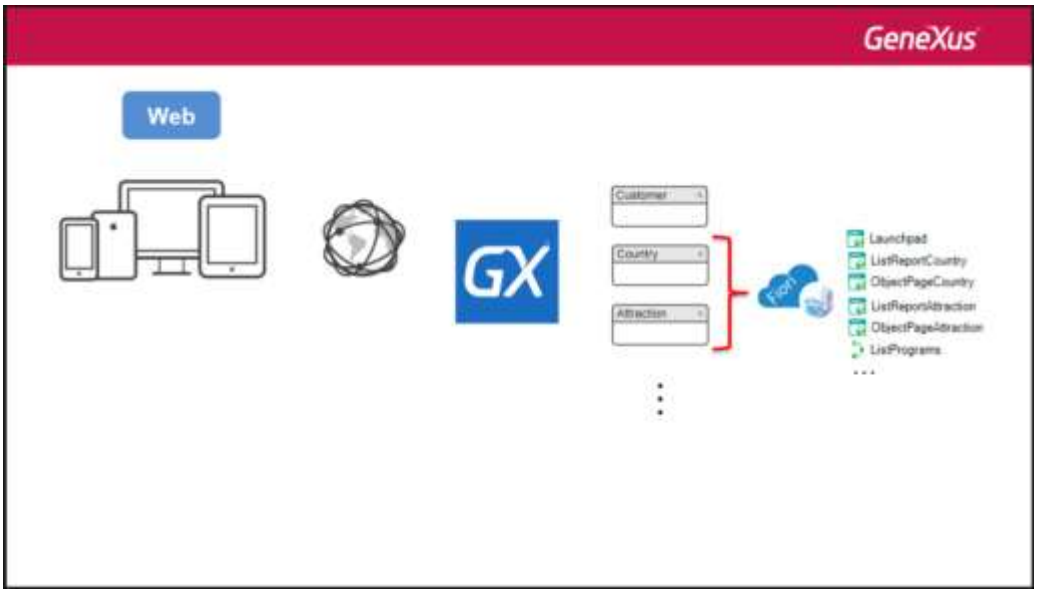
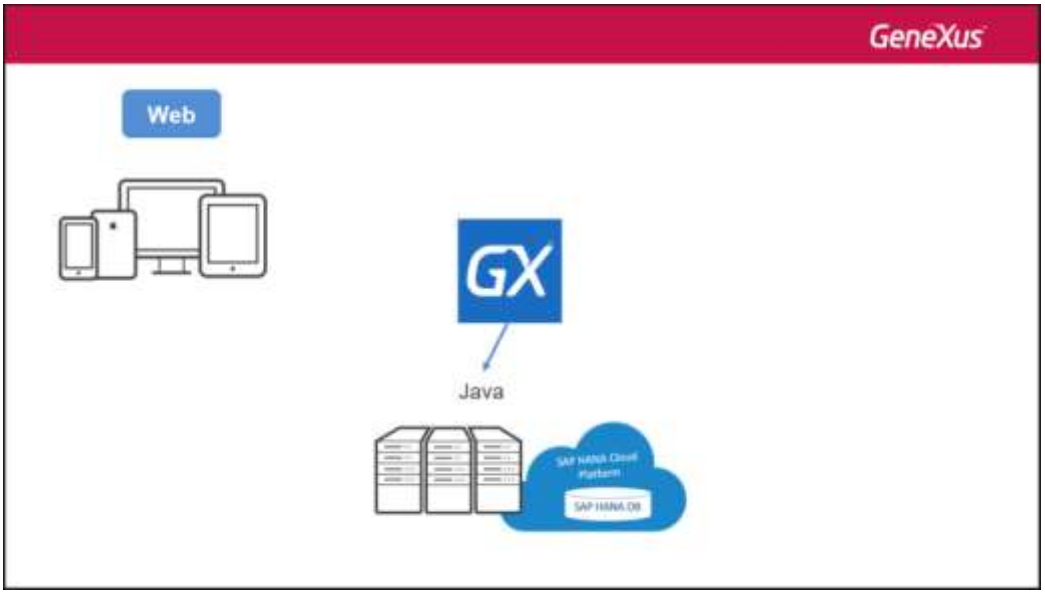
Aplicaciones nativas móviles: patrón Work With for Smart Devices

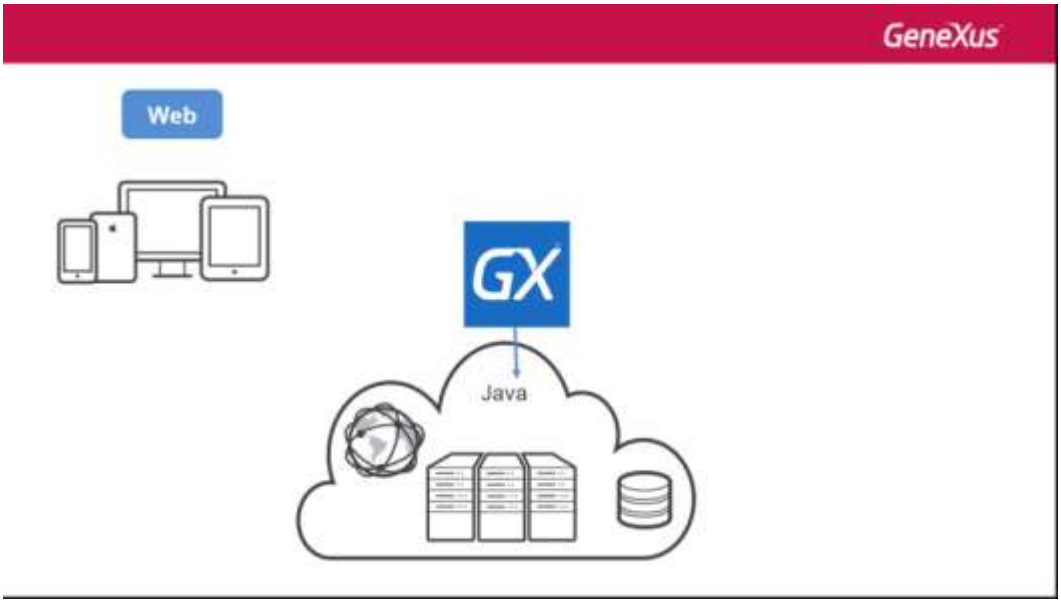
Hemos venido desarrollando una aplicación web para una agencia de viajes. Para ello, creamos objetos transacción, aplicamos pattern fiori que crea objetos como web panels, procedimientos, etc., y así seguiríamos trabajando, desarrollando objetos, hasta completar el desarrollo web.



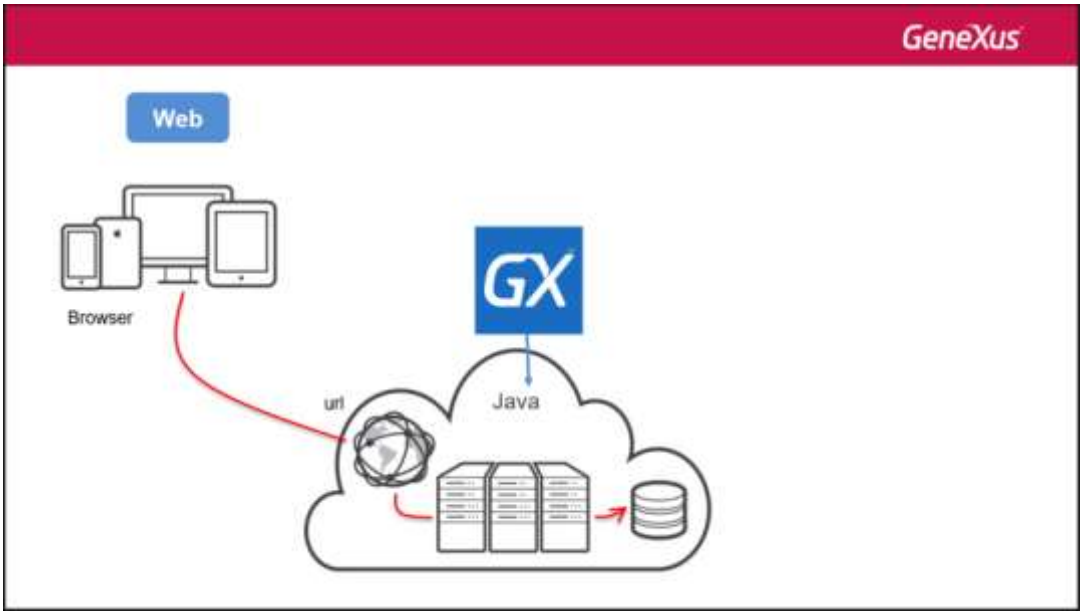
En nuestro caso hemos prototipado en un servidor local que accedía a una base de datos Hana a través de SAP Cloud Platform.



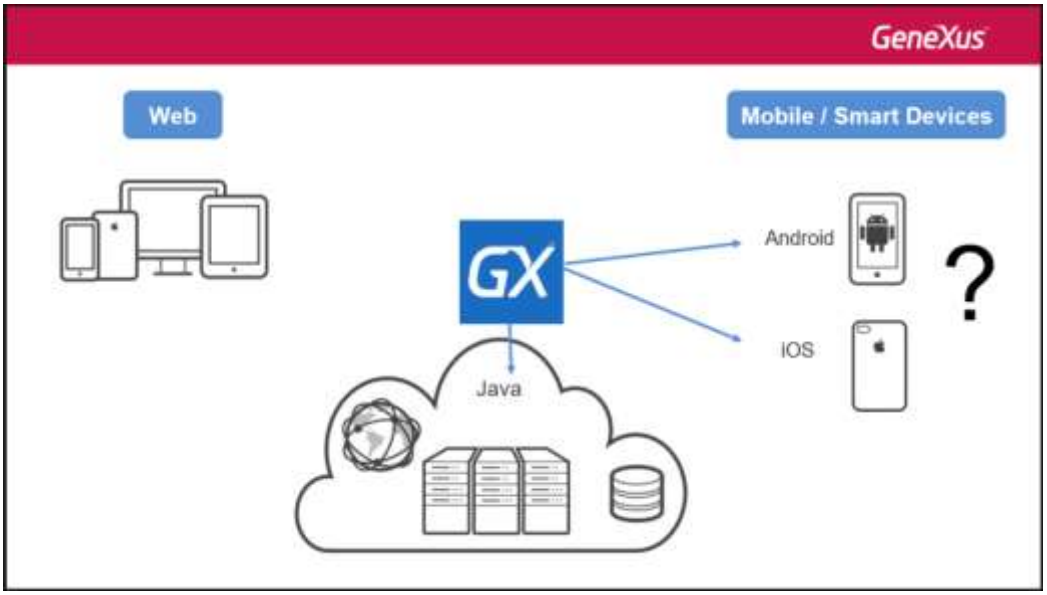
Pero cuando pongamos en producción nuestra aplicación, los programas que GeneXus construye en Java serán ubicados en un servidor accesible desde internet.



Por tanto, el usuario final, utilizando su browser y conociendo la URL de alguna de las páginas de la aplicación, ejecutará el objeto correspondiente (que GeneXus habrá construido en Java), el que, de ser necesario, realizará requests a la base de datos, devolviendo al browser la información para que arme la página con los datos recuperados, presentándosela al usuario en su pantalla.

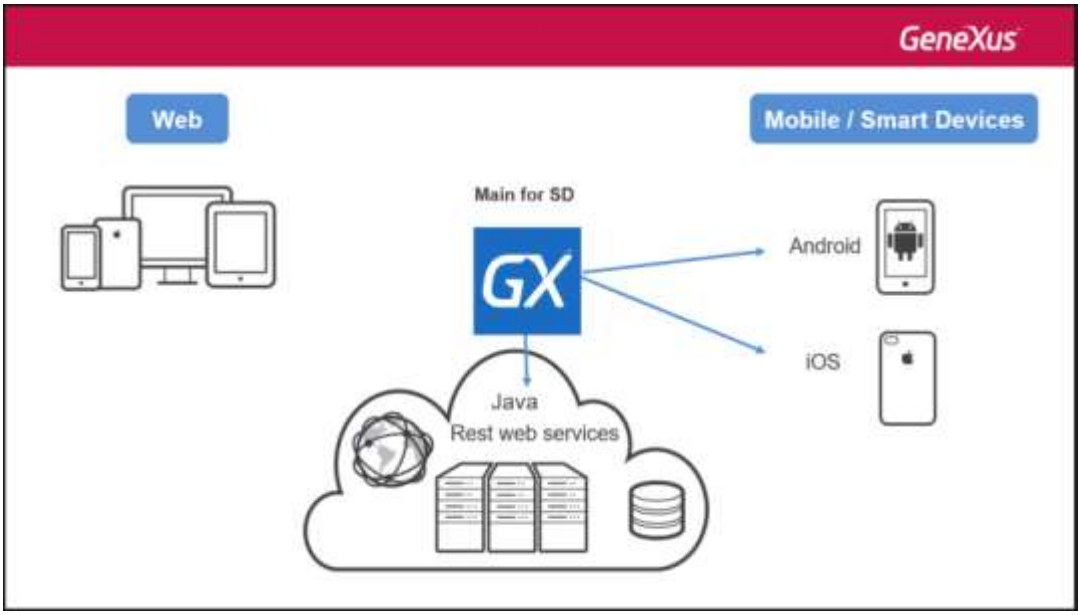


Ahora bien, ¿cómo obtenemos una aplicación similar a la web, pero nativa para dispositivos Android y/o iOS?



Lo haremos desarrollando algunos objetos GeneXus específicos para Smart Devices que se compilarán a partir de un objeto principal.

GeneXus compilará ese main tanto en Java para Android si se va a construir la aplicación en esa plataforma, como en Swift para iOS si se requiere esa otra. Pero además, para que estas apps nativas puedan acceder a los datos de la base de datos centralizada, una parte de la app web (Java) se expone como servicios Rest.

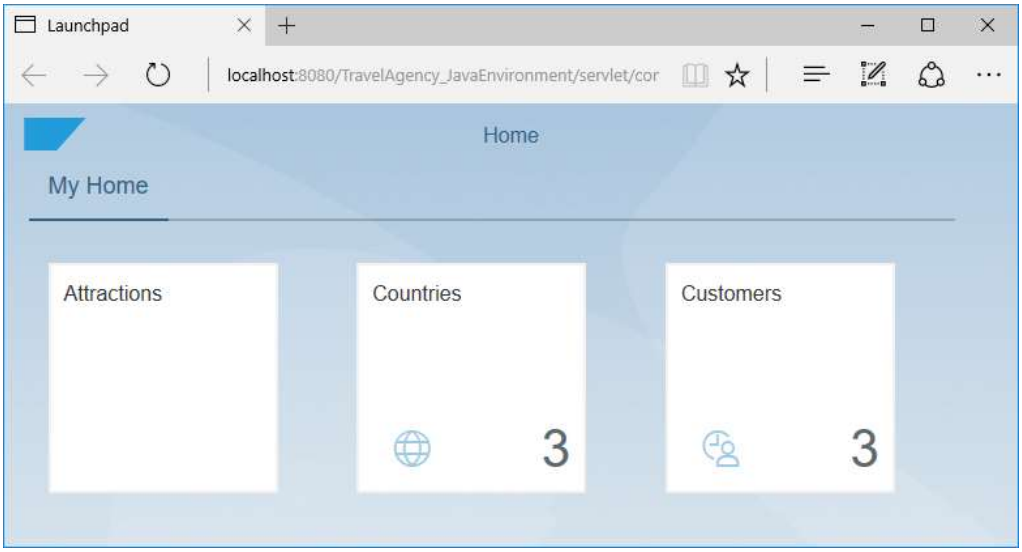


Por tanto, cuando la aplicación Android o iOS necesite datos de la base de datos, deberá consumir los servicios Rest correspondientes. De todo esto el desarrollador no debe preocuparse, pues está oculto dentro de la lógica de los objetos GeneXus para Smart Devices.

Lo veremos con un ejemplo.

Repasando lo que hemos hecho: en la aplicación web tenemos el objeto home de la app, que es el Launchpad,

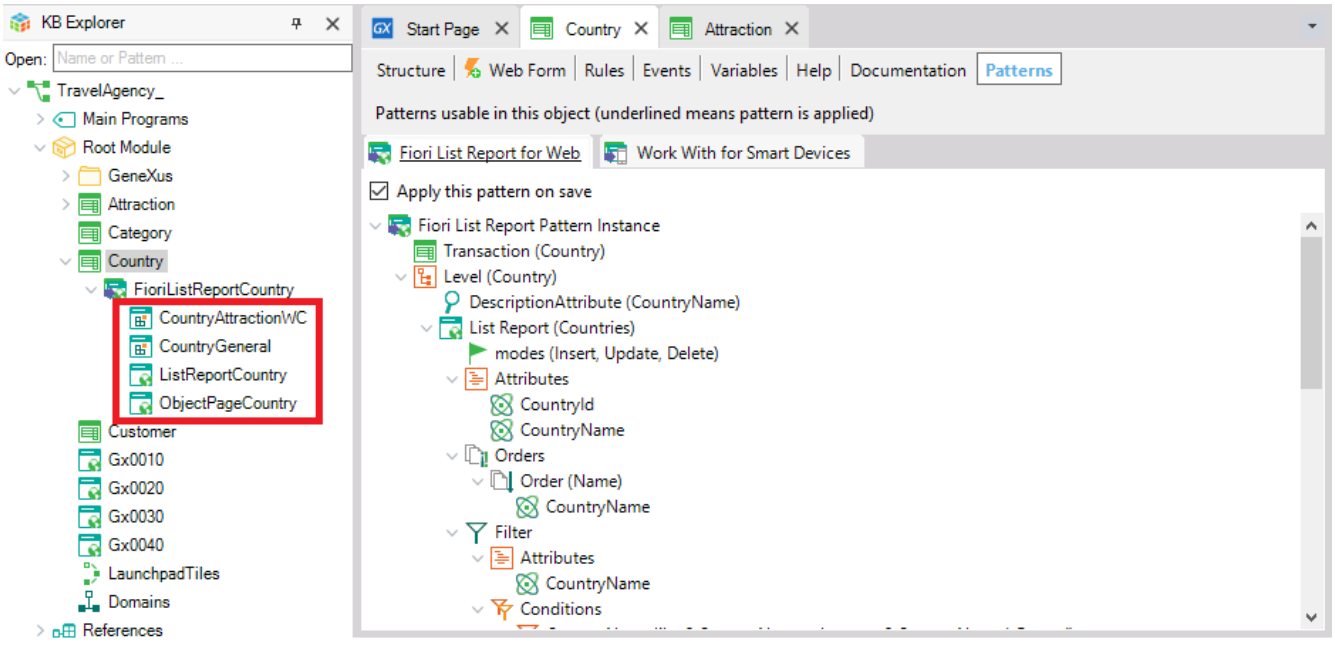
una suerte de menú:



Desde este objeto invocamos a distintos listados (el Fiori List Report de países y de atracciones). Además habíamos agregado explícitamente la invocación a una transacción, Customer. Aquí la dejaremos de lado.

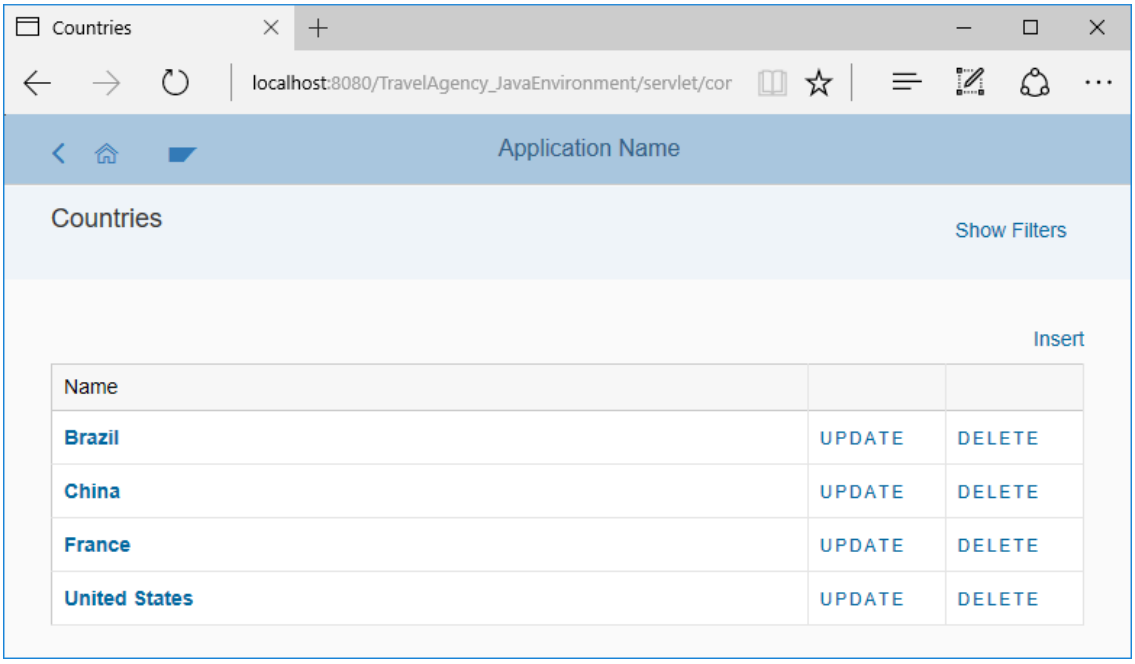
Queremos algo similiar, pero para Smart Devices. Para ello, lo primero que haremos será aplicar el patrón análogo al Fiori List Report, a las transacciones que nos interesan.

Vamos a la sección de Patterns de la transacción Country, patrón Work With for Smart Devices. Podemos ver grandes diferencias con el mismo patrón para Web. El Fiori List Report presentaba un árbol con nodos para declarar lo que GeneXus debía implementar luego a través de estos objetos:

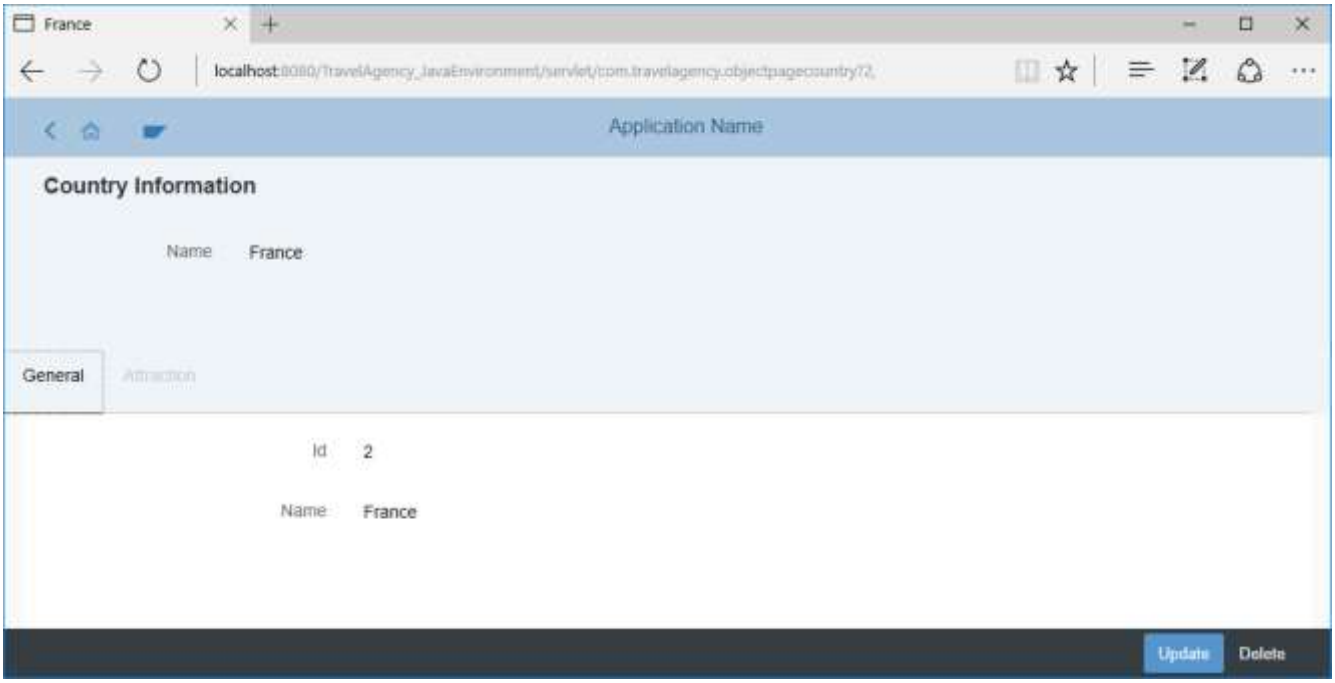


Estos objetos en ejecución se mostraban básicamente como dos pantallas:

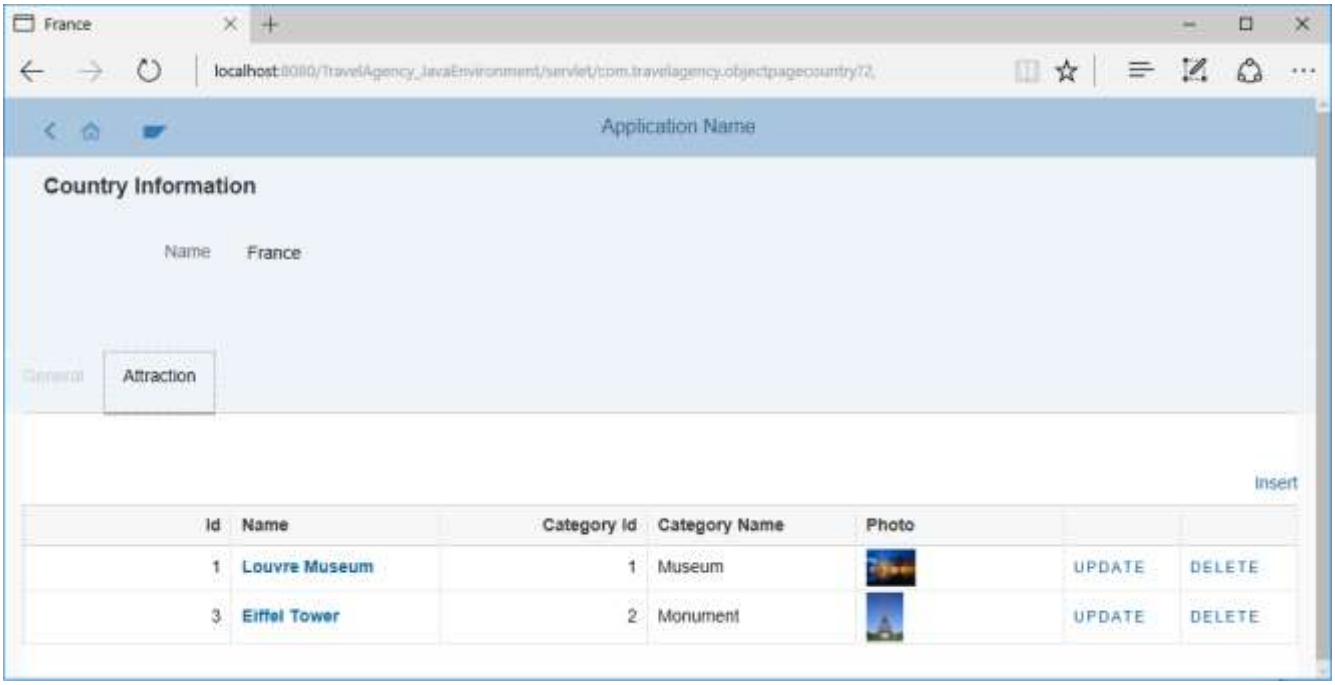
La del List, propiamente dicho:



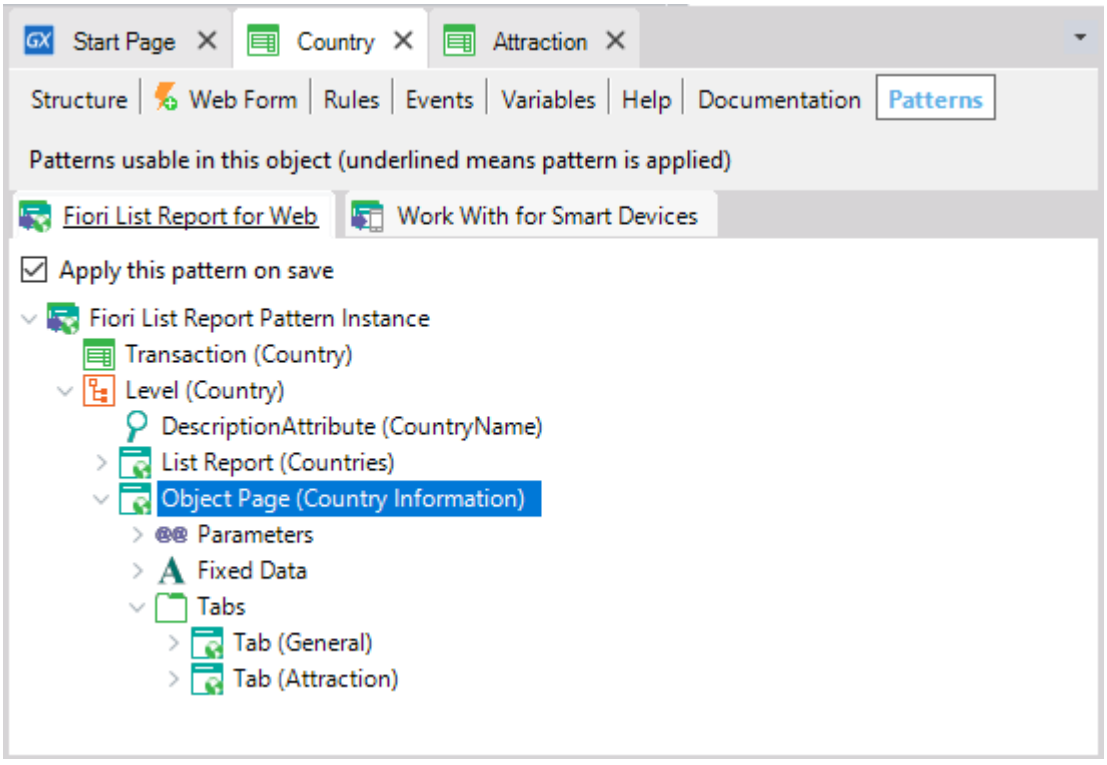
Y la de información del país elegido:



Esta pantalla tiene un tab “General” y, como cada país tiene atracciones asociadas, otro tab “Attraction” con esas atracciones desplegadas en un grid:

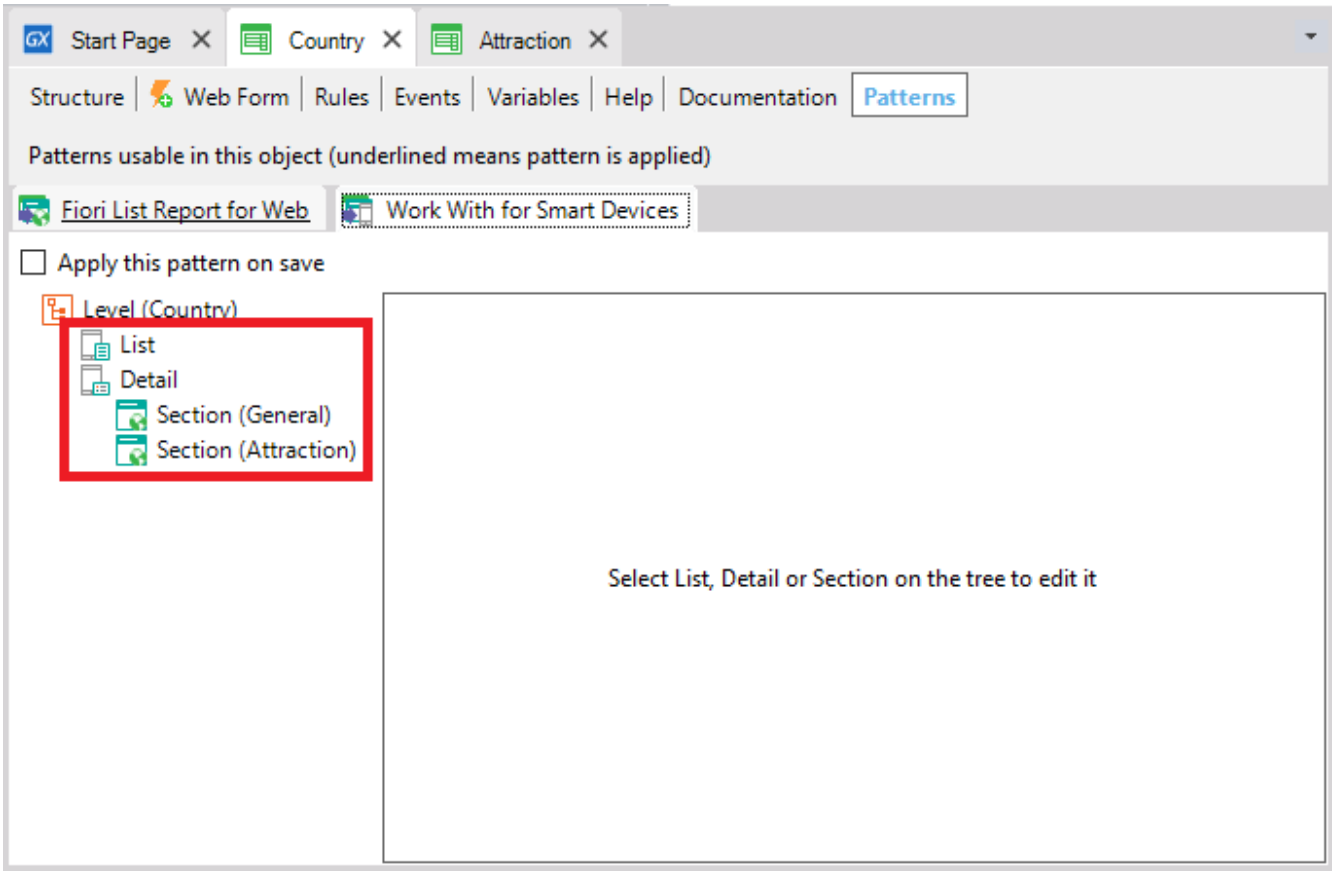


Si colapsamos el árbol adecuadamente, podemos ver fácilmente dónde se configura cada una de estas pantallas:



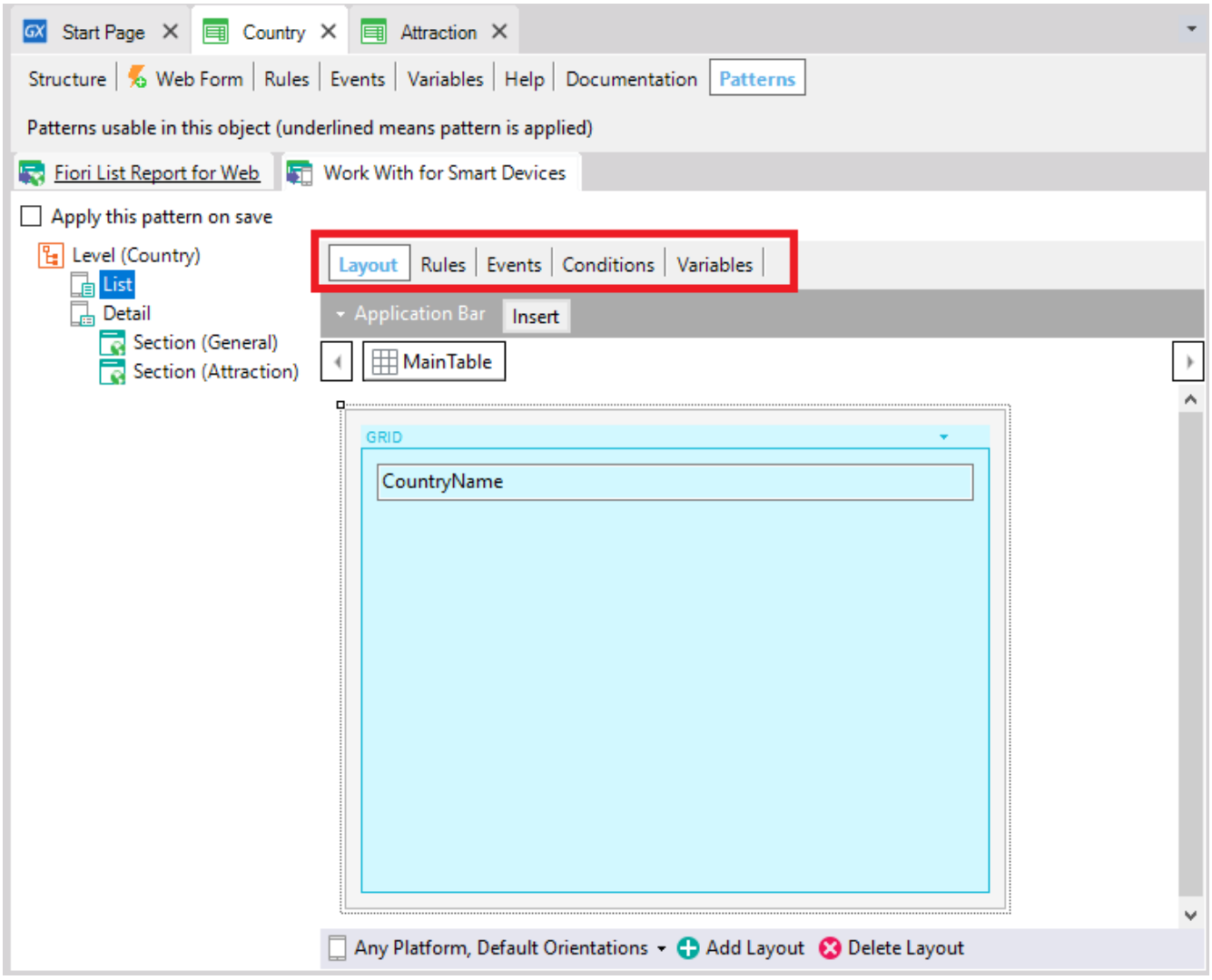
Aquí tenemos la del List y aquí la de información detallada de un país, con sus dos tabs.

Si ahora vamos al pattern análogo para Smart Devices:



Vemos más claramente las dos pantallas que se generarán: la del List de países y la que corresponde a la información detallada de un país, con sus dos tabs, que aquí aparecen como “sections”.

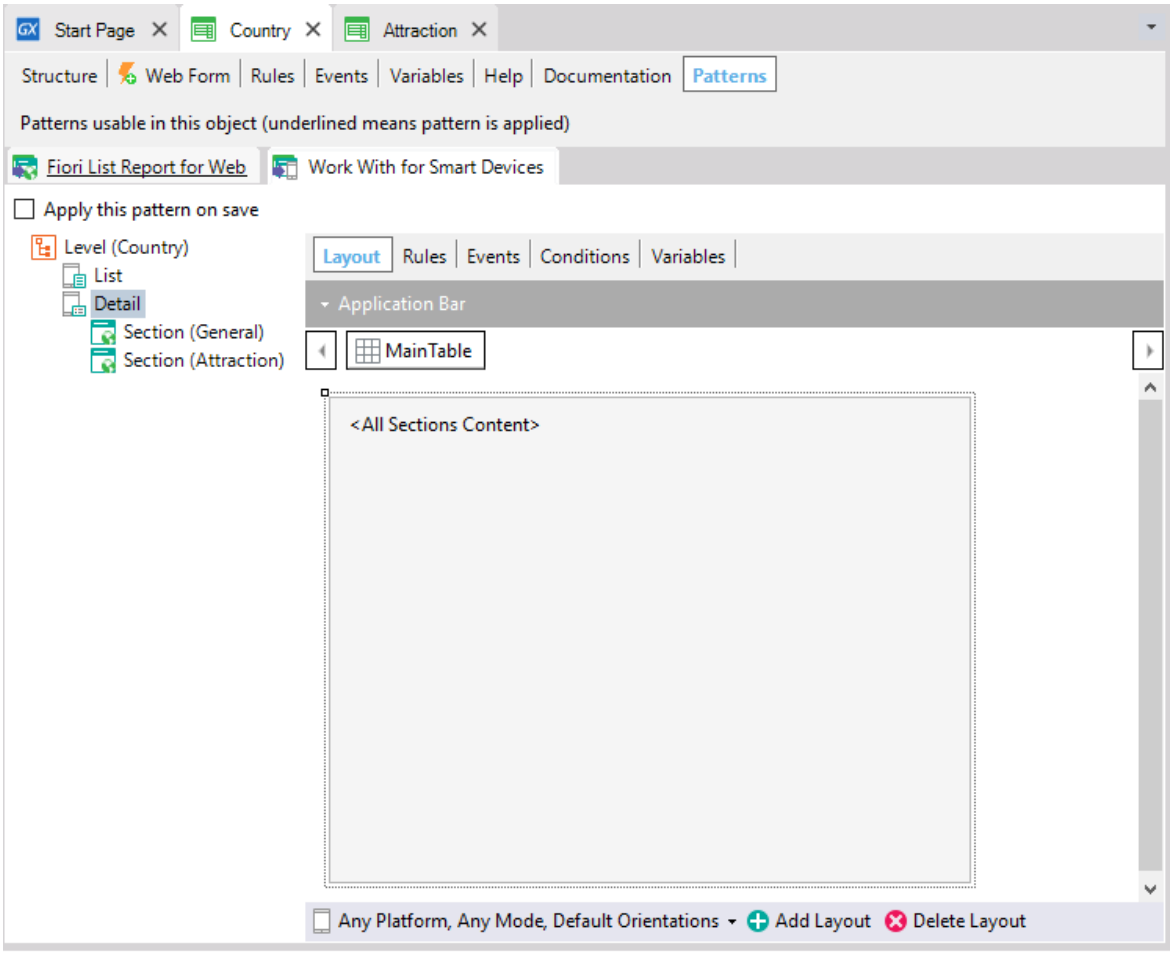
La gran diferencia es que aquí estos nodos corresponderán directamente a los objetos que se generarán y el desarrollador no “declarará” para que luego GeneXus implemente, sino que implementará directamente. Así, si nos posicionamos sobre el nodo List:



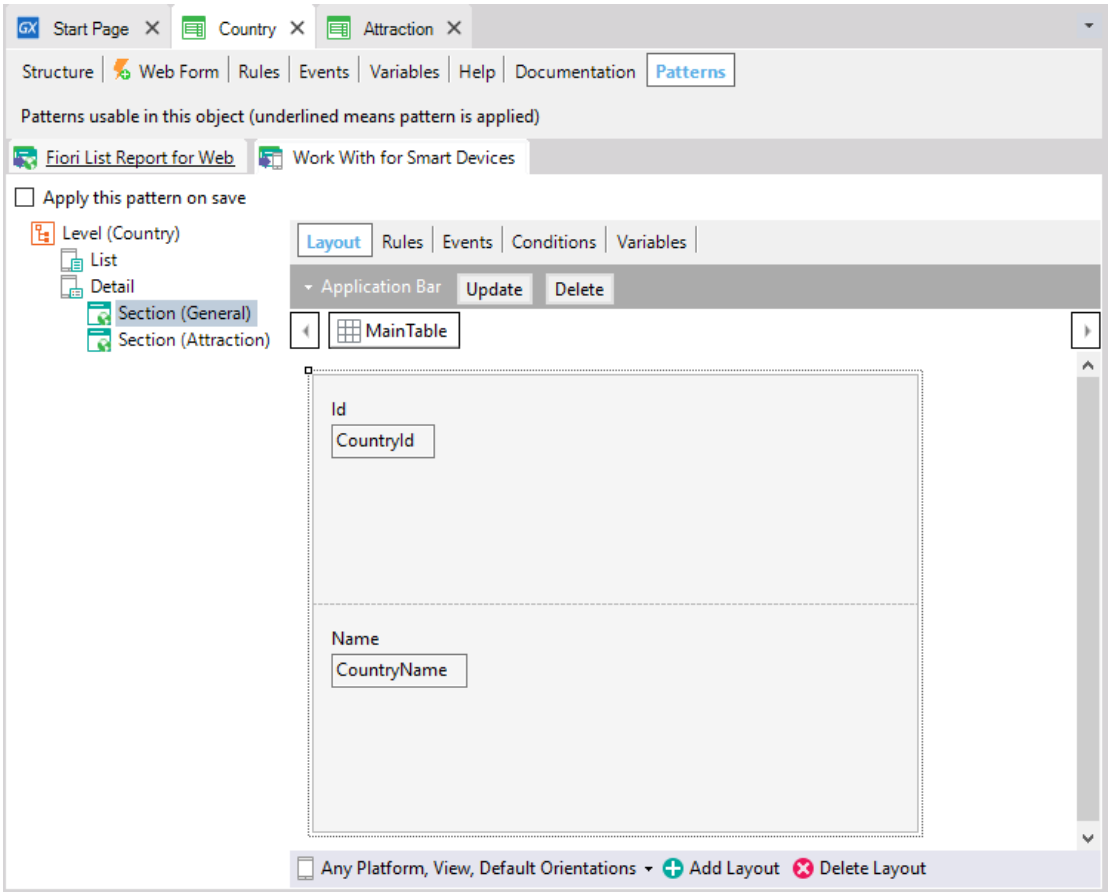
Vemos su Layout ya inicializado, con un grid que mostrará el valor del atributo CountryName de cada país a ser cargado de la base de datos.

Además vemos que se habilitan las secciones Rules, Events, etc, para definir reglas, eventos, y demás aspectos de este objeto. Es decir, aparecen las secciones que permiten implementarlo.

Por otro lado, si vamos al nodo Detail:

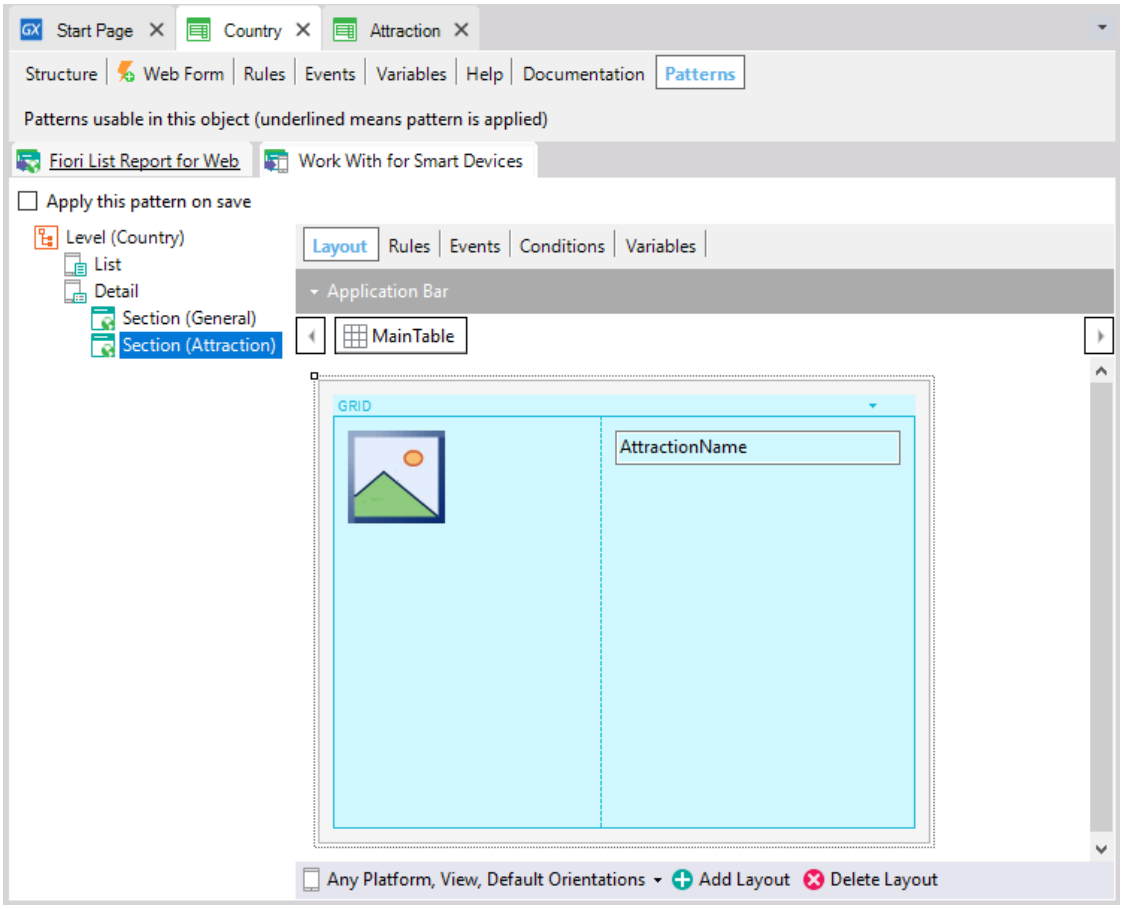


Vemos que el objeto contiene en su pantalla únicamente un control llamado <All Sections Content>. En este control se cargarán todas las secciones contenidas en este Detail. Aquí son dos: la que muestra la información general del país:



Y allí vemos los atributos CountryId y CountryName.

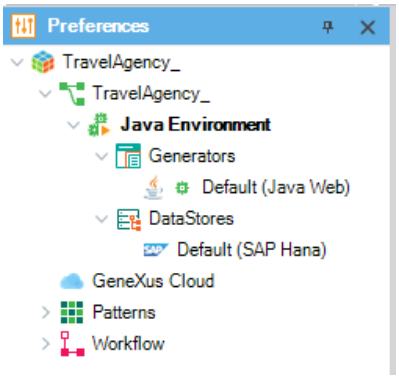
Y la que muestra las atracciones del país en un grid:



Donde se ubicaron los atributos AttractionPhoto y AttrractionName.

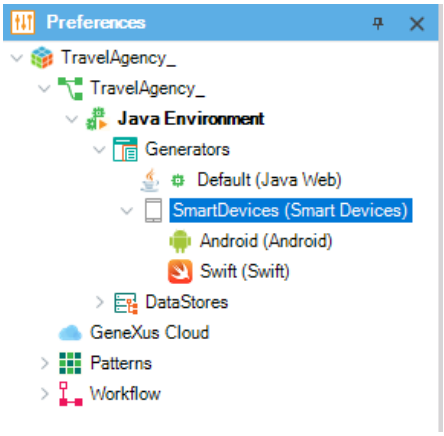
Nos quedará más claro al ejecutar.

Marquemos el check box para aplicar este patrón, y antes de grabar, observemos los generadores (de programas) que tenemos definidos. Por ahora uno sólo: el de Java Web.

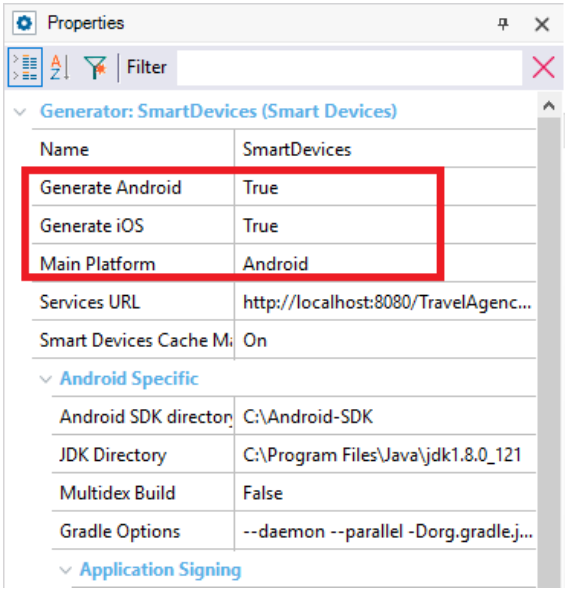


Ahora sí, grabemos.

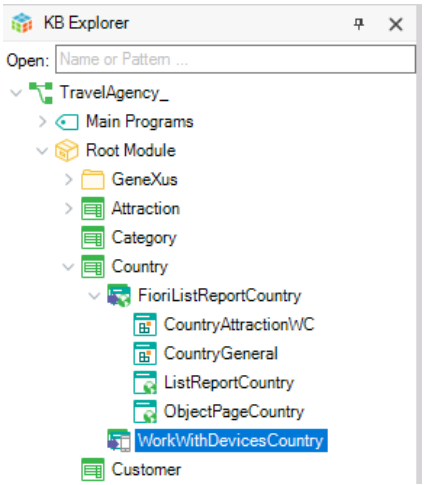
Se ha agregado automáticamente el generador de Smart Devices:



En el que por defecto se generará tanto para Android como para iOS, siendo la plataforma principal Android:



Por otro lado, observemos que bajo el nodo Country aparece ahora **el objeto** WorkWithDevicesCountry:



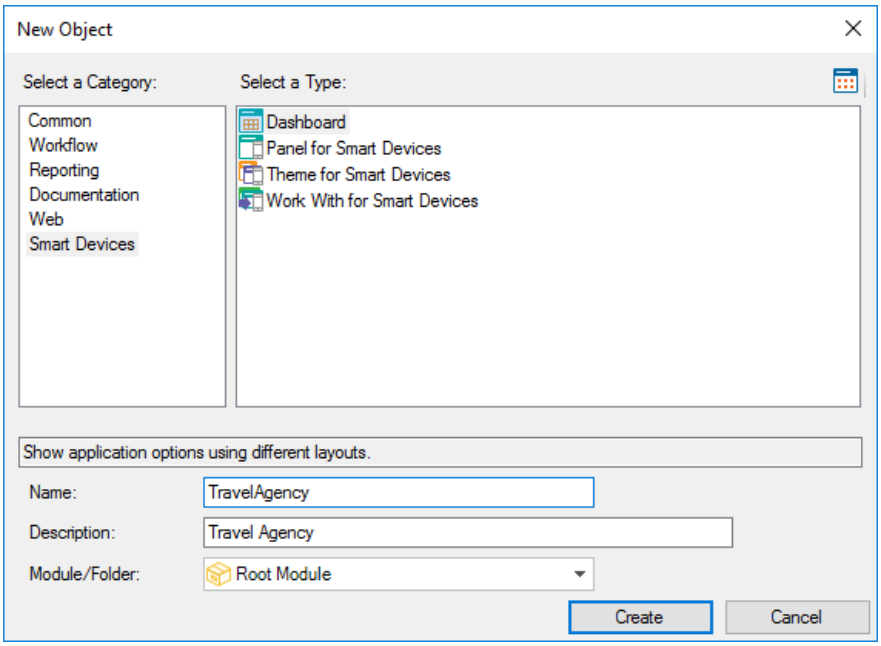
A diferencia del caso web, en el que FioriListReportCountry era un archivo de declaración exclusivamente, y los objetos que efectivamente implementan las funcionalidades del patrón se construyen aparte (los vemos listados abajo), en este caso se trata de un objeto GeneXus, que contendrá sub-objetos que pueden invocarse independientemente, con la sintaxis adecuada.

Apliquemos también el pattern a la transacción Attraction, y a Category.

¿Cómo probamos esto de forma rápida? Una opción es tener un dispositivo conectado a la computadora en la que estamos utilizando GeneXus. Otra es utilizar los emuladores que brindan las plataformas. Por simplicidad, veremos esta opción. (Para ver todas las opciones de prototipación, le recomendamos acceder al Curso "GeneXus for Smart Devices". Siempre será mejor probar con un dispositivo real, pues los emuladores no cuentan con todas las funcionalidades)

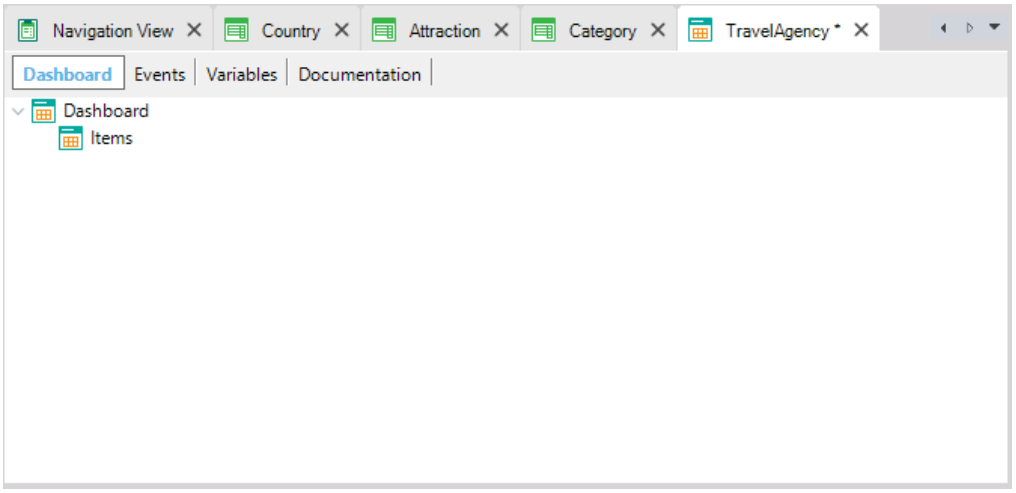
Nos estaría faltando tener un menú —como en web teníamos el Launchpad— para invocar a los Lists que nos interesen. Este menú **no** es creado automáticamente por GeneXus. Debemos crearlo nosotros.

File/New/Object, filtramos por los objetos para Smart Devices, y elegimos el Dashboard, que es un objeto para implementar menús. Le llamamos TravelAgency:

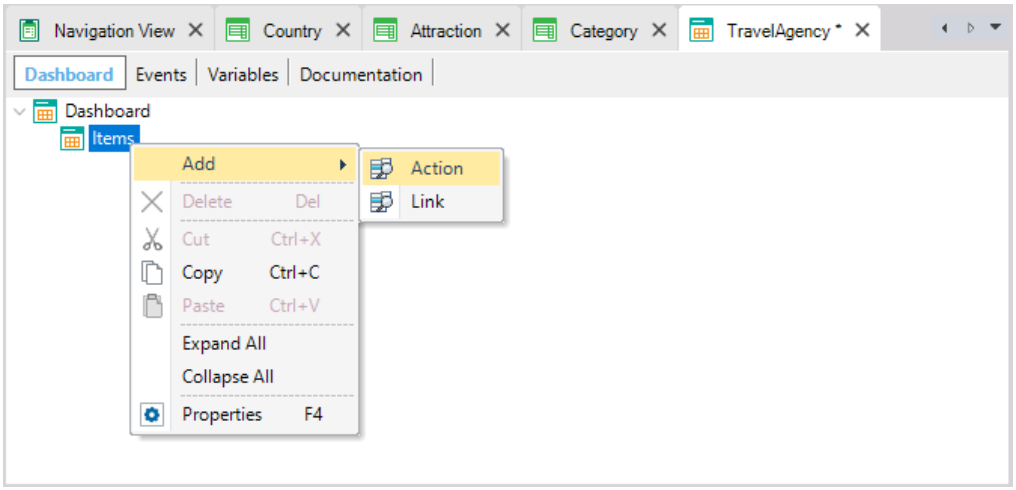


Este será el objeto **main** de la aplicación para Smart Devices. Será el objeto que GeneXus compilará para Android y/o para iOS y que el usuario final, una vez terminado el desarrollo, se instalará en su dispositivo.

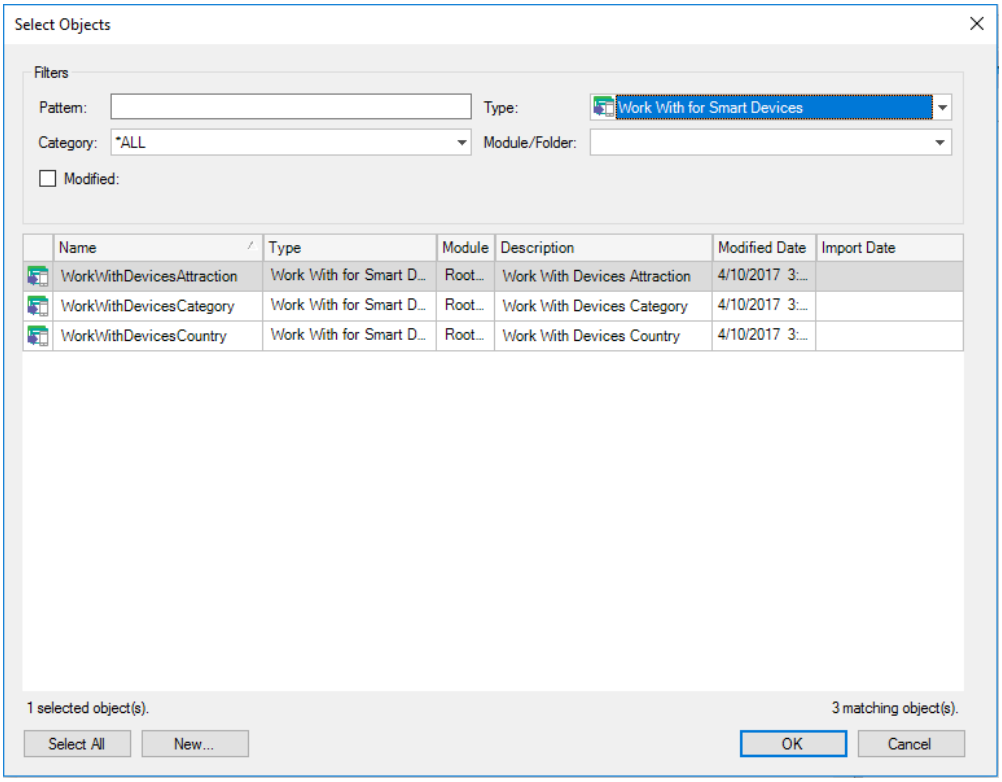
Ingresemos los ítems que queremos que compongan el menú:



El primero corresponderá a la acción de invocar al List de countries:



Se nos abre esta ventana para seleccionar el objeto que queremos invocar cuando el usuario elija este ítem del menú. Filtramos por tipo de objeto Work With for Smart Devices:

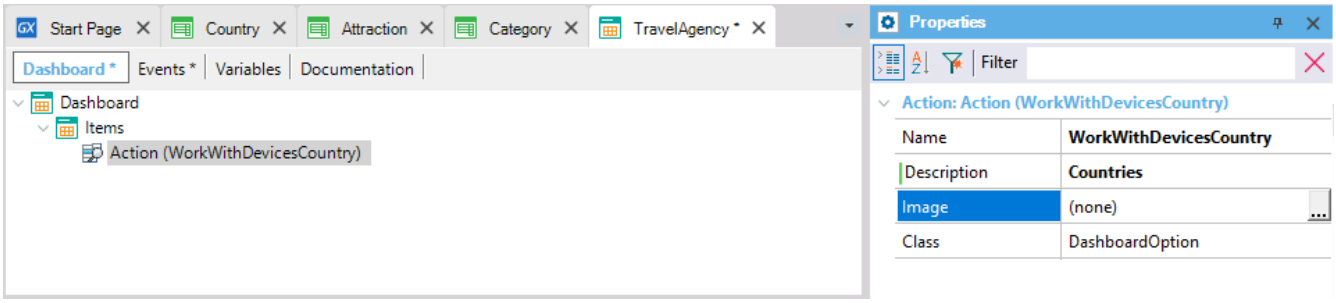


Y elegimos el WorkWithDevicesCountry.

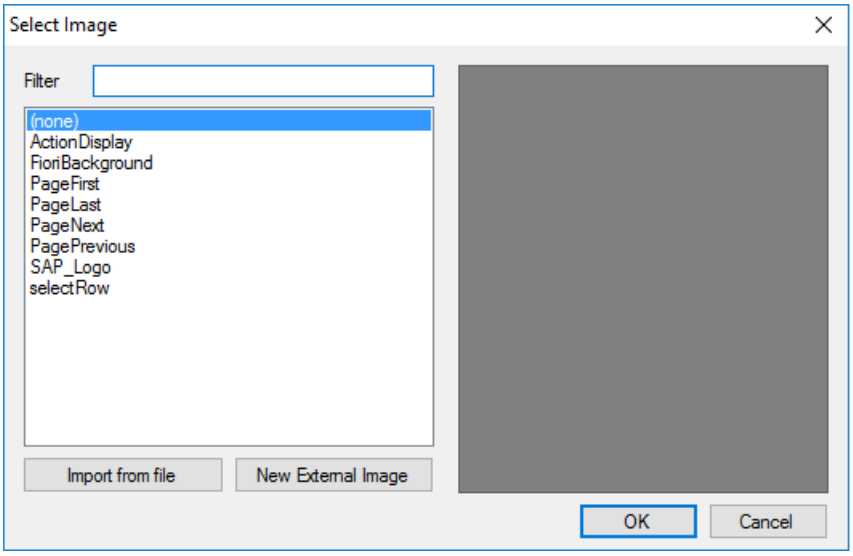
Vemos que aparece la acción, que por defecto recibe el mismo nombre que el objeto invocado. Esta acción —hacemos doble clic— tendrá asociado un evento que programa la invocación:

Allí vemos que se está invocando al List del objeto WorkWithDevicesCountry (y no al Detail).

Vayamos a las propiedades de la acción, y cambiemos la Descripción del ítem, que será la que veremos en el menú en ejecución:



Y también elegiremos una imagen para esta opción, teniendo en cuenta que las apps para Smart Devices suelen ser mucho más iconográficas que las apps web, debido al reducido espacio de las pantallas:



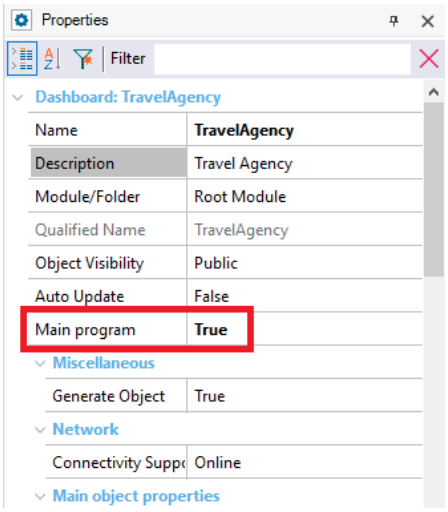
Debemos importar la imagen en la base de conocimiento. Lo haremos a partir de un archivo externo.

Aquí lo elegimos.

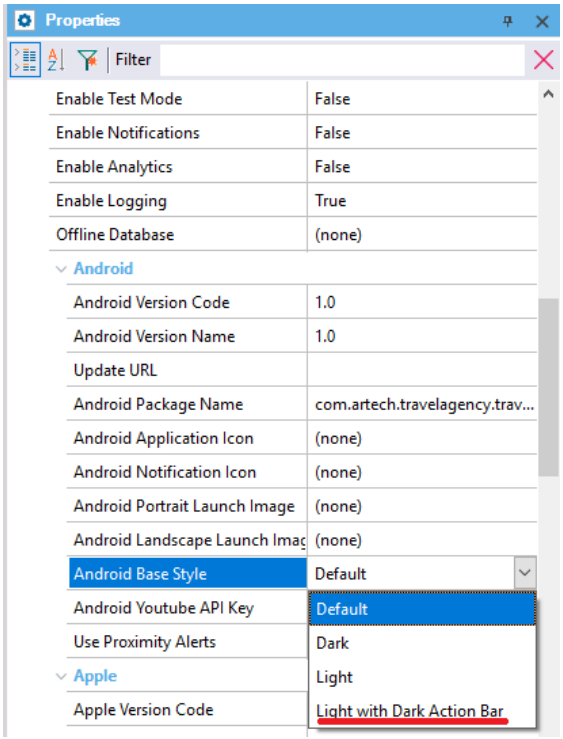
Bien, ahora agreguemos también un ítem para invocar al List de atracciones.
Modifiquemos análogamente sus propiedades.

Grabemos.

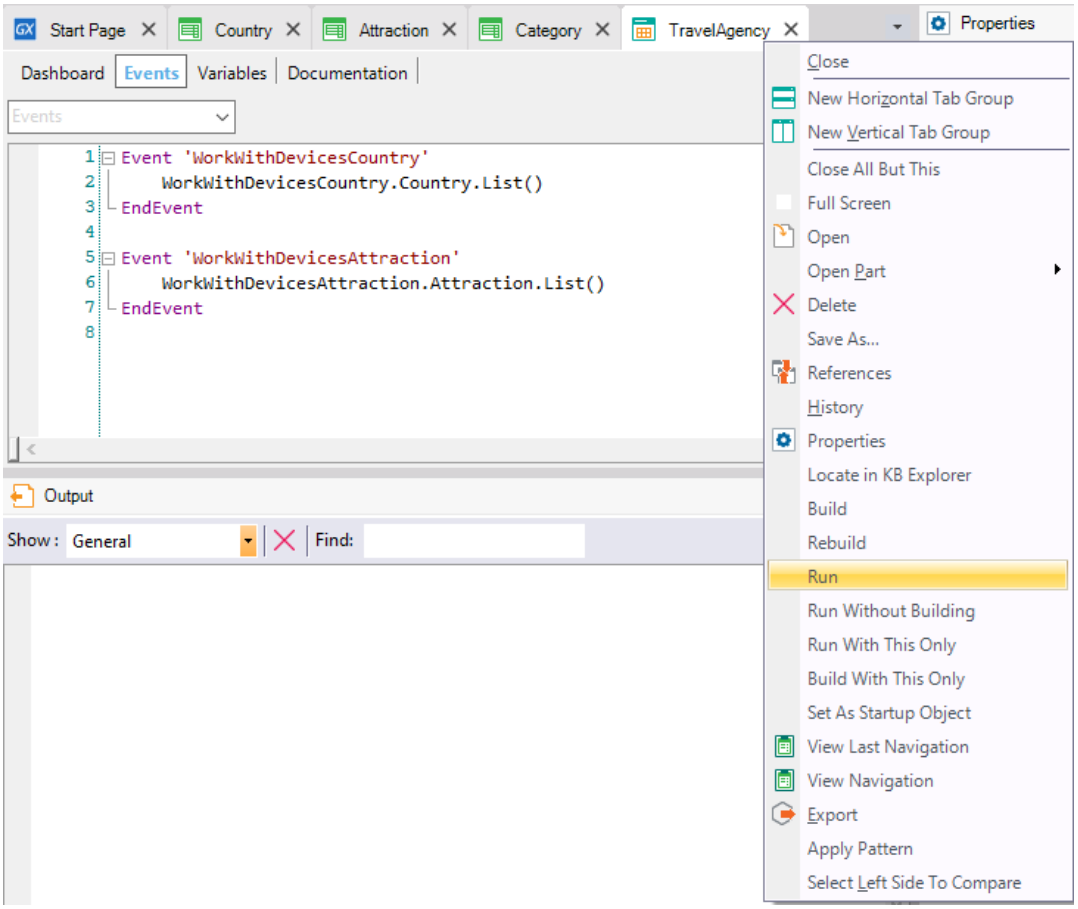
Podemos ver que el objeto está definido como main:



Antes de ejecutarlo, cambiemos la propiedad para Android, “Base Style”, por esta… para que el fondo de la pantalla sea blanco con “Action Bar” oscura:



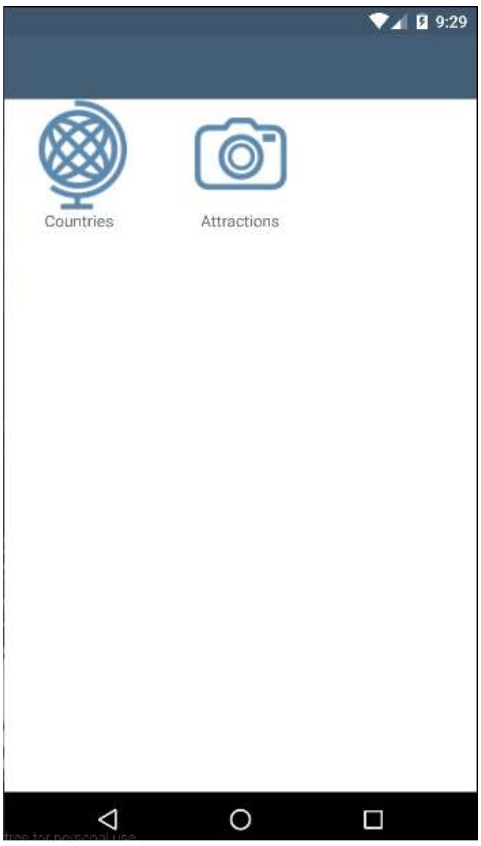
Para ejecutarlo, bastará con hacer botón derecho sobre su solapa y Run:



Si usted tiene un emulador como el Genymotion abierto, el archivo compilado se instalará automáticamente allí y se ejecutará.

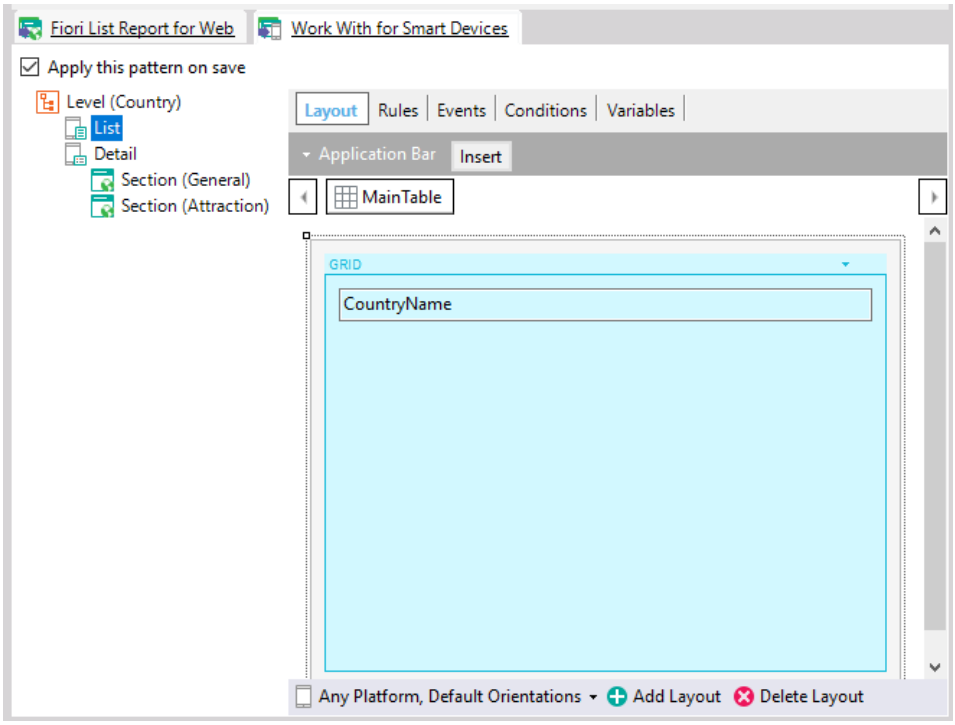
De lo contrario se le abrirá el emulador default del SDK de Android, y sucederá lo mismo. Para no instalarse nuevo software, le sugerimos esta opción.

En este caso teníamos Genymotion abierto, por lo que aquí vemos el dashboard en ejecución, con los dos ítems que definimos, con sus imágenes y descripciones:

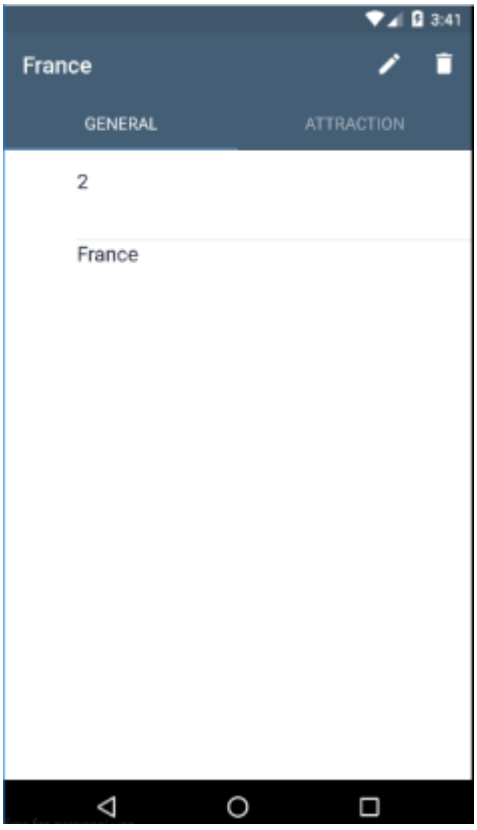


Al hacer tap sobre “Countries”, vemos el List de países, que de cada país muestra en un grid el nombre:

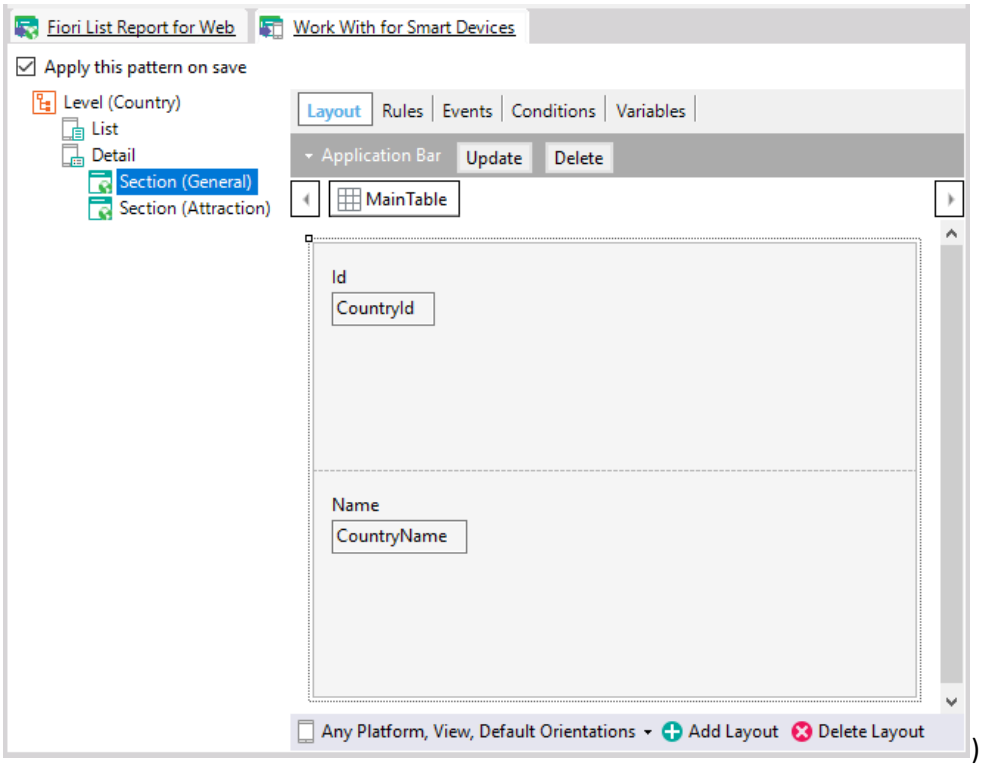
Tal como estaba implementado:



Si hacemos tap sobre un país:

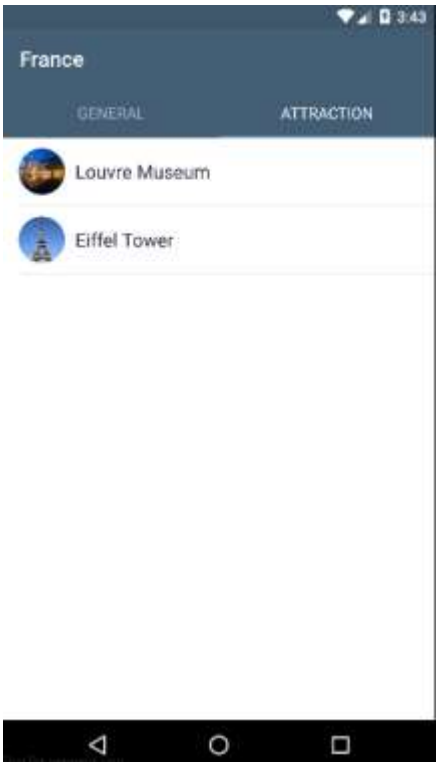


Vemos el detalle, con dos tabs: la información general en un tab…
(lo que está implementado aquí:

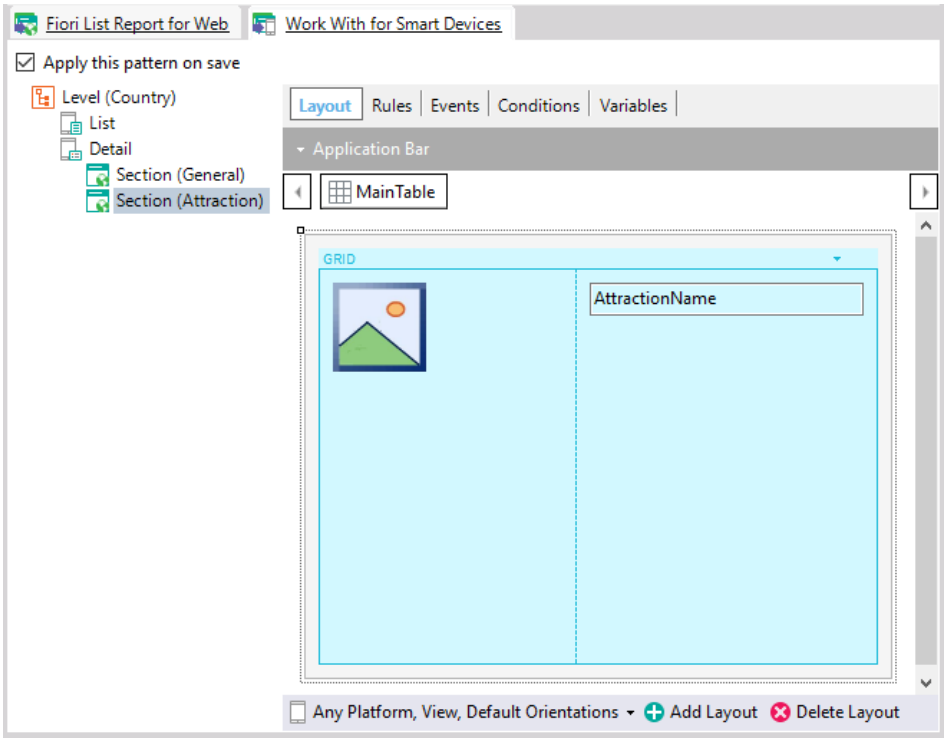


)

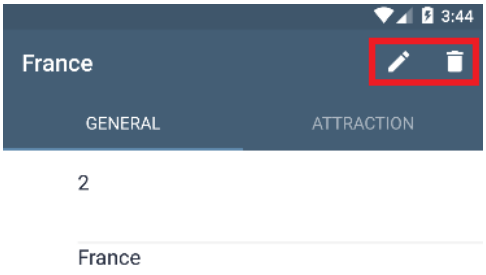
y las atracciones del país en el otro:



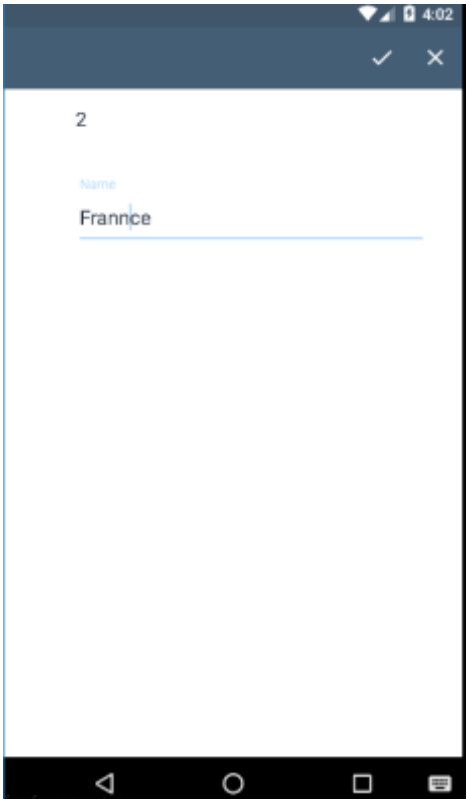
De las atracciones vemos la foto y el nombre, tal como está implementado aquí:



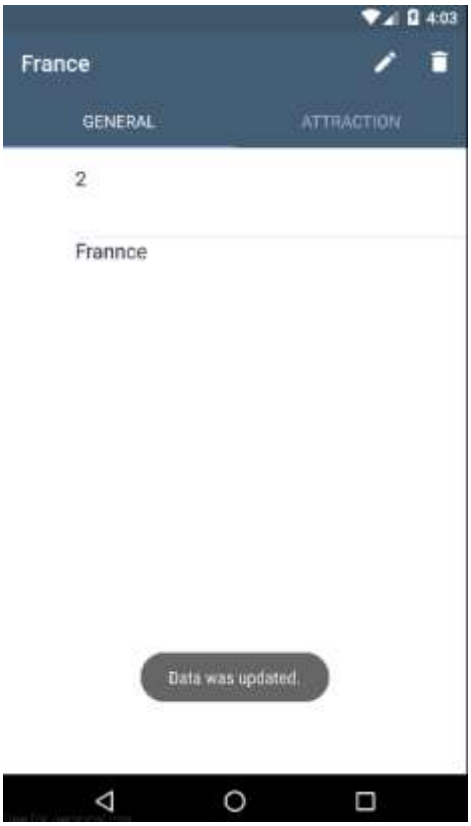
Observemos que desde el tab General podemos modificar la información del país, o eliminarla:



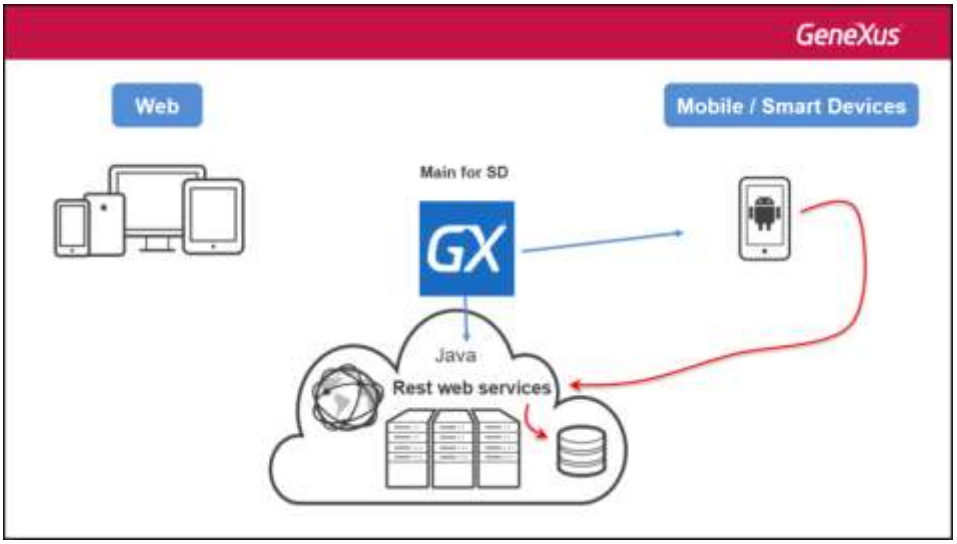
Cambiemos el nombre…



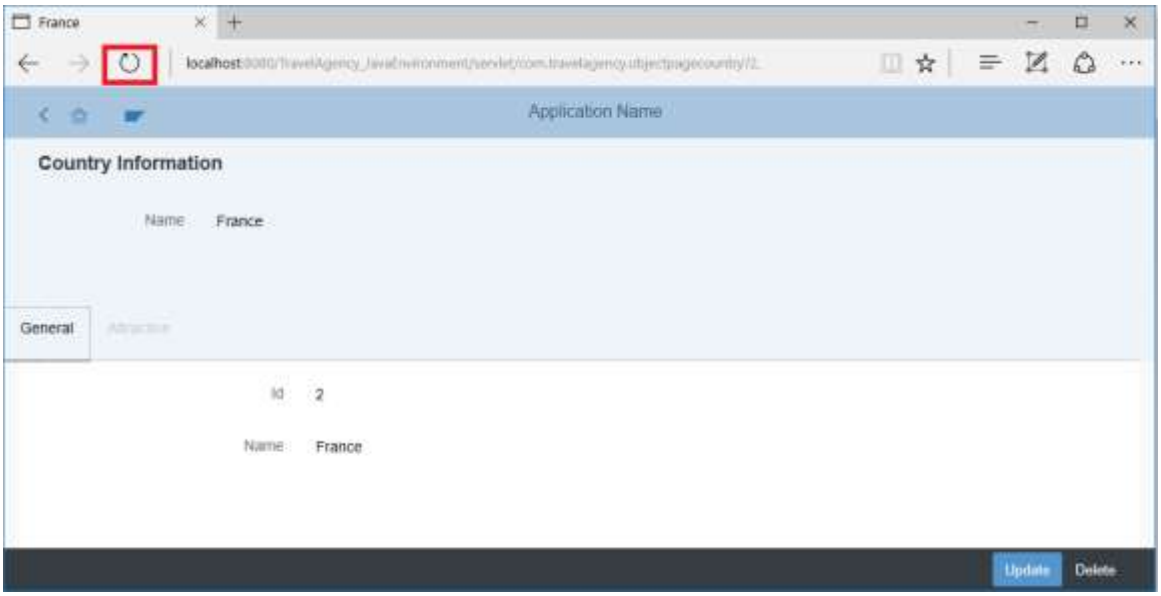
Confirmamos:



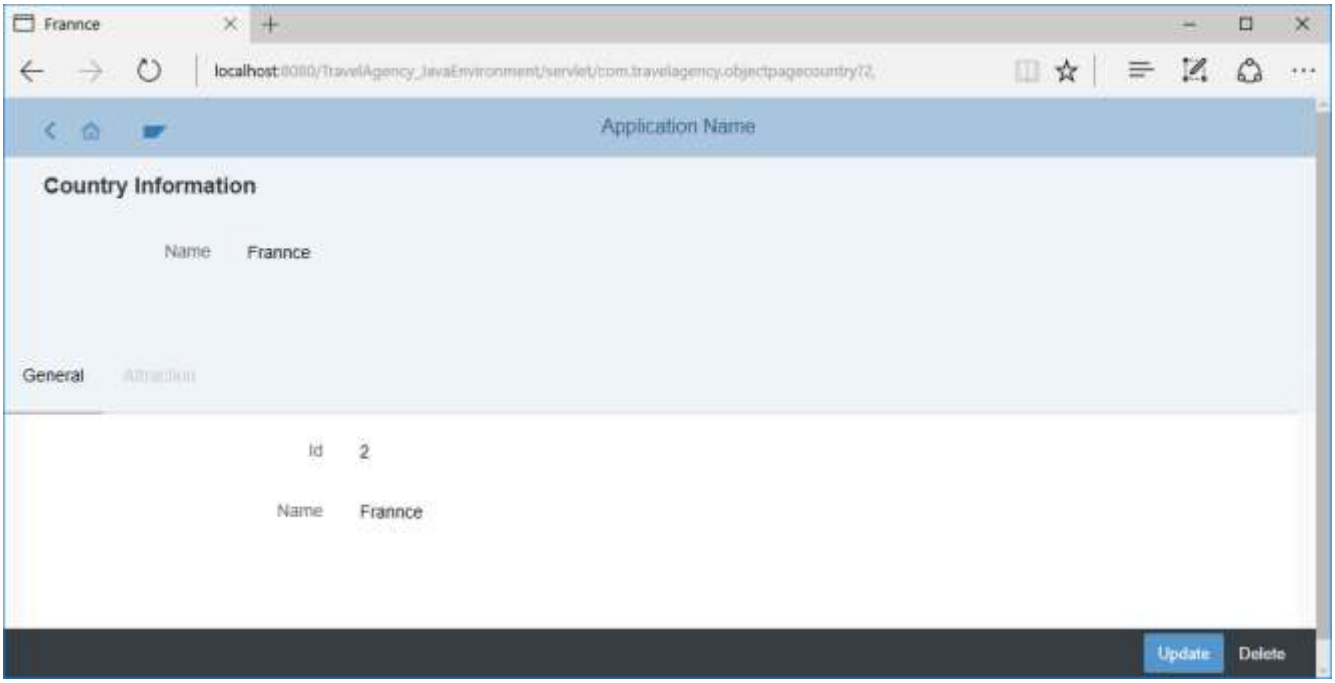
El objeto que realizó la actualización en la base de datos fue uno de los Servicios Rest que mencionamos al principio.



Así, si ahora volvemos a ejecutar la aplicación web:



Refrescamos la página:

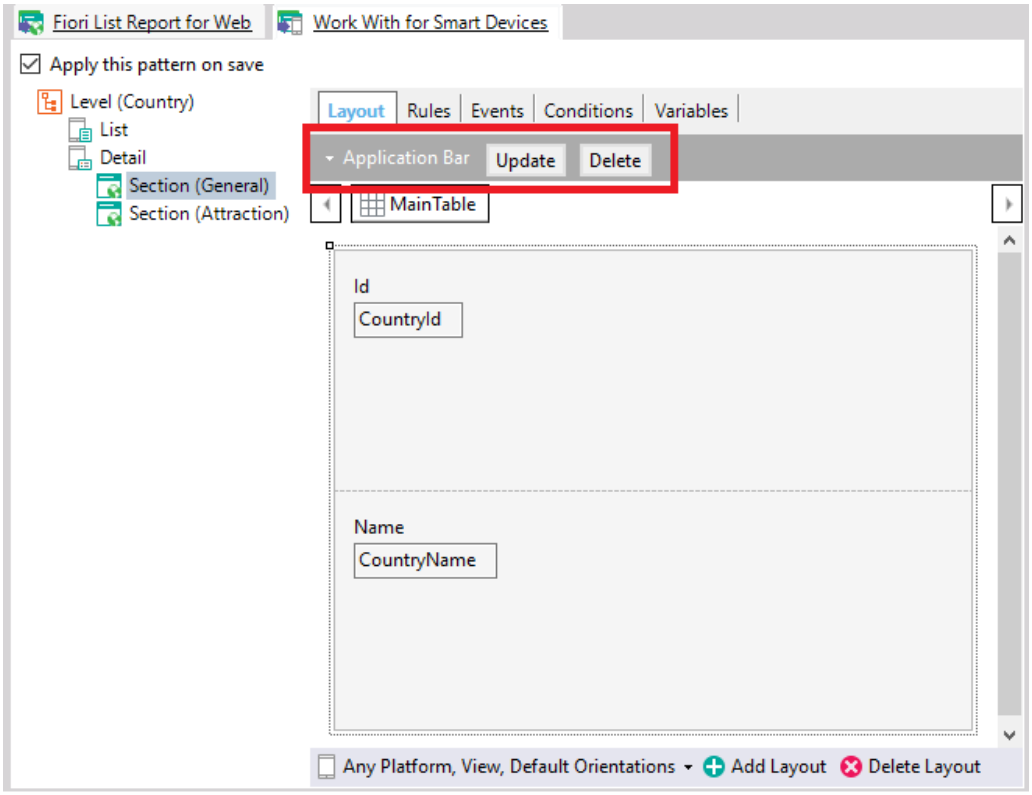


Y ¡voilá!

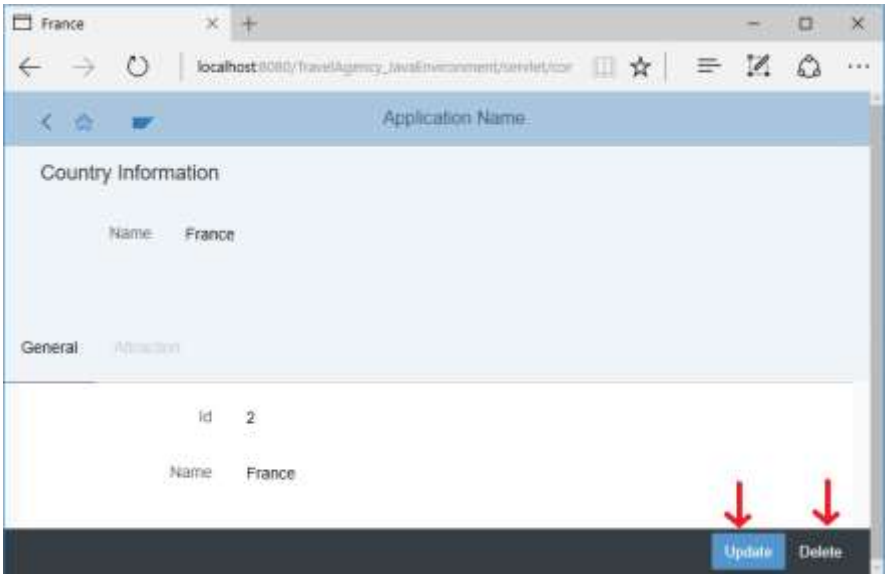
De la misma forma, si desde aquí volvemos a modificar el nombre de Francia… y ahora volvemos al dispositivo Android:

Aparece actualizada.

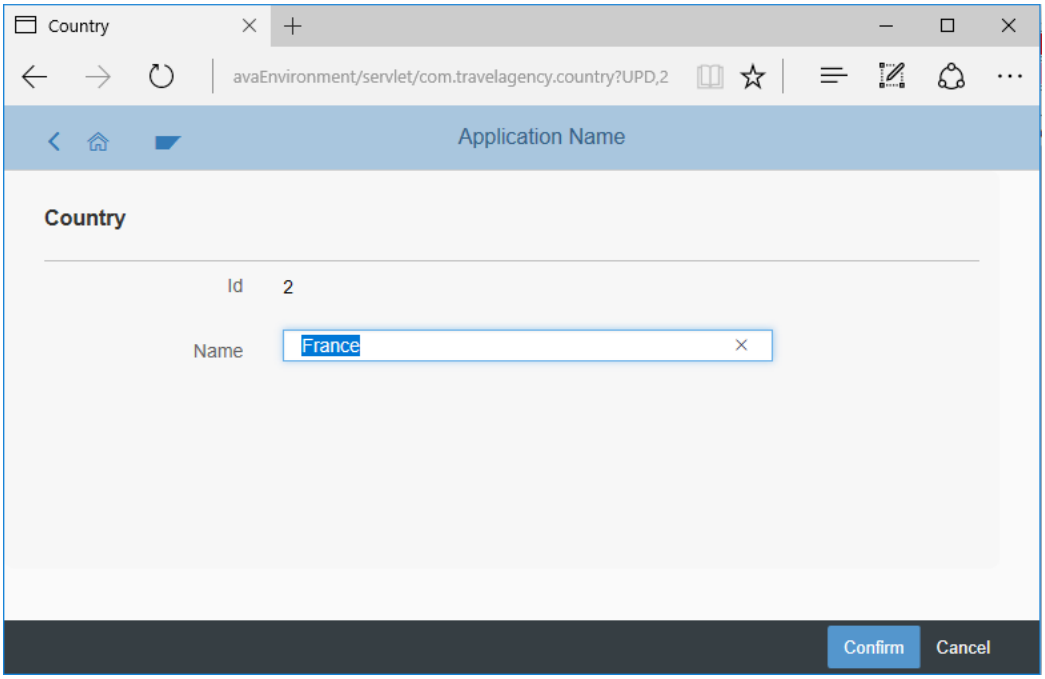
Los dos botones de update y delete del país aparecen en el WorkWith aquí, en la zona de la application bar:



Pero, mientras en el patrón web se invocaba a la transacción al querer actualizar o eliminar:

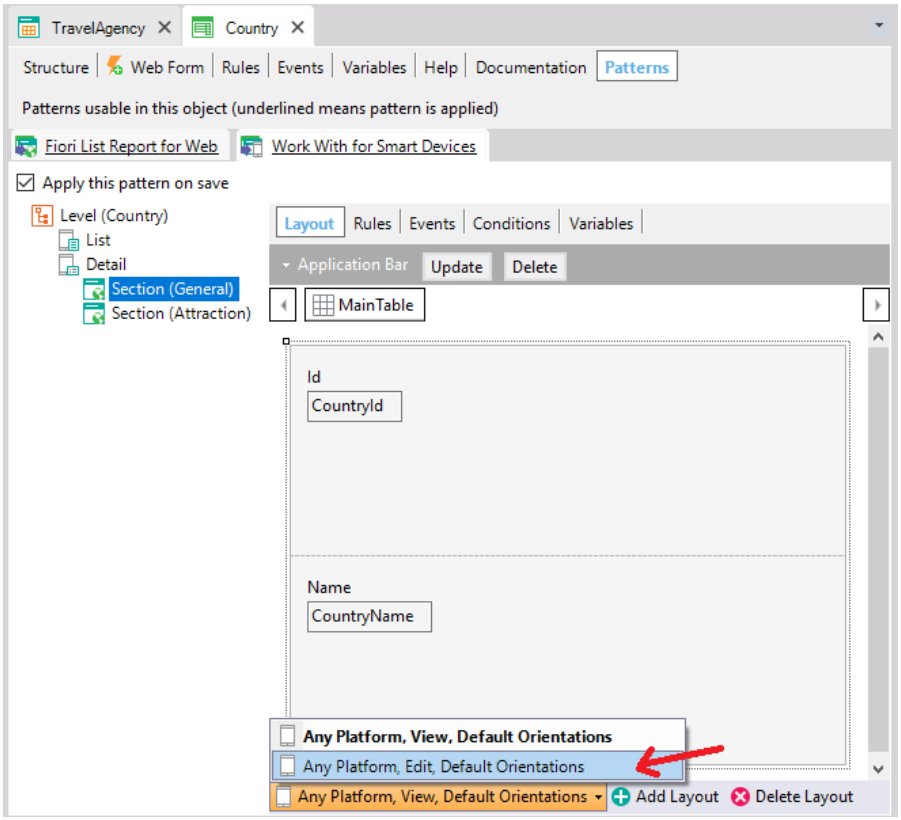


(Por ejemplo, aquí se la invoca en modo UPDATE:

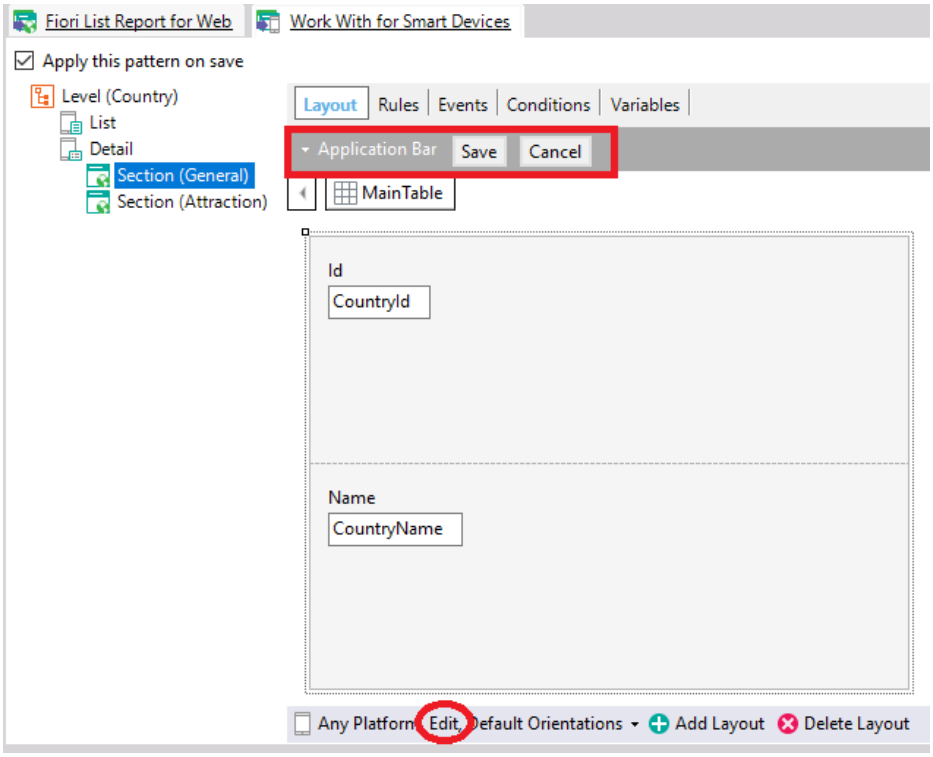


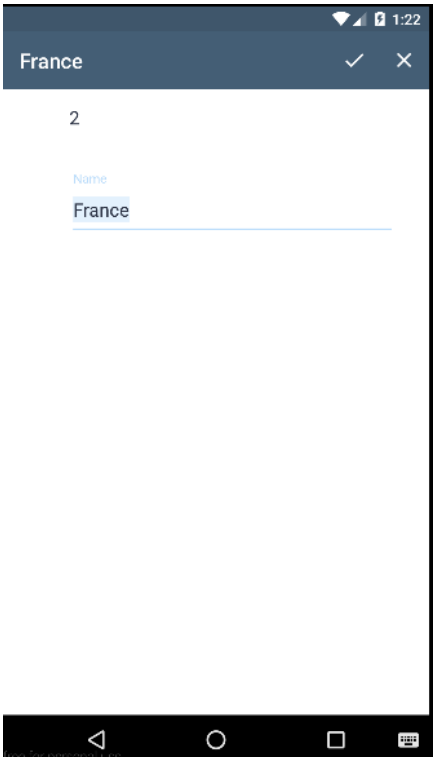
Y lo que estamos viendo es el form de la transacción)

…en aplicaciones nativas móviles se abre el mismo Detail, pero con otra pantalla: la pantalla de edición:



Que se inicializa idéntica, a excepción de los botones Save y Cancel en la Application bar:





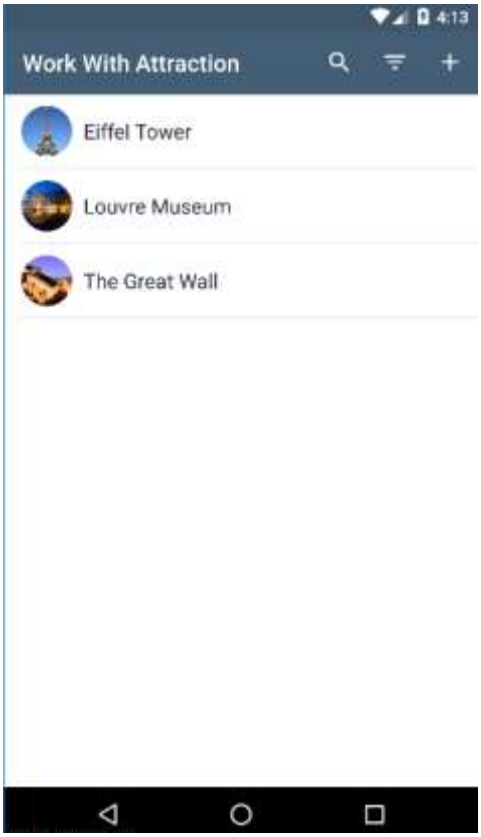
Cuando el usuario presione el botón de Save, la app en el dispositivo invocará, pasando los datos de esta pantalla, al servicio Rest en el servidor web que contiene la lógica de la transacción, y por tanto actualizará la información en la base de datos. Todo esto se hace de manera transparente para el desarrollador, que no tendrá que ocuparse de estos detalles de bajo nivel.

Igual que para el patrón web, si volvemos al List, vemos que tenemos la opción de agregar un nuevo país:

Aquí se invoca a la misma pantalla de Detail que para el Update.

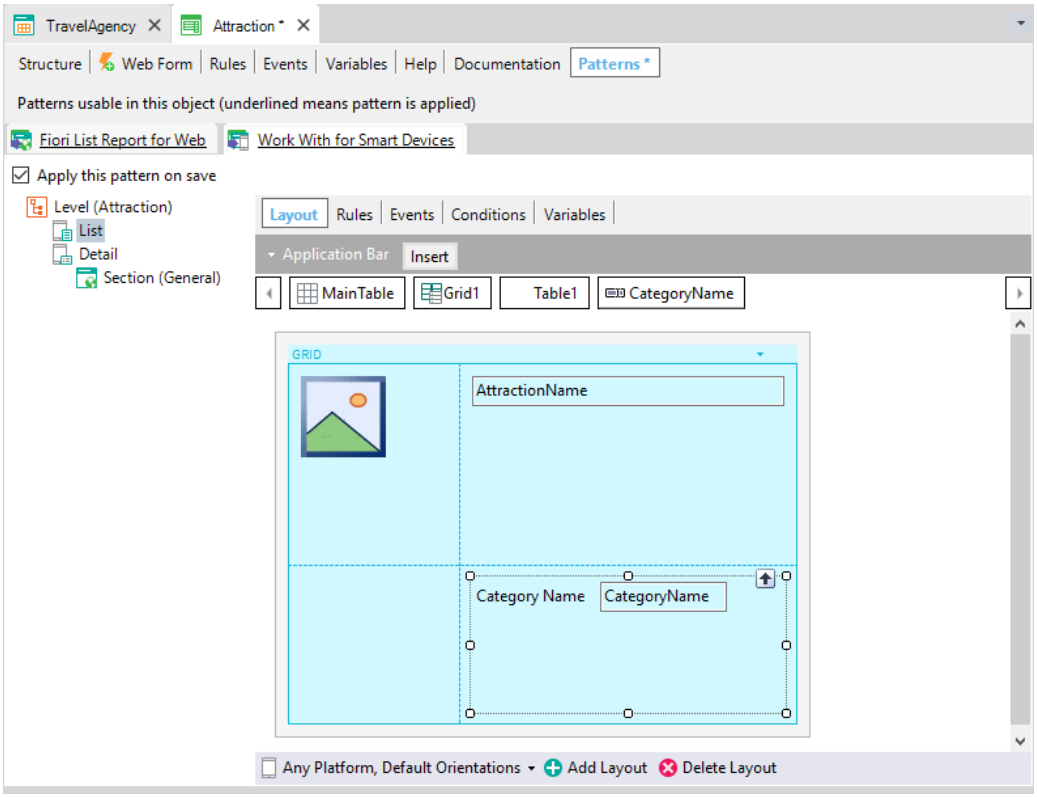
Agreguemos México:

Esto es lo que genera por defecto el patrón. Pero puede personalizarse.
Si vamos al List de Attractions:

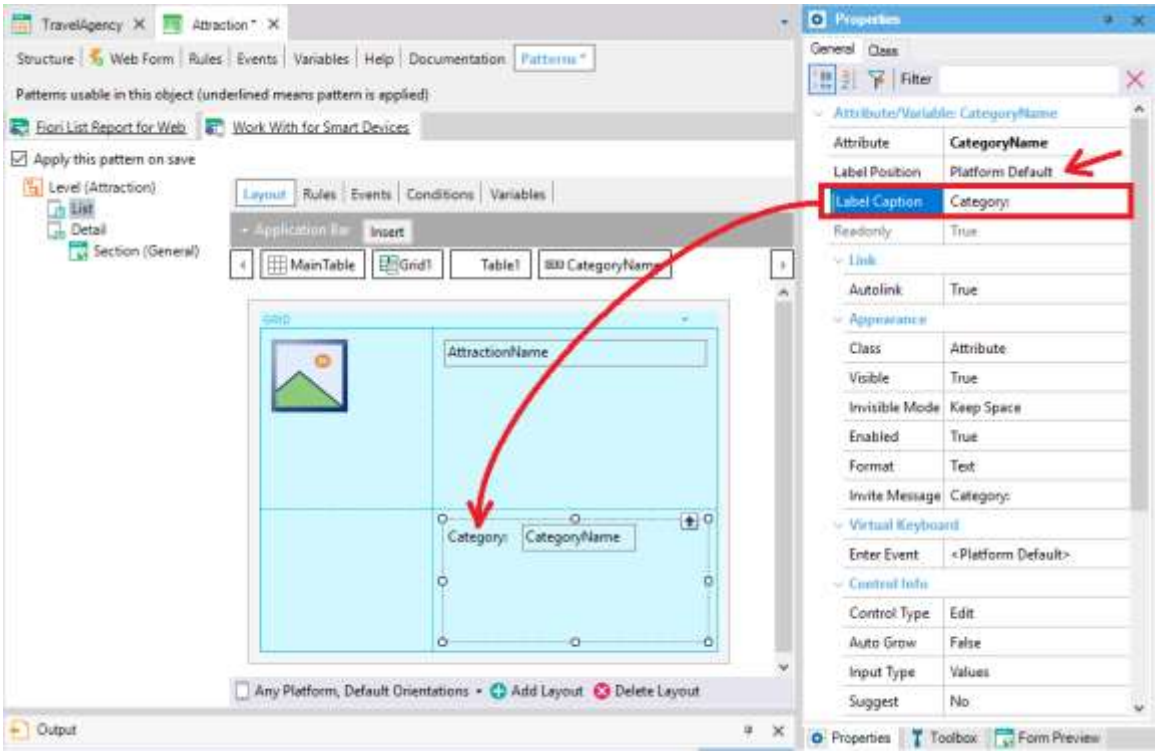


Vemos que está mostrando en el grid la imagen de la atracción y su nombre. Podríamos querer agregar también la categoría de la atracción.

Para ello, vamos a la toolbox y arrastramos el control Attribute/Variable, y elegimos el atributo CategoryName:

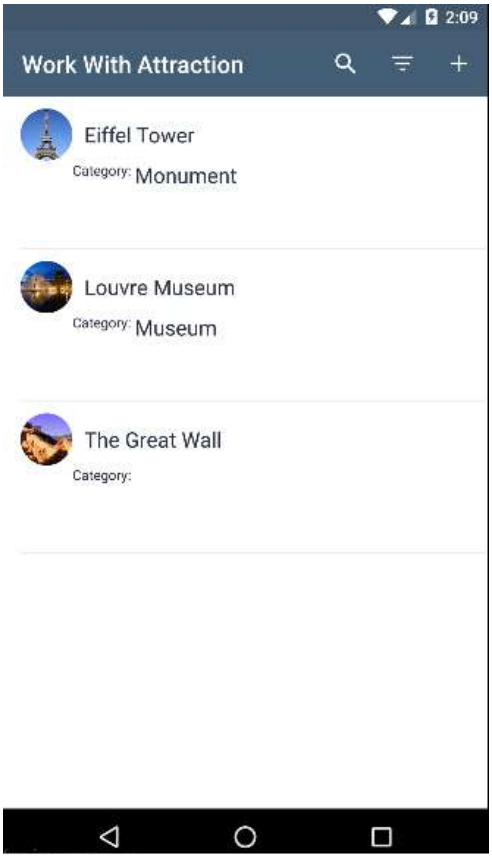


Le modificamos la etiqueta:

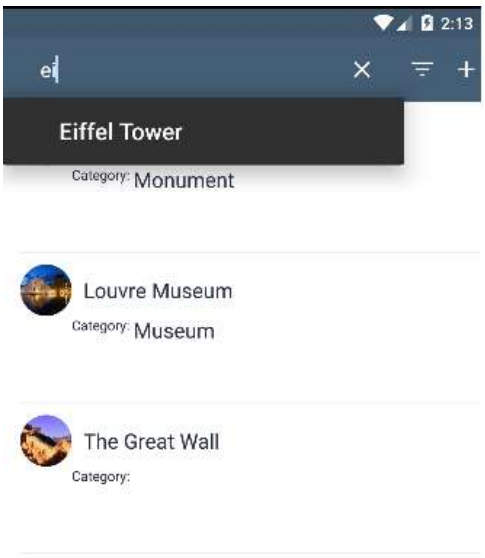


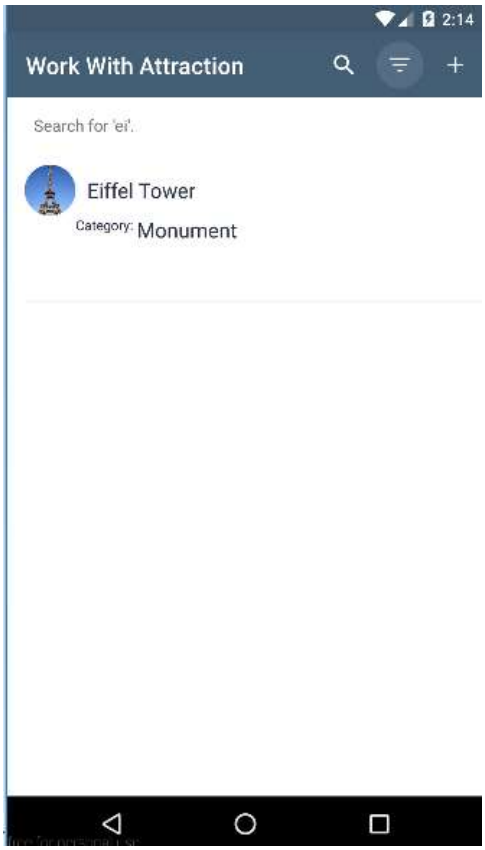
Podríamos eliminarla, con Label position None.

Ejecutemos, haciendo Run sobre el objeto main.

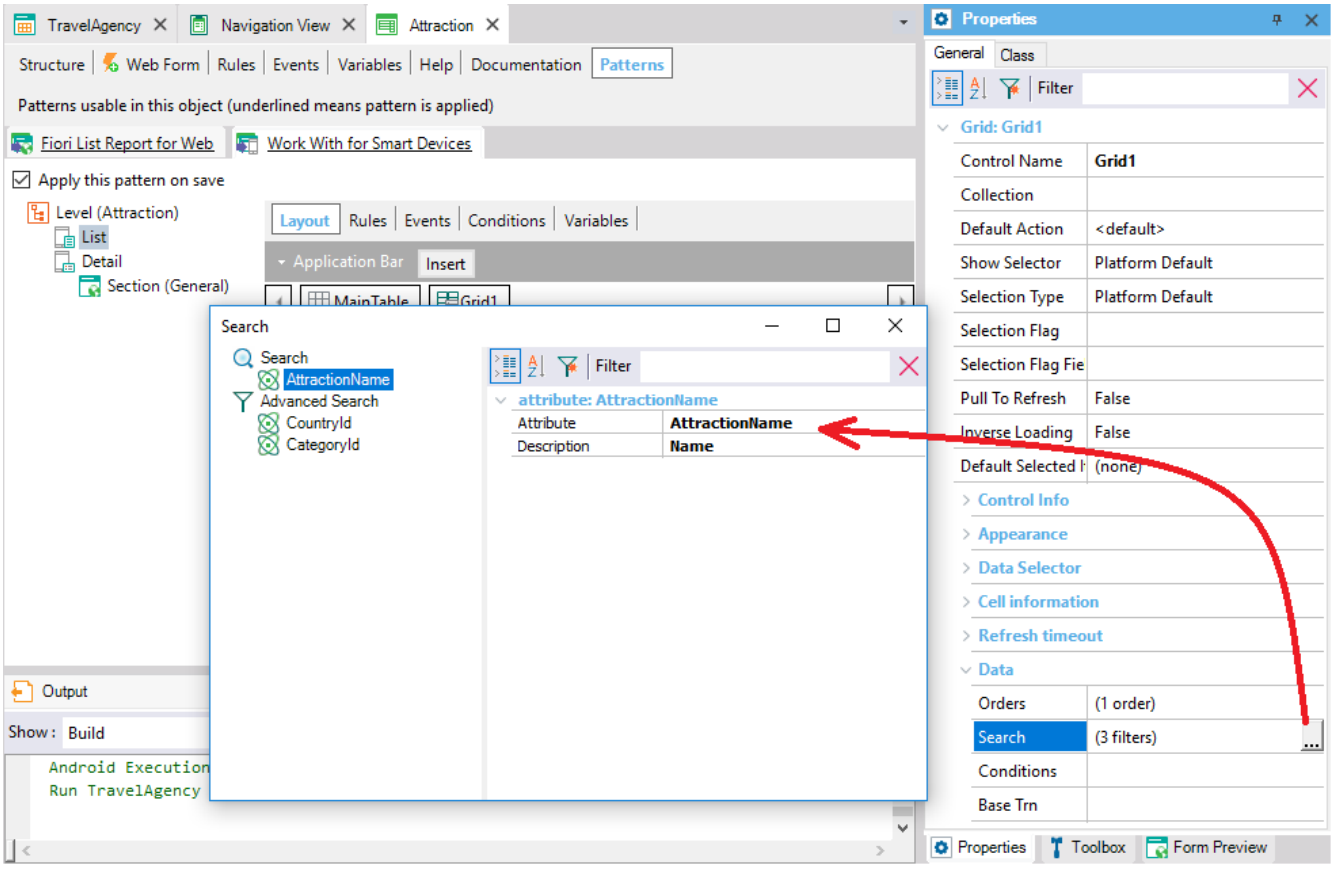


Observemos que por defecto nos ofrece la posibilidad de hacer búsquedas por nombre de atracción:

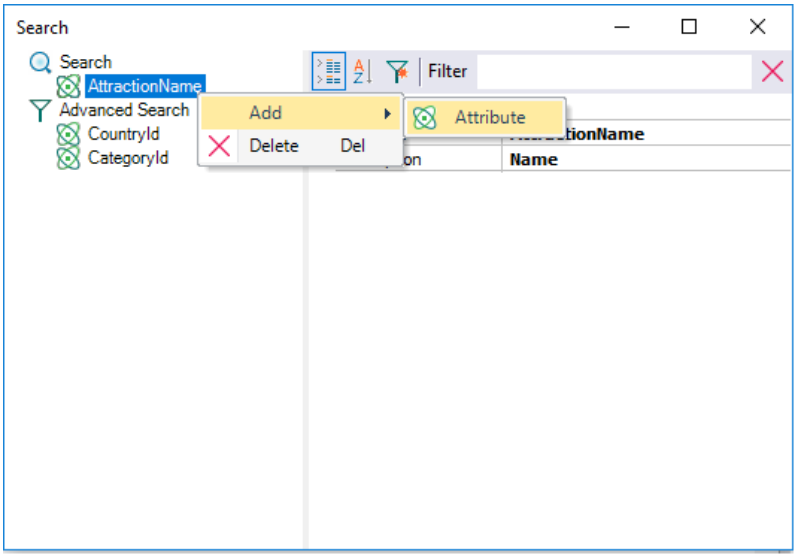

2:13
ei
Eiffel Tower
Category: Monument
Louvre Museum
Category: Museum
The Great Wall
Category:

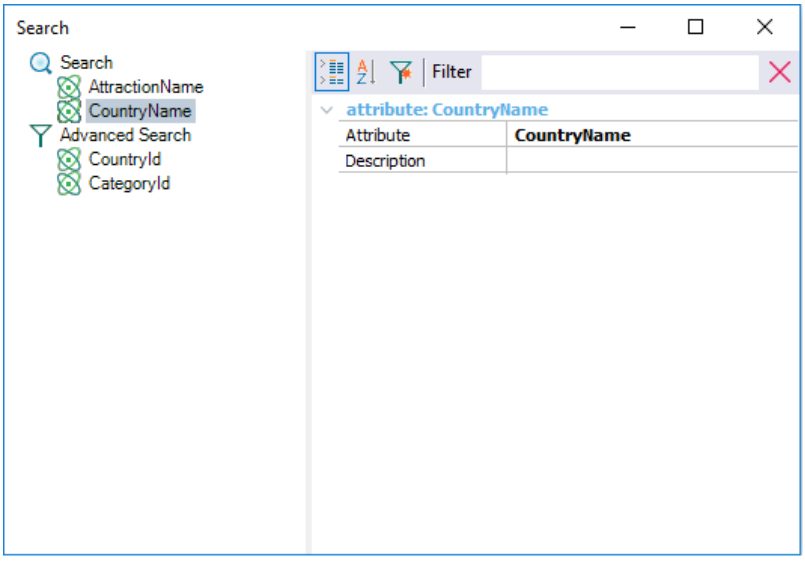

2:14
Work With Attraction
Search for 'ei'.
Eiffel Tower
Category: Monument

Esto está configurado en las propiedades del grid, aquí:

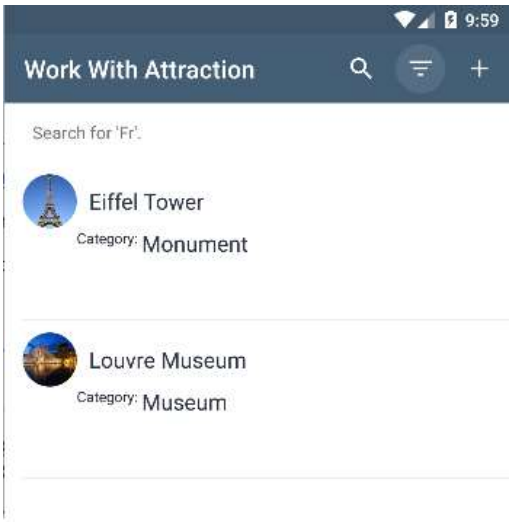


Podríamos agregar CountryName, por ejemplo, para que las búsquedas puedan ser también por nombre de país.



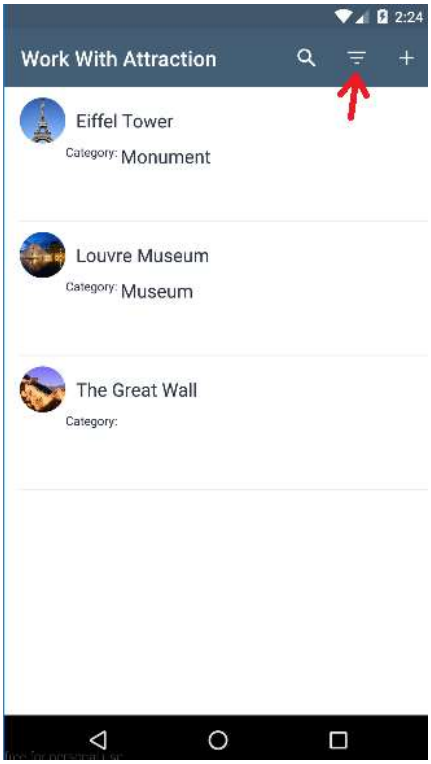


Y entonces, buscando por “Fr”… vemos que aparecen las atracciones de Francia.

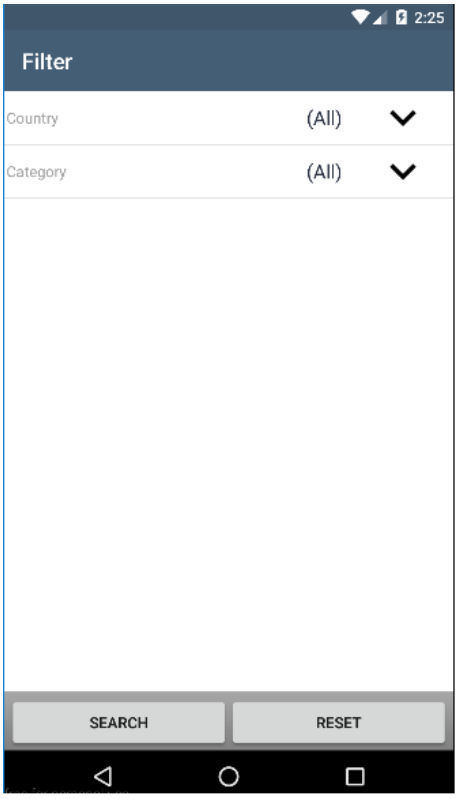


Pero también tenemos filtros avanzados… por los atributos CountryId y CategoryId, que, recordemos, son las claves foráneas que tiene la transacción Attraction. Cambiémosle la descripción por Country y Category respectivamente y ejecutemos el menú principal nuevamente…

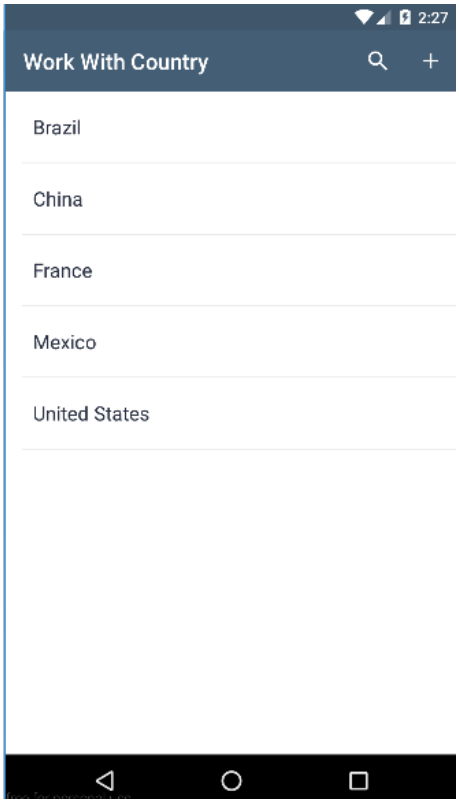
Vamos al listado de atracciones y vemos que tenemos esta opción en la application bar:



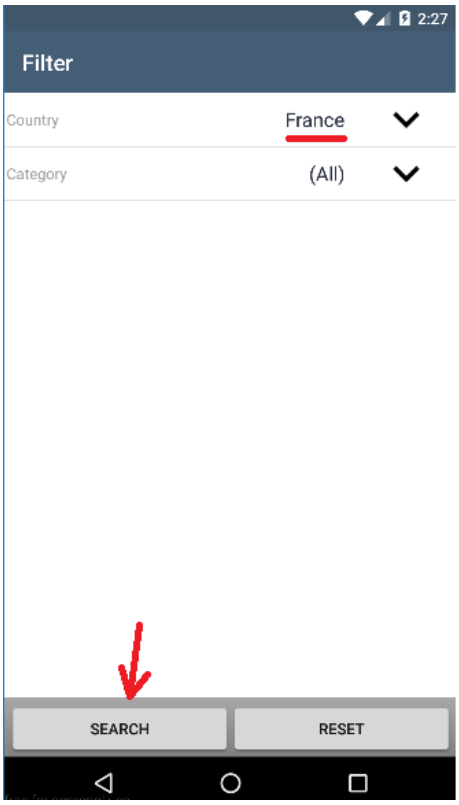
Que justamente nos ofrece los dos filtros avanzados que acabamos de ver:

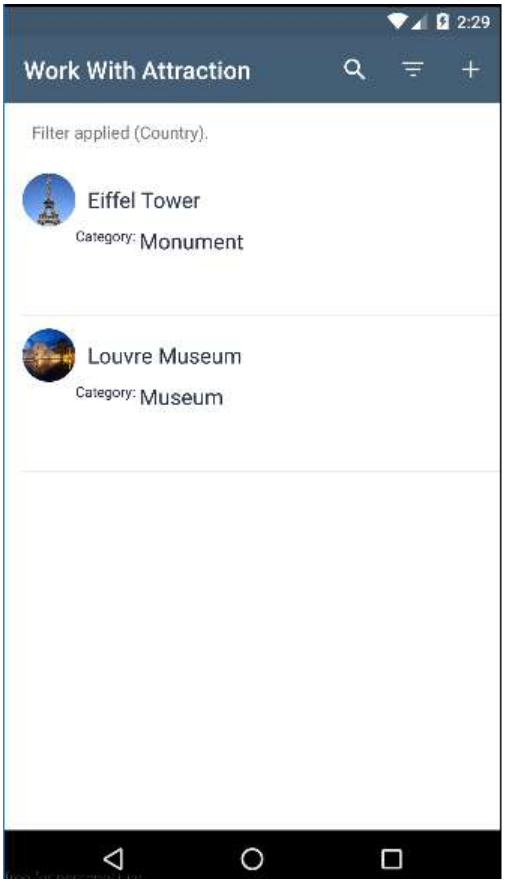


Si queremos filtrar las atracciones por país, por ejemplo, vemos que nos abre el trabajar con países, para que seleccionemos uno de los existentes:



Y una vez hecho esto, presionamos Search:

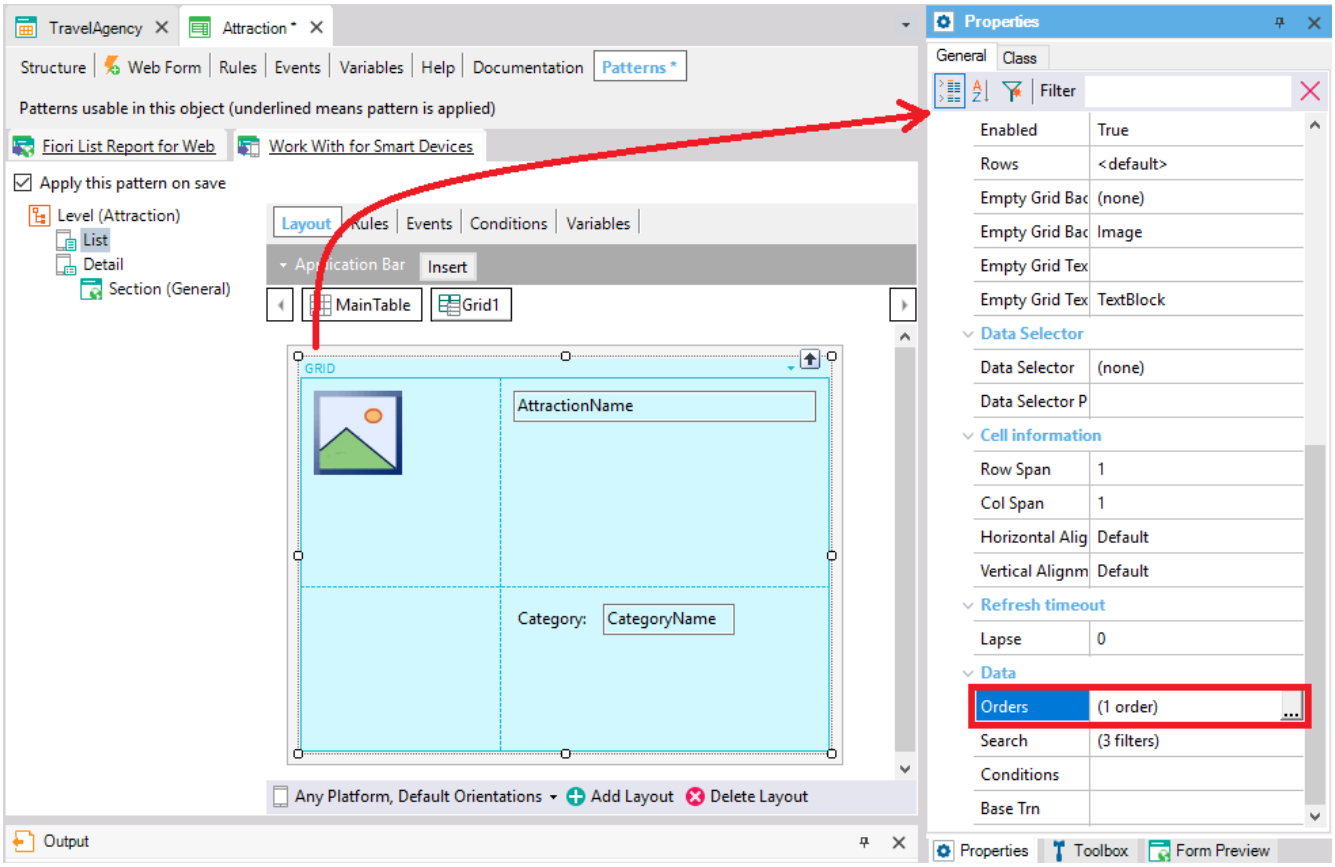


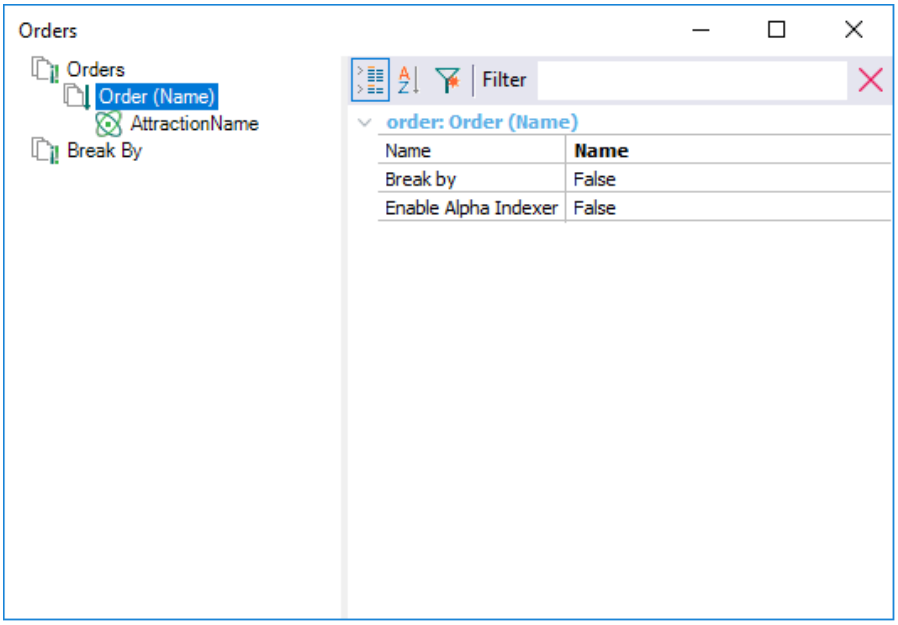


Y lo mismo ocurrirá con el filtro por categoría.

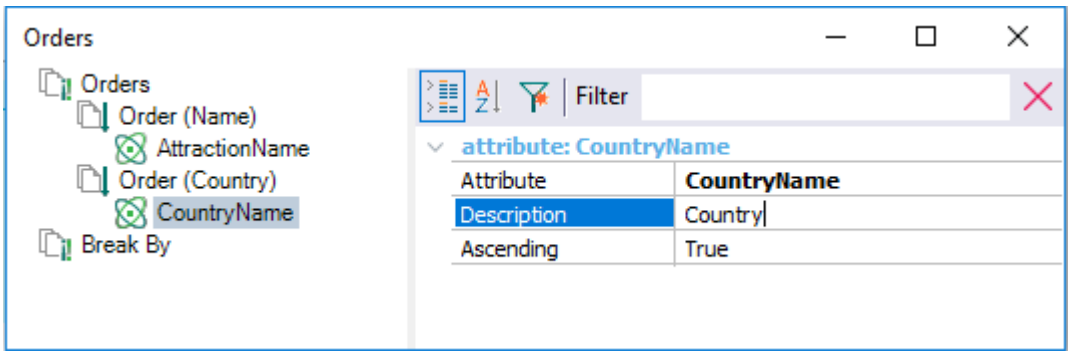
Podemos agregar filtros o eliminarlos.

Por otro lado, si observamos el listado, está saliendo ordenado por AttractionName. Eso está especificado entre las propiedades del grupo Data del grid:

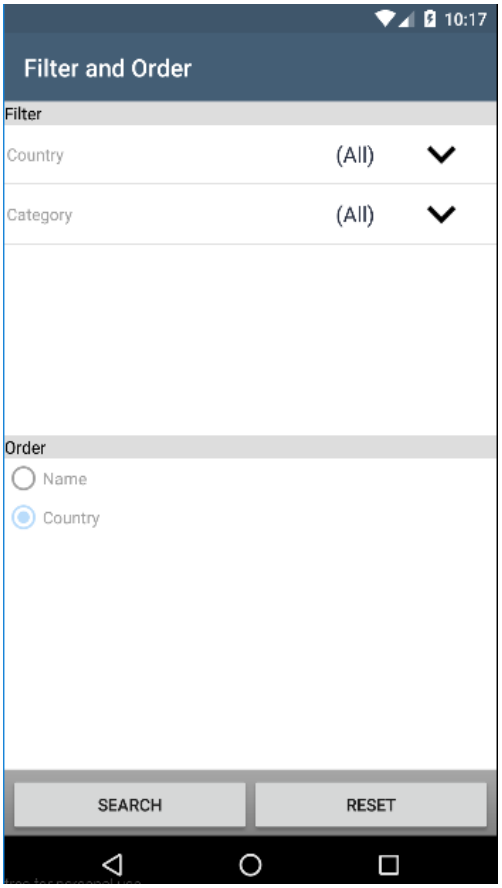




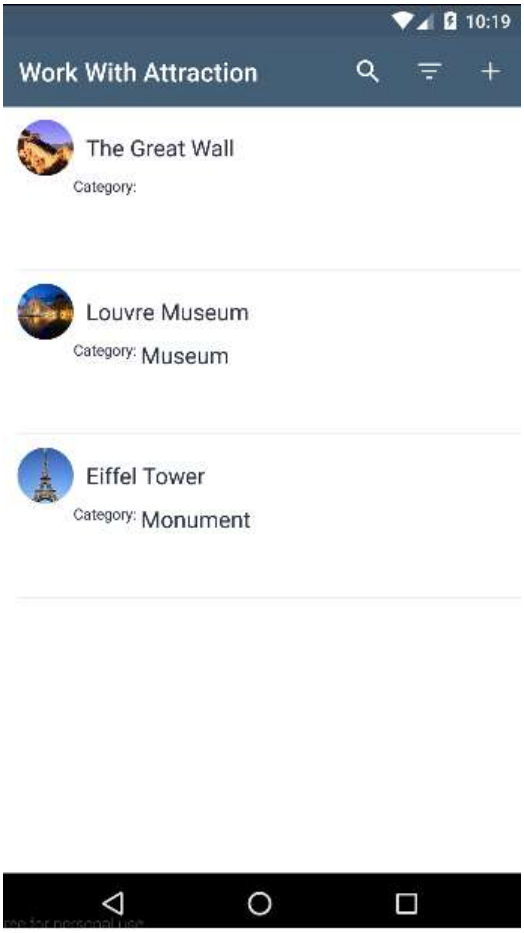
Podemos querer darle al usuario la opción de ordenar por país:



Y eligiendo la misma opción que para filtrar:

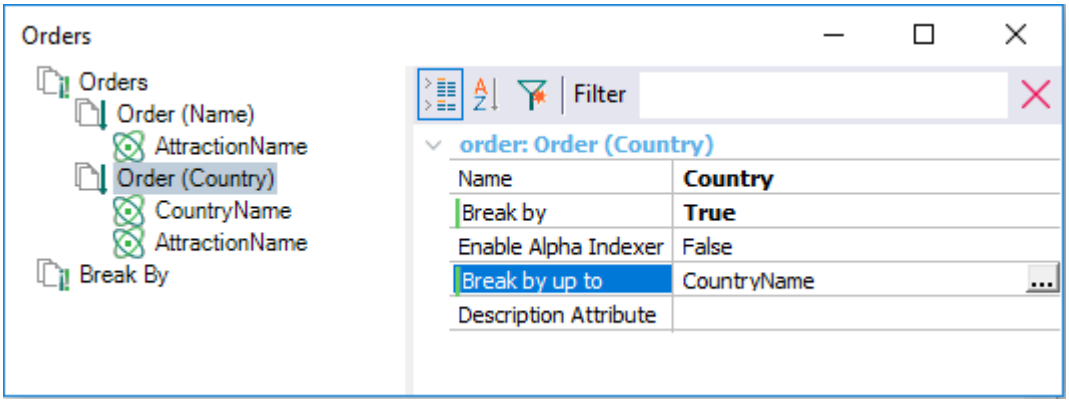


Pedimos ordenar por Country:

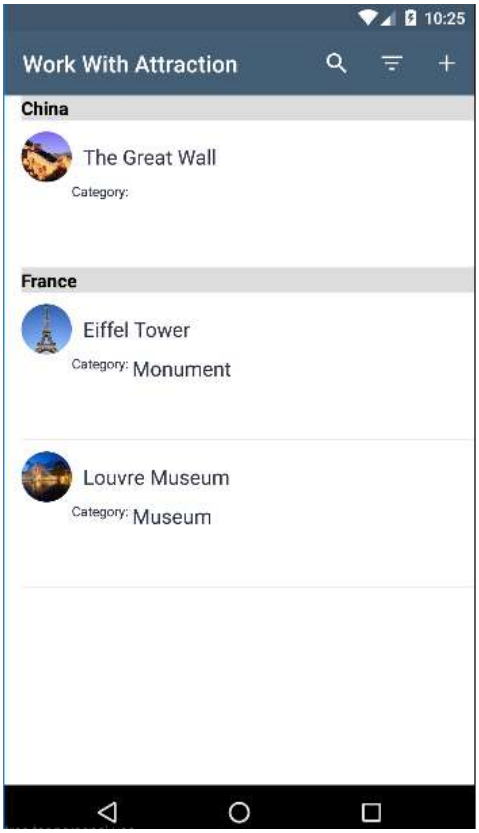


Vemos la gran muralla de China, y luego las dos atracciones de Francia.

Podríamos querer que esta opción muestre la información agrupada por país y para cada país ordenada por nombre de atracción. Para ello, modificamos este orden… agregando el atributo AttractionName… y especificando que se cortará la información… por CountryName:

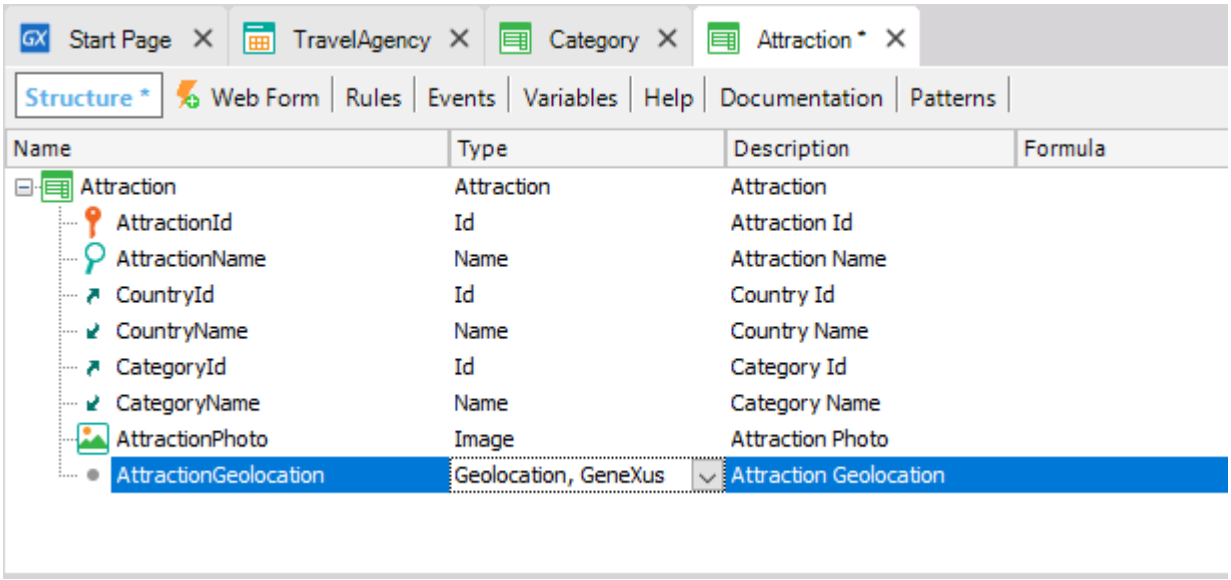


Veámoslo en ejecución:



Veamos alguna otra personalización interesante. Por ejemplo, querremos mostrar el listado de atracciones no como este listado estándar, sino como ubicaciones geográficas en un mapa.

Para ello debemos agregar un atributo a la transacción que registre la geolocalización de cada atracción:

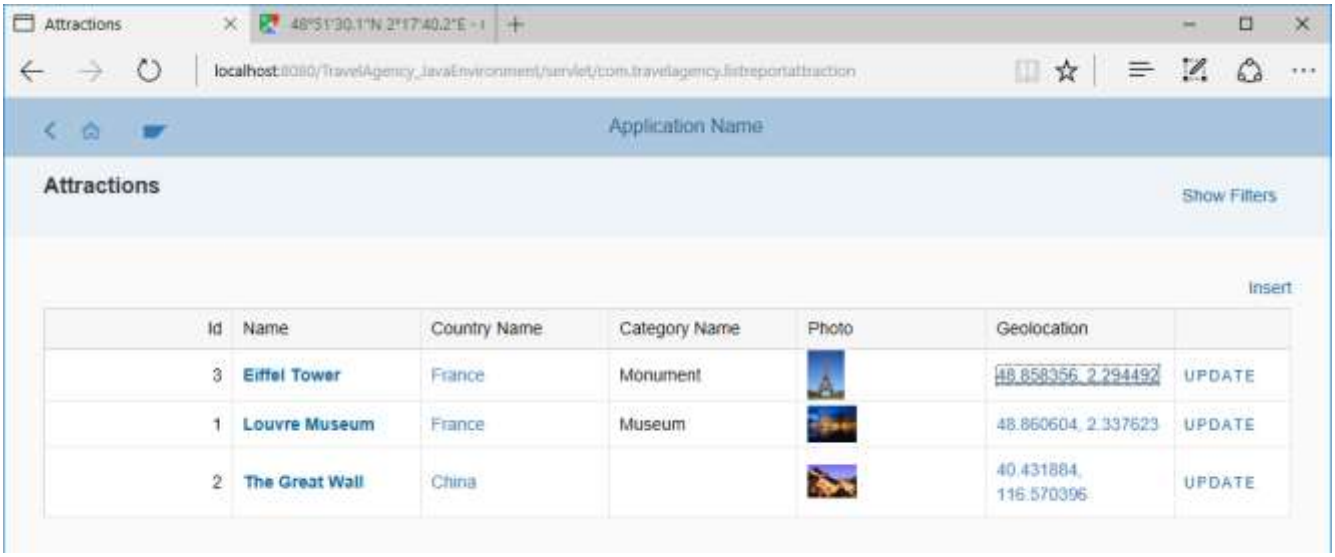


Vemos que asume automáticamente el dominio Geolocation, que es semántico, en el sentido de que la aplicación sabrá que se trata de un par longitud, latitud.

Presionamos F5… GeneXus nos pedirá reorganizar la base de datos, para agregar el atributo nuevo a la transacción Attraction.

Agreguemos mediante la app web las geolocalizaciones de las tres atracciones.

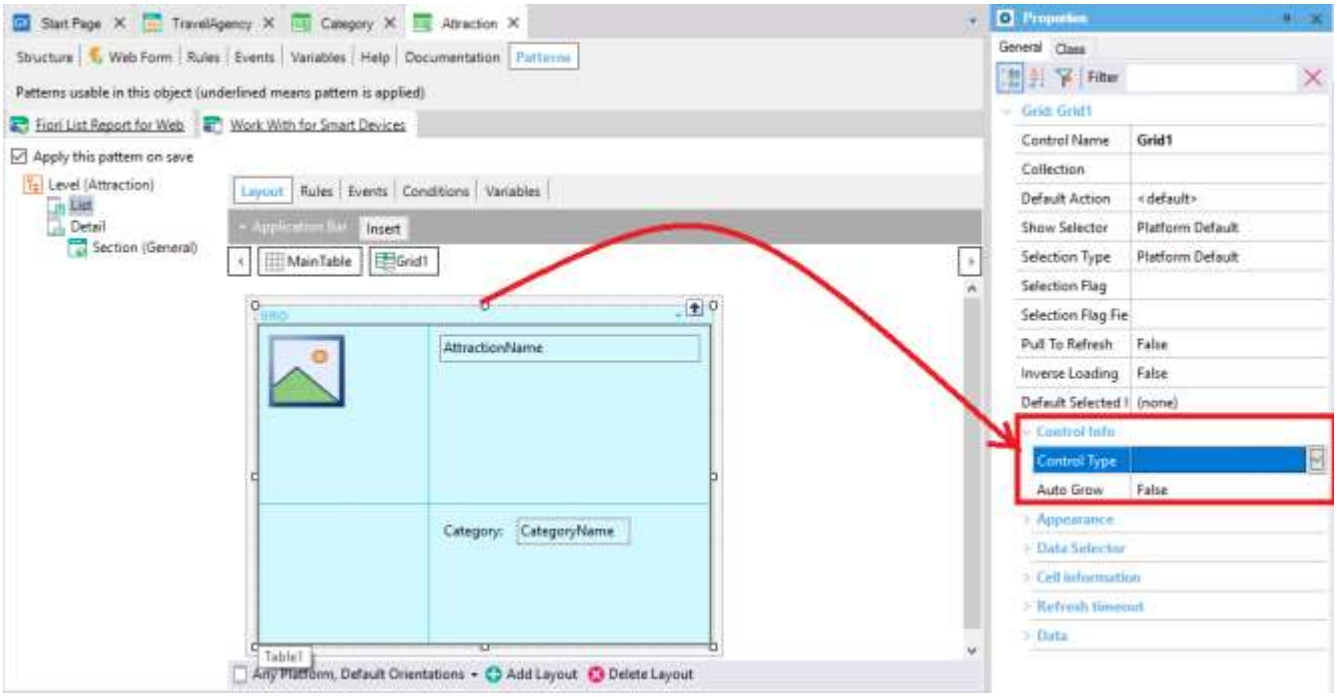
Observemos que si hacemos clic sobre el campo, ya nos abre Google maps mostrando con un pin esa geolocalización.



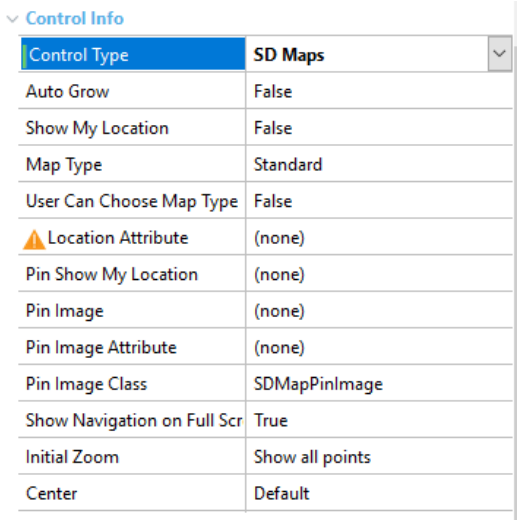
Aquí vemos en la práctica lo que significa que un dominio es semántico.

Queríamos que el Work With de atracciones en el dispositivo sea un listado que muestre sus elementos como puntos en el mapa.

Para ello, vamos al List del pattern…
Editamos las propiedades del Grid, y bajo el grupo Control Info:

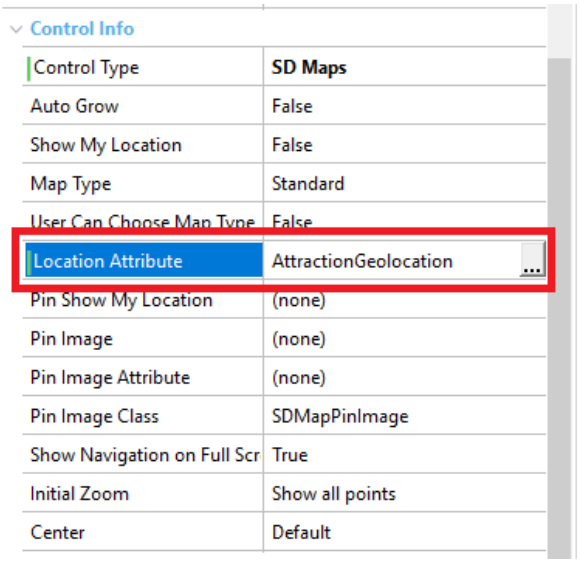


Modificamos el valor por defecto de la propiedad Control Type, seleccionando “SD Maps”:



Tras lo cual se habilitan esta serie de propiedades.

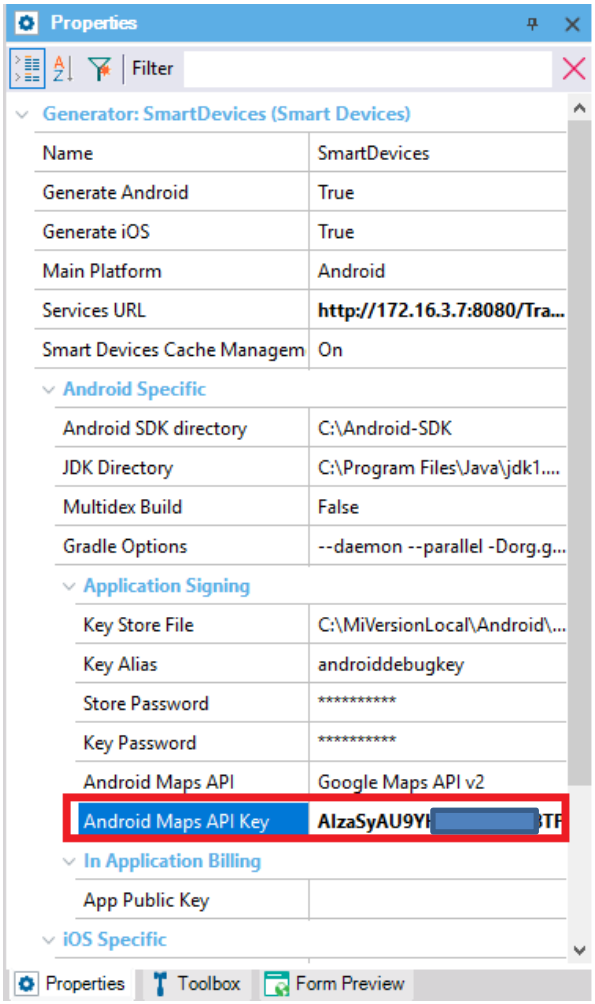
Debemos indicar cuál es el atributo de la transacción que contiene la localización. Lo especificamos:



Antes de ejecutar: para que una aplicación Android permita utilizar la API de mapas, debemos pedir una clave de Google que se corresponda con la firma de nuestra app. La Key se obtiene de forma más o menos sencilla en Google, registrando la app que estamos desarrollando.

Para ver cómo hacerlo, obteniendo la API key, vaya a nuestro wiki. HowTo: Get an API Key from Google (http://wiki.genexus.com/commwiki/servlet/wiki?19055,HowTo%3A+Get+an+API+Key+from+Google).
Con sencillos pasos se le explicará cómo hacer.

En las propiedades del generador Smart Devices, bajo el grupo Android, especificamos esa clave:



Los mapas de google no vienen configurados en los emuladores, por lo que utilizaremos directamente el dispositivo contectado vía USB a la computadora:

Podemos probar la misma aplicación para iOS, modificando solamente estas propiedades.

Con esto vimos una pequeña muestra de lo que podemos hacer. Así como para una aplicación web, existen más objetos para cubrir otras funcionalidades, como los Panels for Smart Devices.