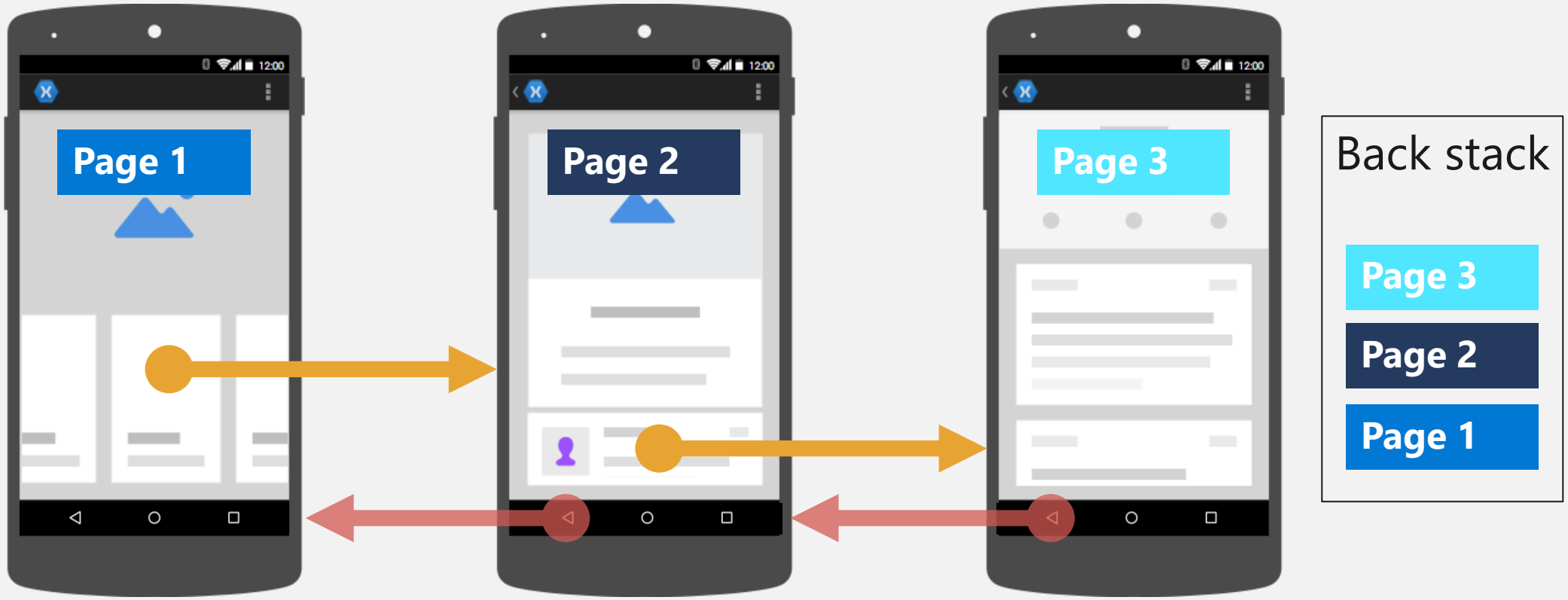
Navegar hacia adelante [definición]

La navegación hacia adelante es el proceso de pasar de una página a otra.

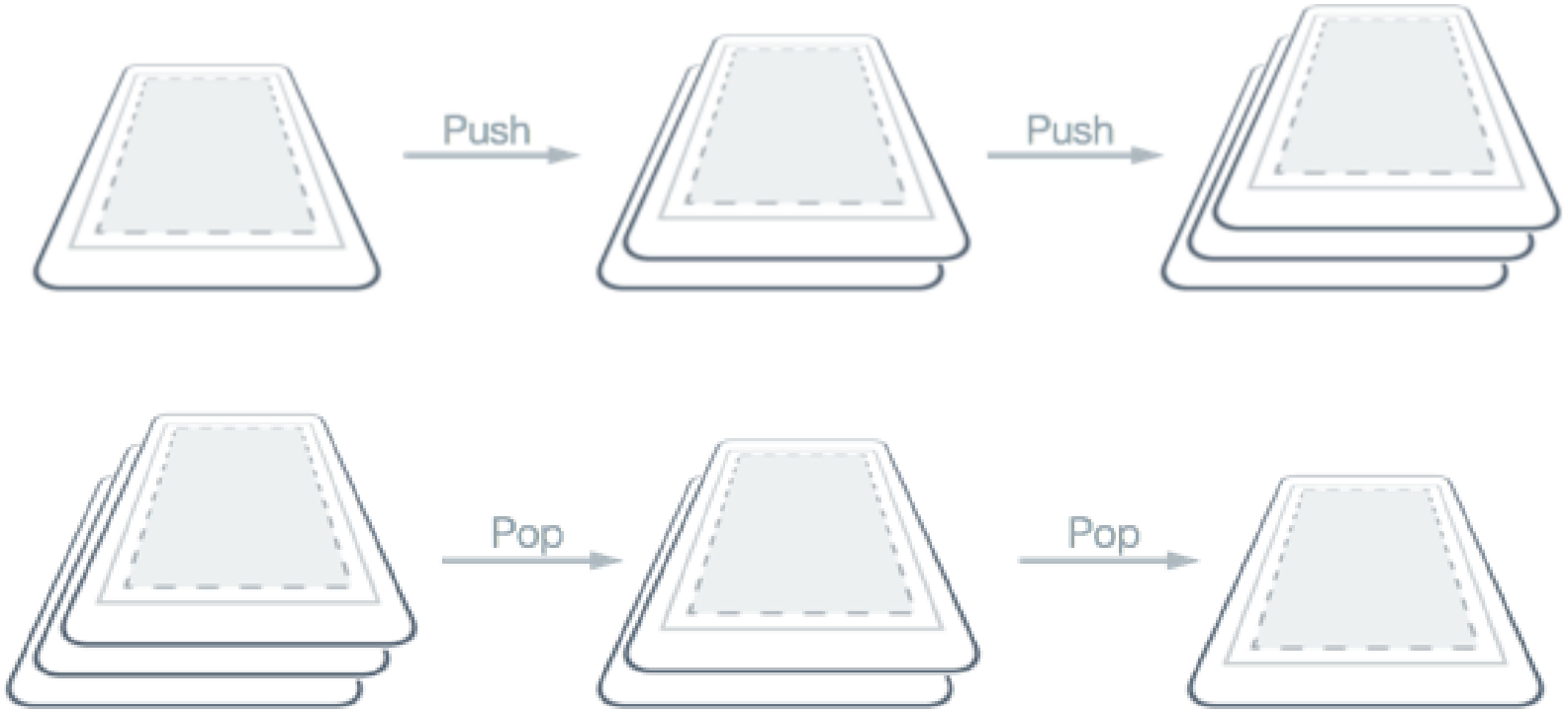


¿Qué es la navegación hacia atrás?

El back-stack es un registro histórico de las páginas por las que pasa el usuario.



Stack



Insertar páginas en la pila de navegación

Para navegar a una página, llame al PushAsync método en la Navigation propiedad de la página actual:

```
await Navigation.PushAsync(new DetailsPage());
```

En este ejemplo, el objeto DetailsPage se inserta en la pila de navegación, donde se convierte en la página activa.

Eliminar página de la pila

La página activa se puede extraer de la pila de navegación presionando el botón Atrás en un dispositivo, independientemente de si se trata de un botón físico en el dispositivo o un botón en pantalla.

Para volver mediante programación a la página anterior, se debe llamar al PopAsync método en la Navigation propiedad de la página actual:

```
await Navigation.PopAsync();
```

En este ejemplo, la página actual se quita de la pila de navegación, con la nueva página superior convirtiéndose en la página activa.

Manipular la pila de navegación

La propiedad Navigation de una página expone una propiedad **NavigationStack** desde la que se pueden obtener las páginas de la pila de navegación. Aunque .NET MAUI mantiene el acceso a la pila de navegación, la propiedad Navigation proporciona los métodos **InsertPageBefore** y **RemovePage** para manipular la pila mediante la inserción de páginas o su eliminación.

El método InsertPageBefore inserta una página especificada en la pila de navegación antes de una página existente especificada, como se muestra en el diagrama siguiente:

El método RemovePage quita la página especificada de la pila de navegación, como se muestra en el diagrama siguiente:

Shell

Shell

La clase **Shell** define las siguientes propiedades relacionadas con la navegación:

- **BackButtonBehavior**, de tipo BackButtonBehavior, una propiedad adjunta que define el comportamiento del botón Atrás.
- **CurrentItem**, de tipo ShellItem, el valor seleccionado actualmente.
- **CurrentPage**, de tipo Page, la página presentada actualmente.
- **CurrentState**, de tipo ShellNavigationState, el estado de navegación actual de Shell.
- **Current**, de tipo Shell, un alias de Application.Current.MainPage.

Navegación por rutas en Shell

Rutas

Para navegar en una aplicación Shell, se especifica un **URI** al que navegar. Los URI de navegación pueden tener tres componentes:

- Una **ruta**, que define la ruta de acceso al contenido que existe como parte de la jerarquía visual de Shell.
- Una **página**. Las páginas que no existen en la jerarquía visual de shell se pueden insertar en la pila de navegación desde cualquier lugar dentro de una aplicación shell. Por ejemplo, una página de detalles no se definirá en la jerarquía visual de Shell, pero se puede insertar en la pila de navegación si es necesario.
- Uno o varios **parámetros**. Los parámetros de consulta son parámetros que se pueden pasar a la página de destino durante la navegación.

Cuando un URI de navegación incluye los tres componentes, la estructura es:
//route/page?queryParameters

Registro de Rutas

Las rutas se pueden definir en objetos FlyoutItem, TabBar, Taby ShellContent a través de sus propiedades Route:

```
<Shell ...>
    <FlyoutItem ...
                Route="animals">
        <Tab ...
             Route="domestic">
            <ShellContent ...
                          Route="cats" />
            <ShellContent ...
                          Route="dogs" />
        </Tab>
        <ShellContent ...
                      Route="monkeys" />
        <ShellContent ...
                      Route="elephants" />
        <ShellContent ...
                      Route="bears" />
    </FlyoutItem>
    <ShellContent ...
                  Route="about" />
    ...
</Shell>
```

Registro de Rutas

En el ejemplo anterior se crea la siguiente jerarquía de ruta, que se puede usar en la navegación mediante programación:

animals
  domestic
    cats
    dogs
  monkeys
  elephants
  bears
about

Para desplazarse al objeto ShellContent de la ruta dogs, el URI de la ruta absoluta es *//animals/domestic/dogs*.

Registro de Rutas

En el constructor de subclases de Shell, o en cualquier otra ubicación que se ejecute antes de invocar una ruta, se pueden registrar explícitamente rutas adicionales para cualquier página que no esté representada en la jerarquía visual de Shell.

```
Routing.RegisterRoute("monkeydetails", typeof(MonkeyDetailPage));
Routing.RegisterRoute("beardetails", typeof(BearDetailPage));
Routing.RegisterRoute("catdetails", typeof(CatDetailPage));
Routing.RegisterRoute("dogdetails", typeof(DogDetailPage));
Routing.RegisterRoute("elephantdetails", typeof(ElephantDetailPage));
```

Navegar

Para realizar la navegación, se debe obtener primero una referencia a la subclase Shell. Esta referencia se puede obtener mediante la conversión de la propiedad App.Current.MainPage en un objeto Shell, por medio de la propiedad Shell.Current. Luego, se puede realizar la navegación mediante la llamada al método **GoToAsync** en el objeto Shell.

```
await Shell.Current.GoToAsync("//animals/monkeys");
```

Navegar hacia atrás

La navegación hacia atrás se puede llevar a cabo especificando ".." como argumento en el método **GoToAsync**:

```
await Shell.Current.GoToAsync("..");
```

Pasar datos

Los datos primitivos se pueden pasar como parámetros de consulta basados en cadenas al realizar la navegación mediante programación basada en URI. Esto se logra anexando ? después de una ruta, seguida de un identificador de parámetro de consulta, =, y un valor:

```
async void OnCollectionViewSelectionChanged(object sender, SelectionChangedEventArgs e)
{
    string elephantName = (e.CurrentSelection.FirstOrDefault() as Animal).Name;
    await Shell.Current.GoToAsync($"elephantdetails?name={elephantName}");
}
```

Procesamiento de datos de navegación

Para recibir datos de navegación, se puede decorar la clase receptora con un **QueryPropertyAttribute** para cada parámetro de consulta basado en cadenas y cada parámetro de navegación basado en objetos.

```
[QueryProperty(nameof(Bear), "Bear")]
public partial class BearDetailPage : ContentPage
{
    Animal bear;
    public Animal Bear
    {
        get => bear;
        set
        {
            bear = value;
            OnPropertyChanged();
        }
    }

    public BearDetailPage()
    {
        InitializeComponent();
        BindingContext = this;
    }
}
```

Botón atrás

La apariencia y el comportamiento del botón Atrás se pueden volver a definir estableciendo la propiedad adjunta **BackButtonBehavior**. La clase BackButtonBehavior define las propiedades siguientes:

- **Command**, de tipo ICommand, que se ejecuta cuando se presiona el botón Atrás.
- **CommandParameter**, de tipo object, que es el parámetro que se pasa a Command.
- **IconOverride**, de tipo ImageSource, el icono usado para el botón Atrás.
- **IsEnabled**, de tipo boolean, indica si se ha habilitado el botón Atrás. El valor predeterminado es true.
- **IsVisible**, de tipo boolean, indica si el botón Atrás está visible. El valor predeterminado es true.
- **TextOverride**, de tipo string, el texto usado para el botón Atrás.

Botón atrás

En el código siguiente se muestra un ejemplo de cómo volver a definir la apariencia y el comportamiento del botón Atrás:

```
<ContentPage ...>
    <Shell.BackButtonBehavior>
        <BackButtonBehavior Command="{Binding BackCommand}"
                            IconOverride="back.png" />
    </Shell.BackButtonBehavior>
    ...
</ContentPage>
```