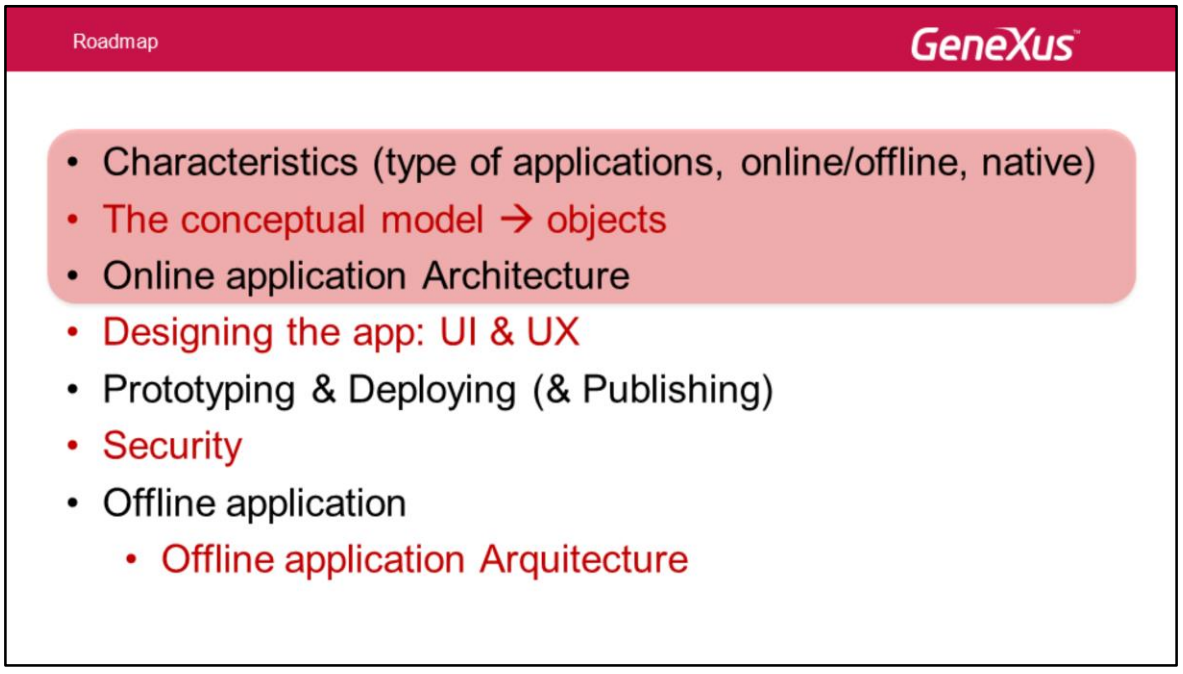
Mobile applications with GeneXus

Introduction

GeneXus X Evolution 3



En el curso “Aplicaciones móviles con GeneXus” se trabaja sobre:

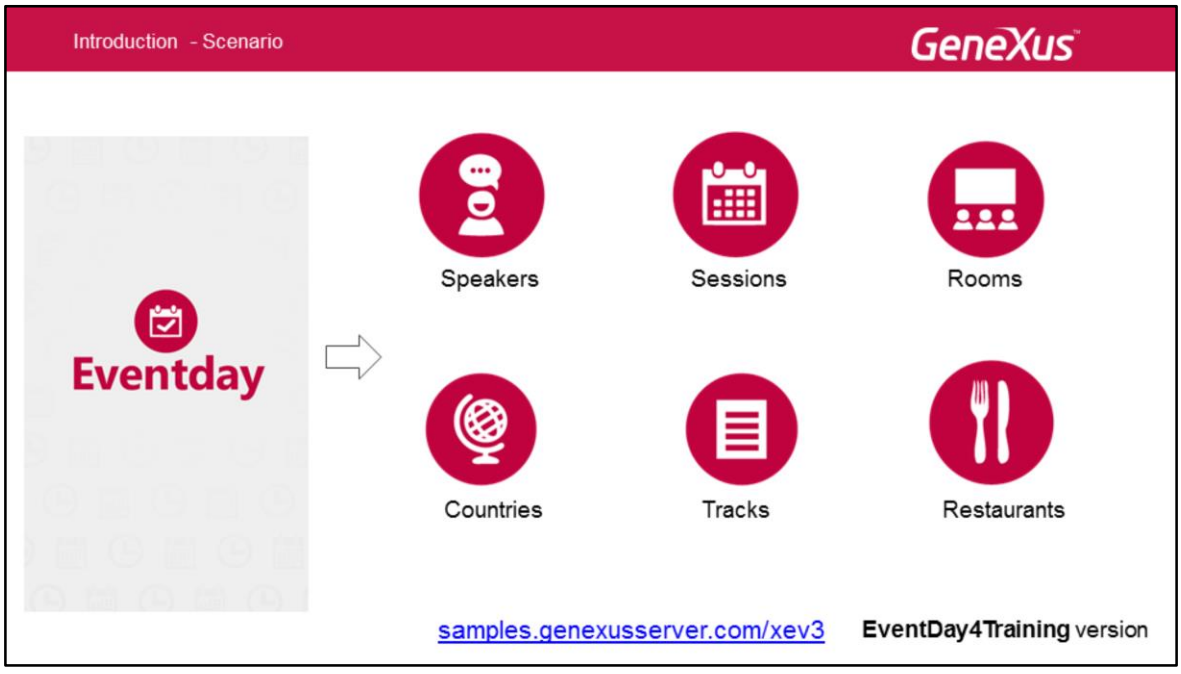
- Las **características** de estas aplicaciones (tipos de aplicaciones, la posibilidad de desarrollar aplicaciones que sólo funcionan conectadas, o que también funcionan completa o parcialmente desconectadas)
- el **modelo conceptual** subyacente, y los **objetos** en los que descansa.
- la **arquitectura** tanto de las aplicaciones totalmente **conectadas**, (que se verá al principio) como de las parcial o totalmente **desconectadas**, y cómo implementar cada una,
- los **objetos** y cómo se implementan, los aspectos de **diseño**, tan importantes en estas aplicaciones, tanto la **User Interface** como la **User Experience**
- las formas de **prototipar** y **poner en producción**, para las distintas plataformas y cómo se publica en los stores de cada plataforma,
- cómo agregar **seguridad** a la aplicación, de forma bien sencilla.
- las aplicaciones **desconectadas**.

En este material veremos solamente los primeros tres puntos, que conforman una buena introducción al desarrollo de aplicaciones móviles con GeneXus.
Si desea continuar con los puntos del índice que no se abordarán aquí, le sugerimos ver los videos asociados en http://training.genexus.com/gx-for-smart-devices-course-xev3?es.

GeneXus

Getting Started:

a tour through the mobile application at runtime

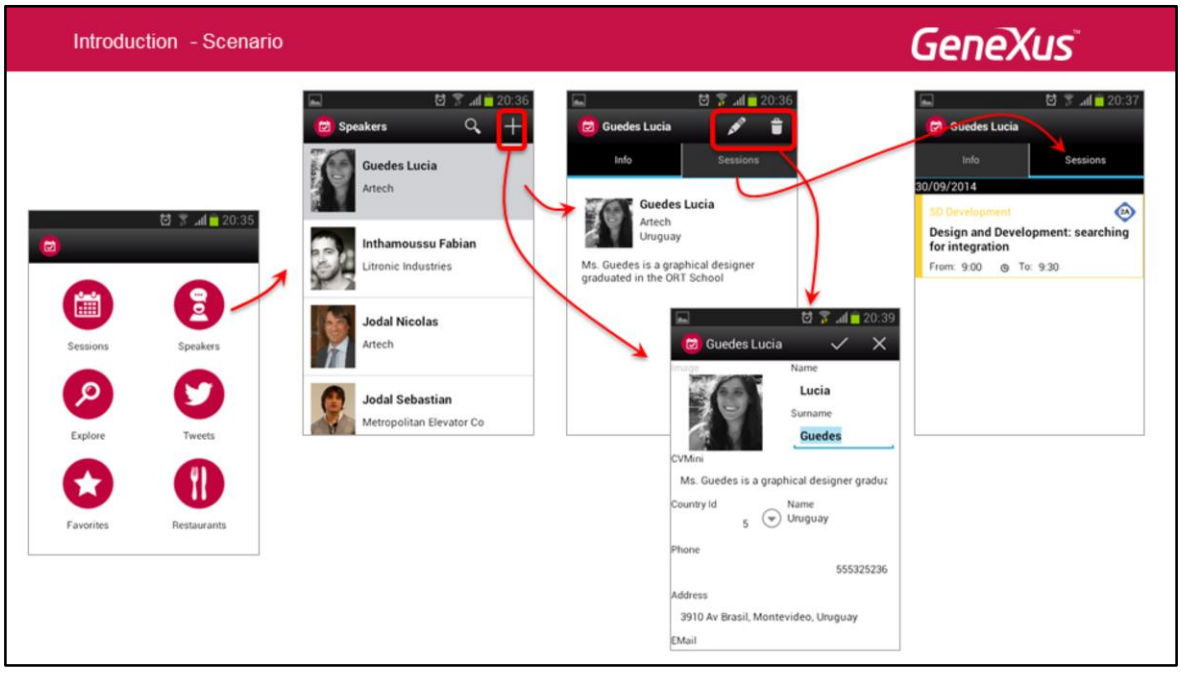


Empezaremos por tomar de uno de nuestros sitios de samples (samples.genexusserver.com/xev3) una aplicación ya desarrollada, EventDay (versión EventDay4Training), y la veremos **solamente** en ejecución, tanto en Android, como en iOS. Nuestro objetivo será lograr desarrollar desde cero una versión simplificada, en GeneXus, como excusa para poder abordar los temas más relevantes en lo que hace al desarrollo de aplicaciones para Smart Devices. Luego, usted podrá descargarse la versión completa para profundizar en los aspectos que no abordaremos en este curso.

La aplicación permitirá visualizar los oradores, y la agenda de conferencias, entre otras cosas, de un Evento (por ejemplo, de software) a realizarse en ciertos días, en cierto lugar.

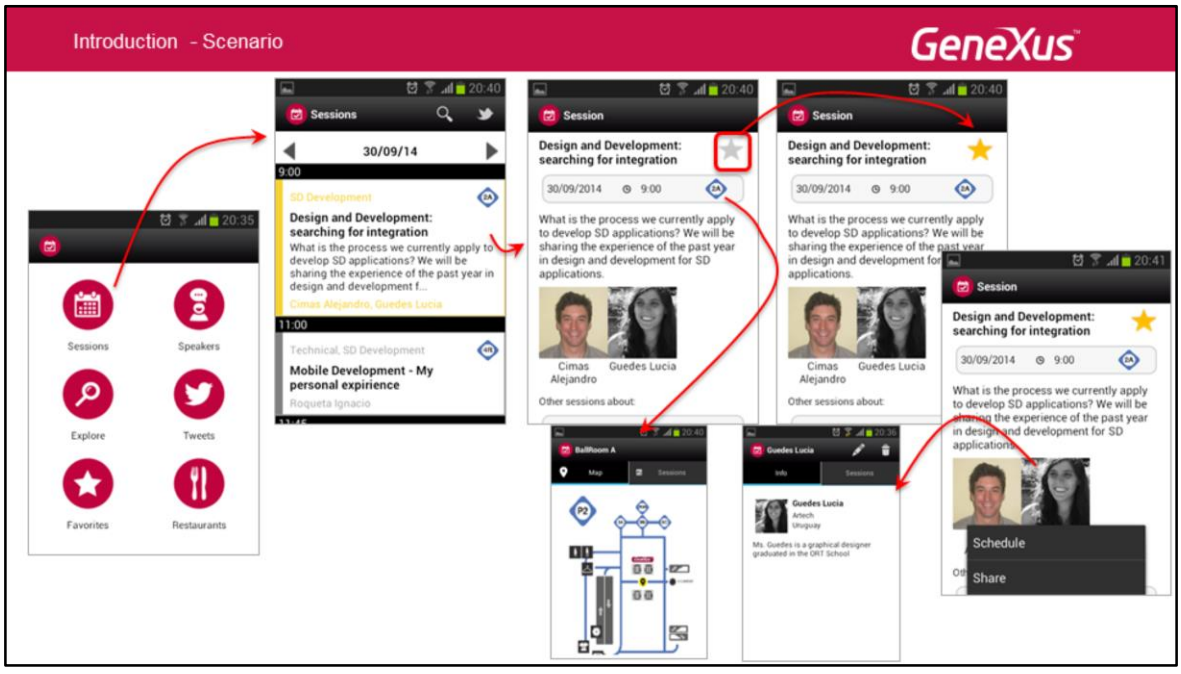
Permitirá visualizar los oradores (speakers), y sus nacionalidades (countries), las conferencias (sessions) con toda su información, las salas en las que se realizarán (rooms), los tipos de conferencias (tracks), así como los restaurantes cercanos al lugar del Evento para poder salir a comer.

Queremos que esa aplicación pueda ser descargada por los asistentes al evento en sus dispositivos inteligentes, y también que una parte pueda ser utilizada por los organizadores para poder modificar la información de manera móvil, además de a través del backend web.



Veamos parte de la aplicación ya desarrollada, en ejecución.

Vemos que tenemos un menú principal, que nos permite elegir entre las diferentes opciones.
Por ejemplo, si hacemos “tap” (el gesto con el dedo que en web corresponde a hacer click) sobre la opción Speakers, nos abre la pantalla que vemos, que nos muestra una **lista** de todos los oradores. Si queremos **ver** la información completa de uno de los oradores, lo elegimos haciendo tap. Ahí vemos tanto su información general, como las conferencias en las que participa.
Además, podemos **modificar** la información general del orador, **eliminarlo**, o incluso **insertar** un nuevo orador (más adelante esta parte sólo estará disponible para usuarios con roles especiales).

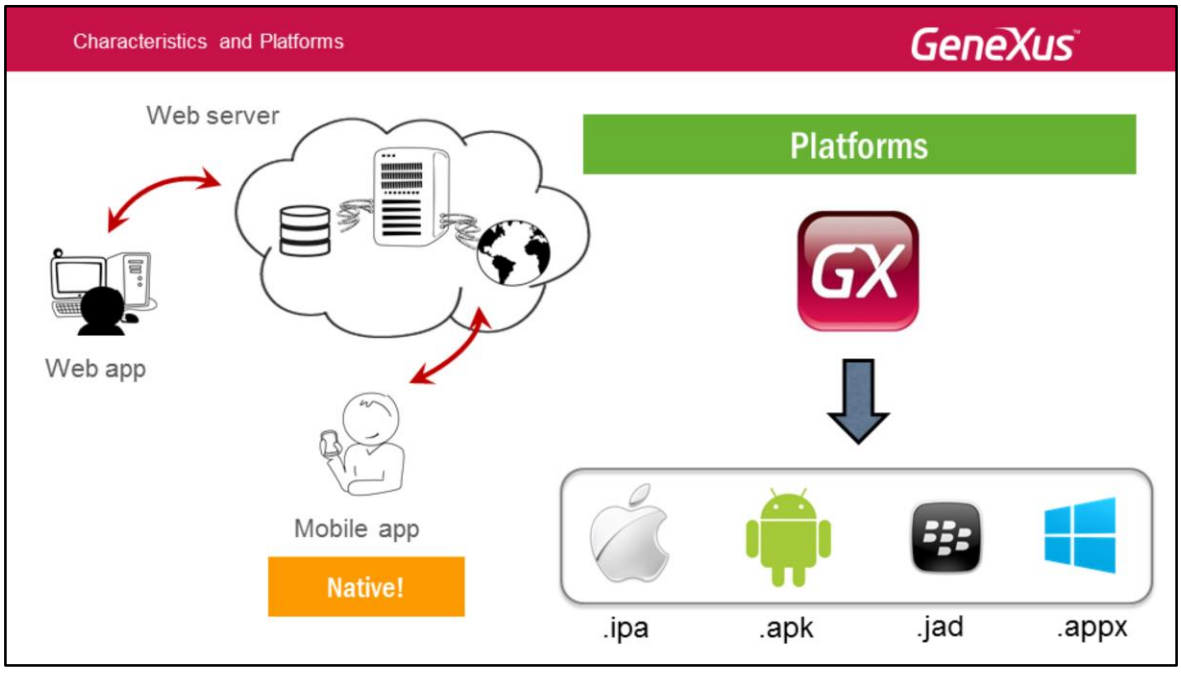


Si elegimos las Sessions, vemos una lista con un abstract de las conferencias de cada día, agrupadas por horario. Si hacemos tap sobre una, se nos despliega la información completa de esa conferencia, incluyendo horario y sala (haciendo tap sobre ésta, podemos ver los datos de esa sala, como el mapa que muestra dónde se encuentra), los speakers (haciendo tap sobre uno, nos lleva nuevamente a la pantalla que muestra la información de ese orador), los tipos de charla. Esta pantalla nos permite, además, ejecutar otras acciones, como marcar la charla como favorita para mí, agendarla en el calendario de mi dispositivo, o compartirla (con los programas que tenga instalados, como Facebook).

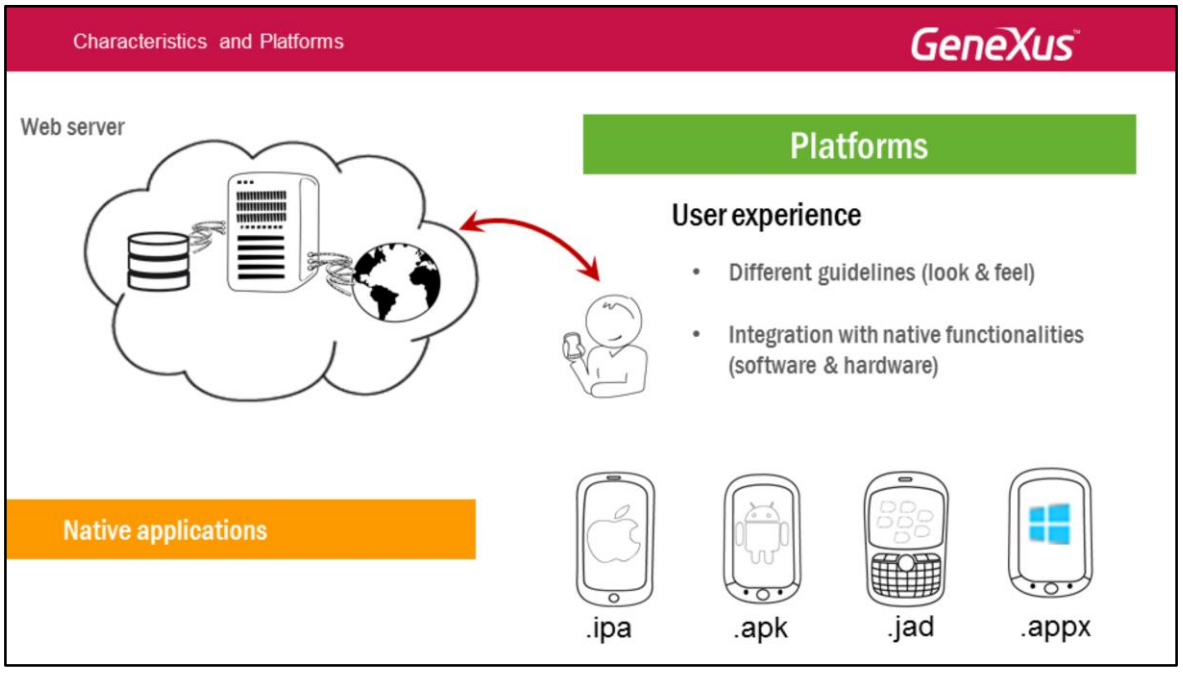
Abordaremos las principales características de las aplicaciones para Smart Devices: el que puedan ser en la línea del negocio o independientes de ella, el que puedan funcionan solamente con conexión a internet, o parcial o totalmente desconectadas, la necesidad de que sean nativas.

Luego veremos la interfaz y el modelo conceptual, para pasar por último a presentar la forma de integración con recursos del dispositivo.

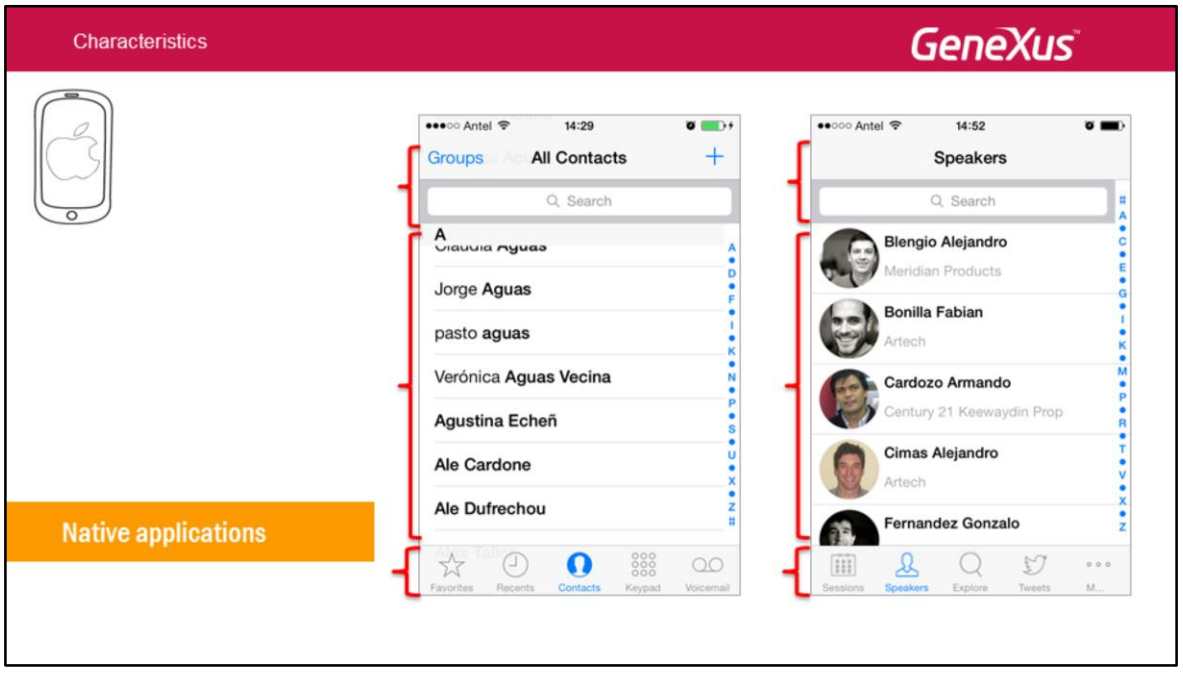
Terminados estos puntos, trabajaremos sobre la arquitectura de las aplicaciones que requieren conexión para funcionar (las online).



GeneXus desarrolla para las cuatro plataformas de dispositivos inteligentes que hay actualmente en el mercado: Apple, Android, Blackberry y Windows 8 y phone.
El producto final del desarrollo va a ser un binario compilado a partir del lenguaje de la plataforma. Aquí vemos las extensiones de acuerdo a cada una de ellas.
Ya podemos introducir una primera característica: las aplicaciones para Smart Devices van a ser nativas.

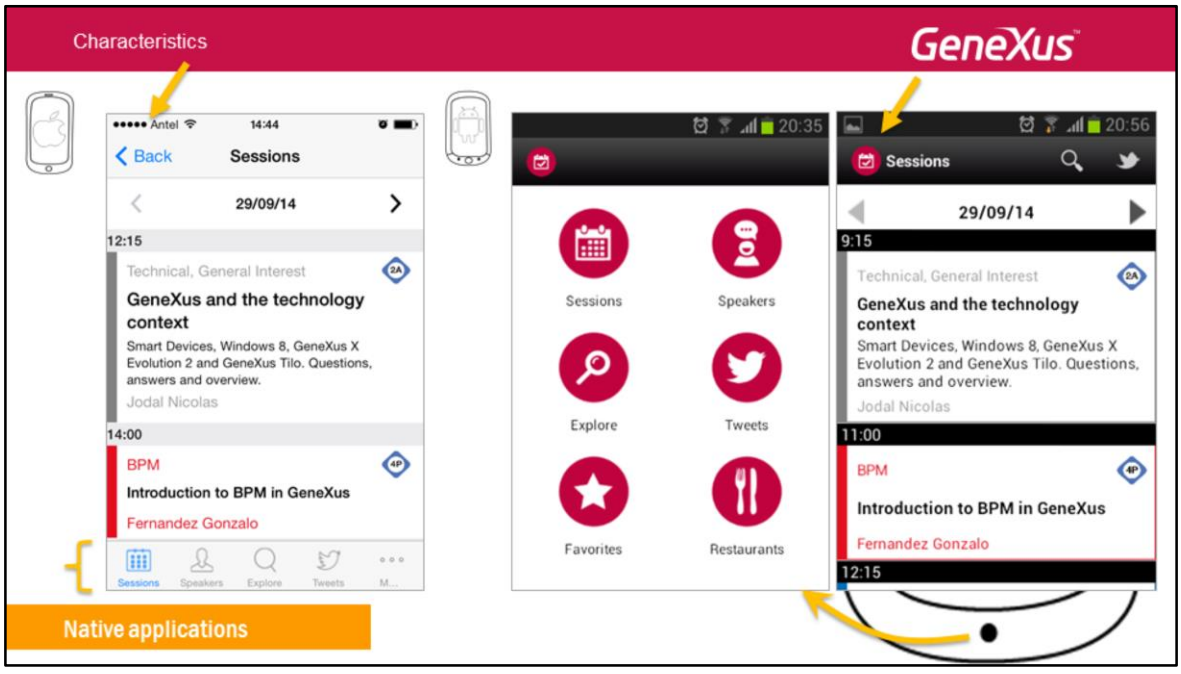


El usuario va a exigir que toda aplicación para su dispositivo tenga un look & feel similar al resto de sus aplicaciones nativas (como su aplicación de contactos, su calendario, la forma de hacer back, etc.), es decir, que cumpla las guidelines de su plataforma. Además, que se integre con las demás funcionalidades del dispositivo: de software, como el calendario, o la agenda de contactos; de hardware, como la cámara de fotos, la llamada telefónica o el gps.



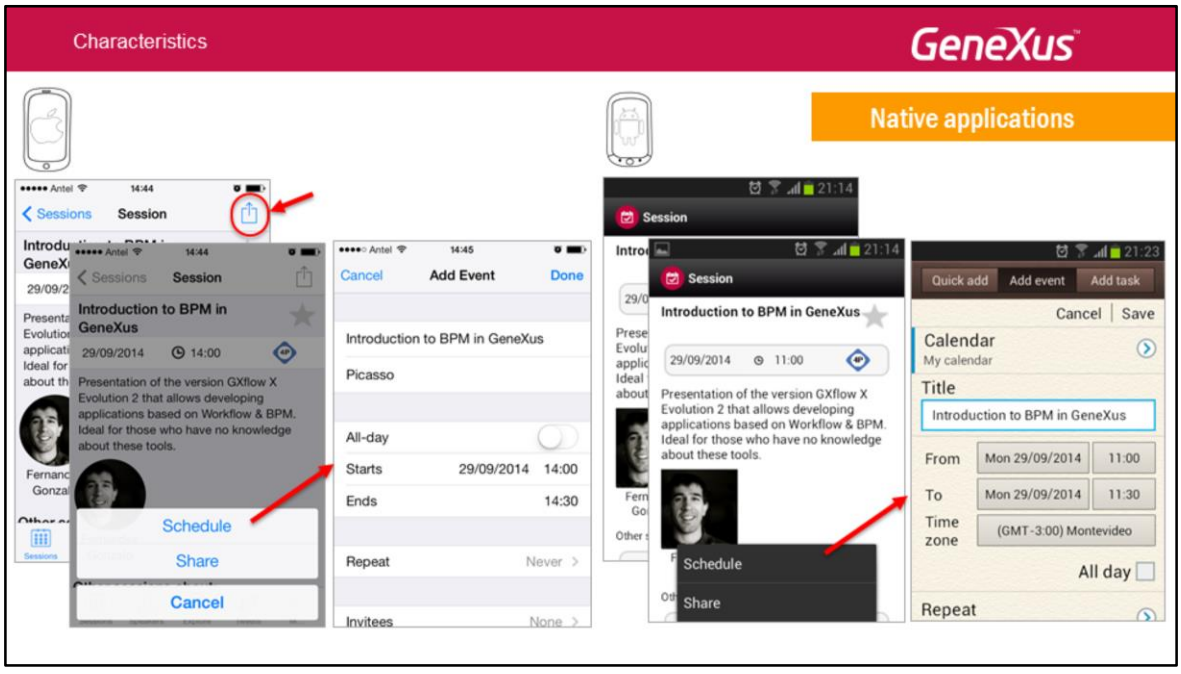
Aquí vemos, por ejemplo, cómo para un iphone 7, se ve la aplicación nativa de los contactos (izquierda) y cómo se ve la aplicación desarrollada con GeneXus, EventDay (derecha).

Independientemente de las definiciones de diseño, colores, etc, si observamos bien, tienen determinado patrón común de apariencia: una barra superior con algunas acciones, una opción de search, la información en el centro y luego un “menu” en forma de “tabs” para cambiar la opción a visualizar.



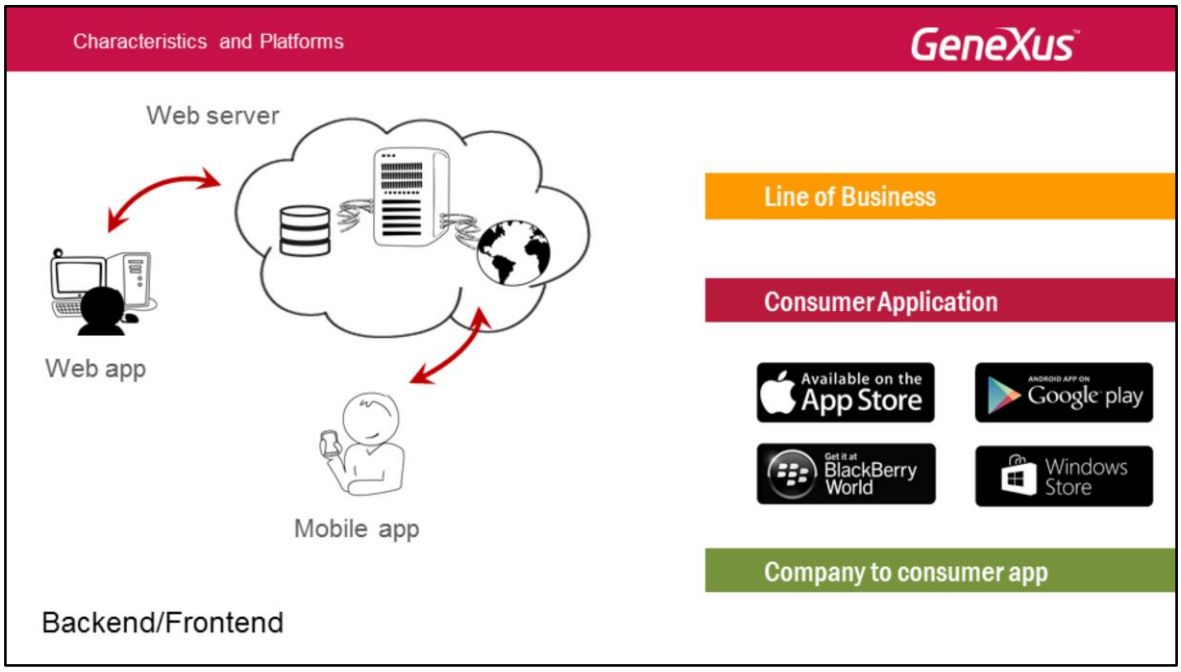
Sin embargo, cuando cambiamos de dispositivo, si bien la aplicación es esencialmente la misma (muestra una lista de conferencias en este caso), tiene sutiles pero importantes diferencias estéticas y funcionales.

Por ejemplo, en el caso de iphone tengo el menu permanentemente visible porque sus “guidelines” así lo indican, mientras que en Android phone no. En Android para volver al menú se hace tap sobre el ícono de la aplicación, o, si la pantalla que se está visualizando fue directamente invocada desde el menú, con el botón de back del propio dispositivo se vuelve. En cambio en iphone no hay botón de back, y el volver al llamador se incorpora en la propia aplicación (arriba a la izquierda).



Aquí podemos ver cómo la aplicación para iPhone, una vez posicionados sobre una conferencia (Session), nos permite a través de un ícono especial ver las acciones que podemos realizar sobre esa conferencia, como agendarla en el calendario del dispositivo (abre el programa nativo del dispositivo), o compartirla.

En cambio, la aplicación para un teléfono Android nos permite acceder a las acciones a través de un “botón” especial en el propio dispositivo.



Como veremos, toda aplicación para Smart Devices que desarrollemos tendrá necesariamente una parte ejecutándose en un servidor Web.

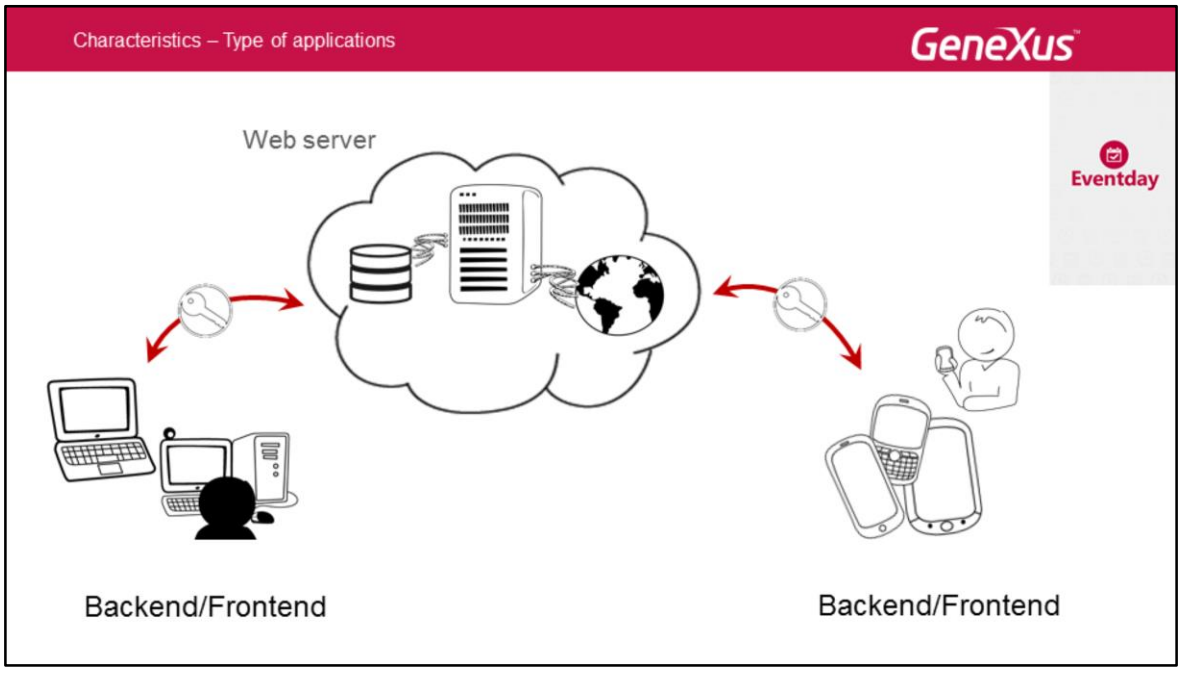
Podemos clasificar las aplicaciones de acuerdo a su integración o no a la línea de negocio de una compañía. Así, tendremos aplicaciones que son parte de aplicaciones más generales del negocio (**line of business**). Imaginemos, por ejemplo, que estamos en el escenario de un proveedor de productos que además de su aplicación web interna para manejar toda la operativa de la empresa, requiere una parte móvil para que los vendedores salgan a la calle, a las distintas empresas clientes, para levantar pedidos. El binario correspondiente a la aplicación móvil no necesita subirse a la tienda de cada plataforma, pues no será público. Alcanzará con que esté en los servidores de la empresa, para que pueda ser descargado por sus empleados en sus dispositivos e instalarlo. El caso de Apple es un poco más complejo, pues requerirá, para ello, de una cuenta enterprise que permitirá hacer un tipo de distribución ad hoc, donde no hay un proceso de aprobación de la tienda sino que se hace una validación para todos los que son empleados…

El otro tipo de aplicaciones son aquellas que están destinadas a subirse a las distintas tiendas de las plataformas, pues van dirigidas a las personas en general (**consumer application**). Podrán ser pagas o gratuitas, o tener partes pagas. Para subir los binarios a las tiendas se necesitará cuenta de desarrollador y certificados. Salvo Google Play, las demás tiendas, para publicar la aplicación, antes la validan. La más restrictiva es Apple, que valida la interfaz y la adecuación general de la aplicación a las guidelines, demorando su aprobación o rechazo entre 8 y 9 días. Google Play, que no valida, disponibiliza la aplicación en 2 hrs.

El tercer tipo de aplicaciones, un híbrido entre las primeras dos, es el de aquellas aplicaciones

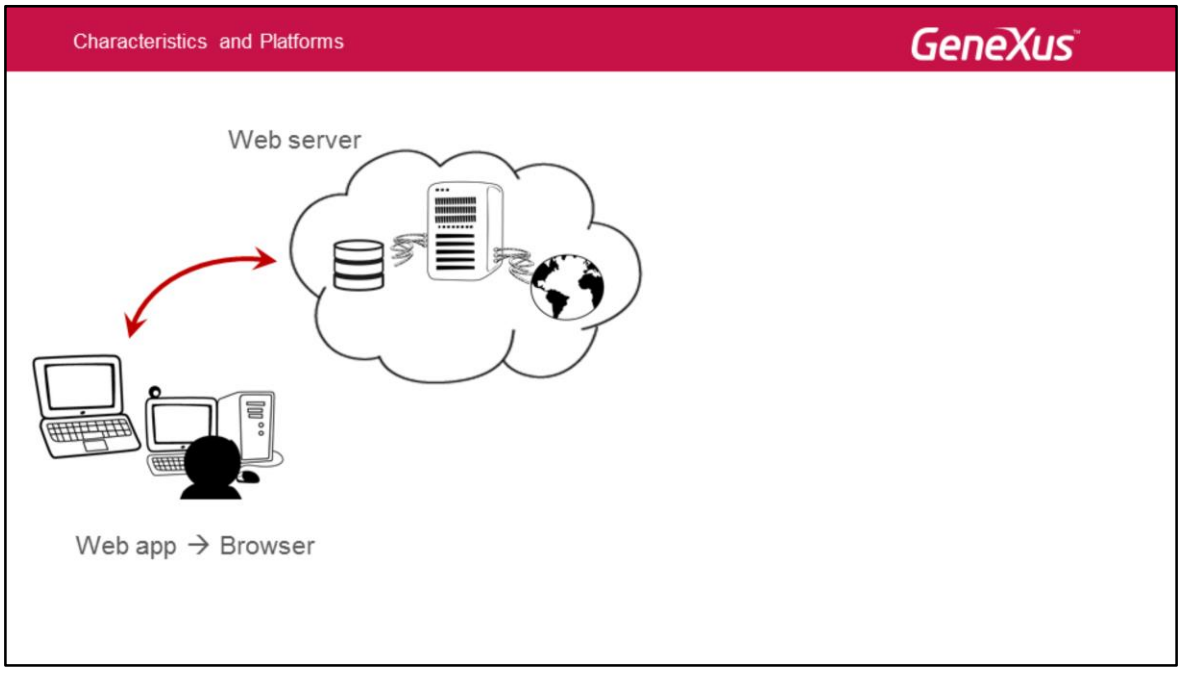
desarrolladas en el marco de una línea de negocio de una compañía, para extender su base de usuarios (**Company to consumer applications**). Por ejemplo, la aplicación que una cadena de supermercados disponibiliza para el público, de modo de que puedan hacer pedidos vía la aplicación.

Nuestra aplicación EventDay puede considerarse un híbrido, puesto que tendrá una parte cuyo foco estará puesto en los asistentes al evento en cuestión (frontend, Company to consumer), y otra parte que será utilizada por los organizadores del evento, para poder modificar datos de la base de datos de forma móvil (backend, Line of Business).



En definitiva, así como podemos querer publicar las conferencias, oradores, y demás a través de un sistema web (frontend), también queremos brindar esas funcionalidades y más, a través de una aplicación móvil. Y de igual manera, así como necesitamos un sistema backend web para que los usuarios autorizados puedan administrar la información del sistema, podemos necesitar también que todas o parte de esas manipulaciones de los datos, puedan realizarse por esos mismos usuarios autorizados, en la aplicación para el Smart Device. De este modo se podrán estar haciendo cambios en los datos desde el mismo lugar del evento, un minuto antes de que una conferencia dé inicio.

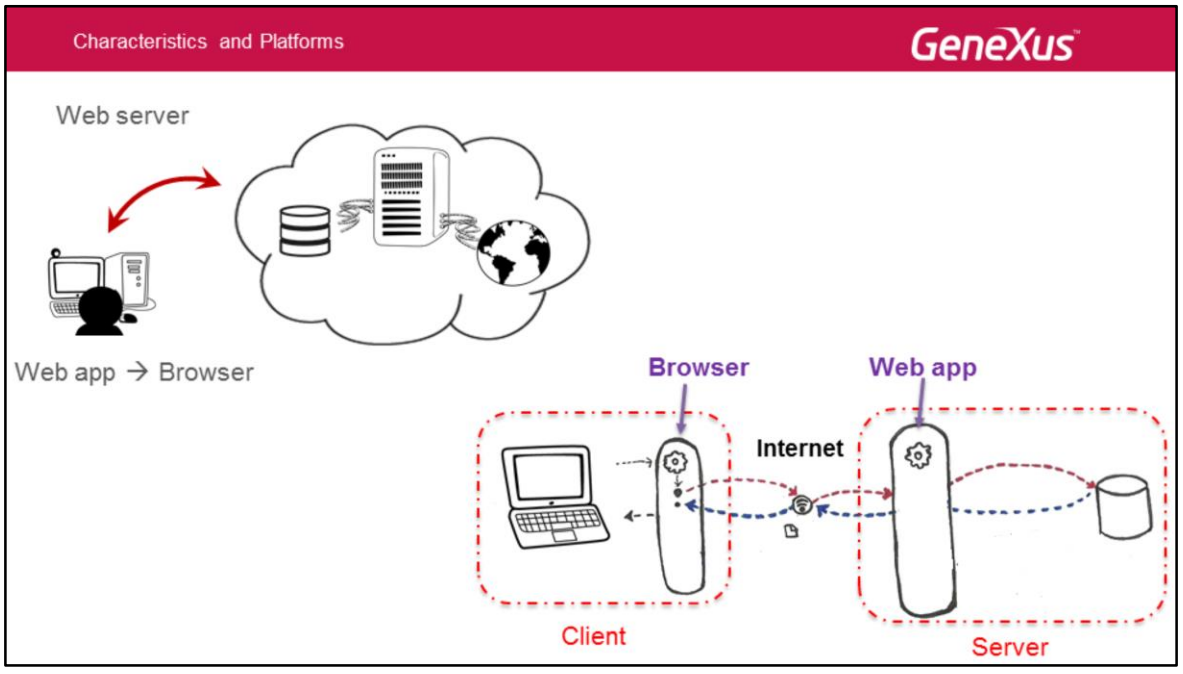
No hay casi diferencias funcionales entre la implementación del backend y la del frontend. La diferencia estará dada en que para usuarios autorizados (necesitaremos agregar seguridad, de modo que los usuarios se autentiquen y además, se autoricen) se permitirán las operaciones de CRUD (create/update/delete) sobre algunos datos, a través de pantallas que para usuarios no autorizados no estarán disponibles.



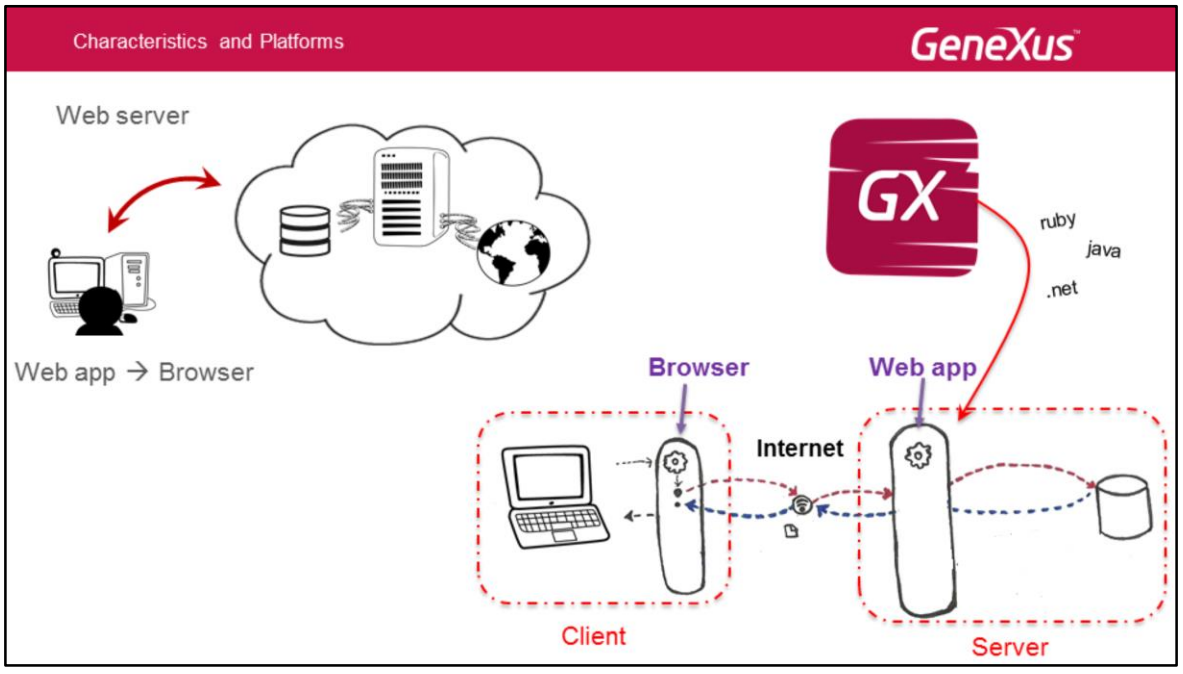
En cualquiera de los casos mencionados, la aplicación para dispositivos inteligentes tendrá una parte web, aunque ésta sólo se utilice para exponer servicios, como veremos..

Muchas veces la parte web funcionará como backend, pero también podrá tener un frontend (en nuestro ejemplo, el evento tendrá su sitio web).

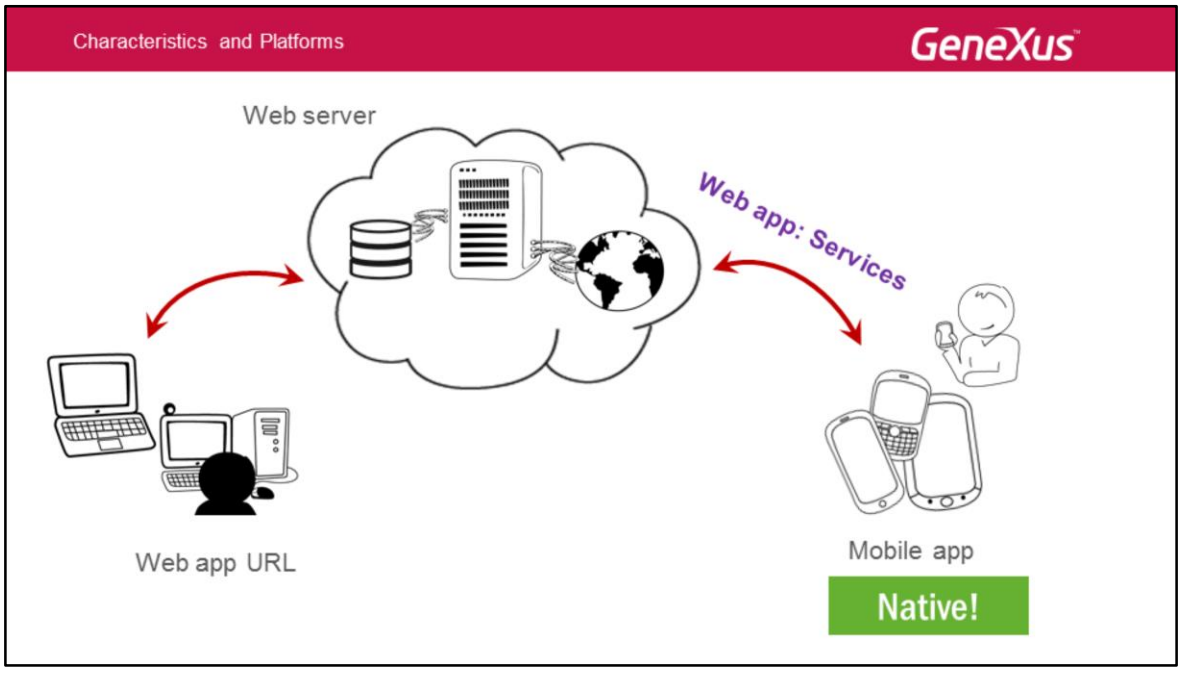
Todo lo que necesita el usuario para ejecutar esta parte web, es un browser…



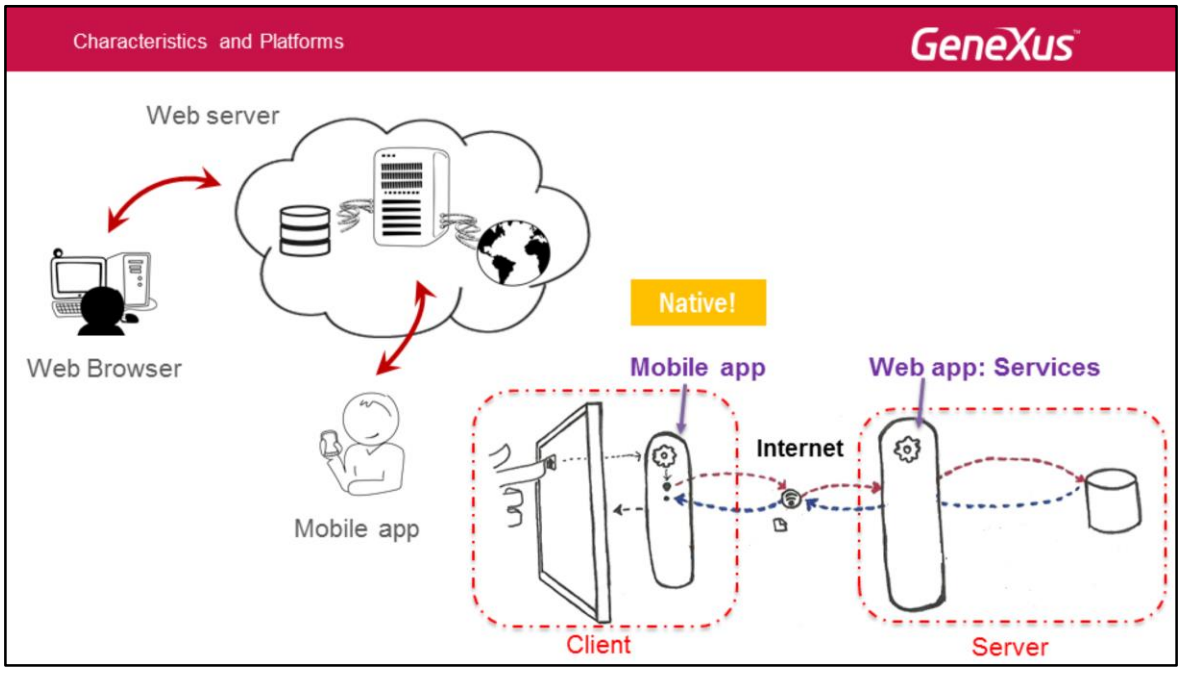
La aplicación web se encuentra en el servidor web, que es el que accede a la base de datos. El cliente únicamente necesita un browser.



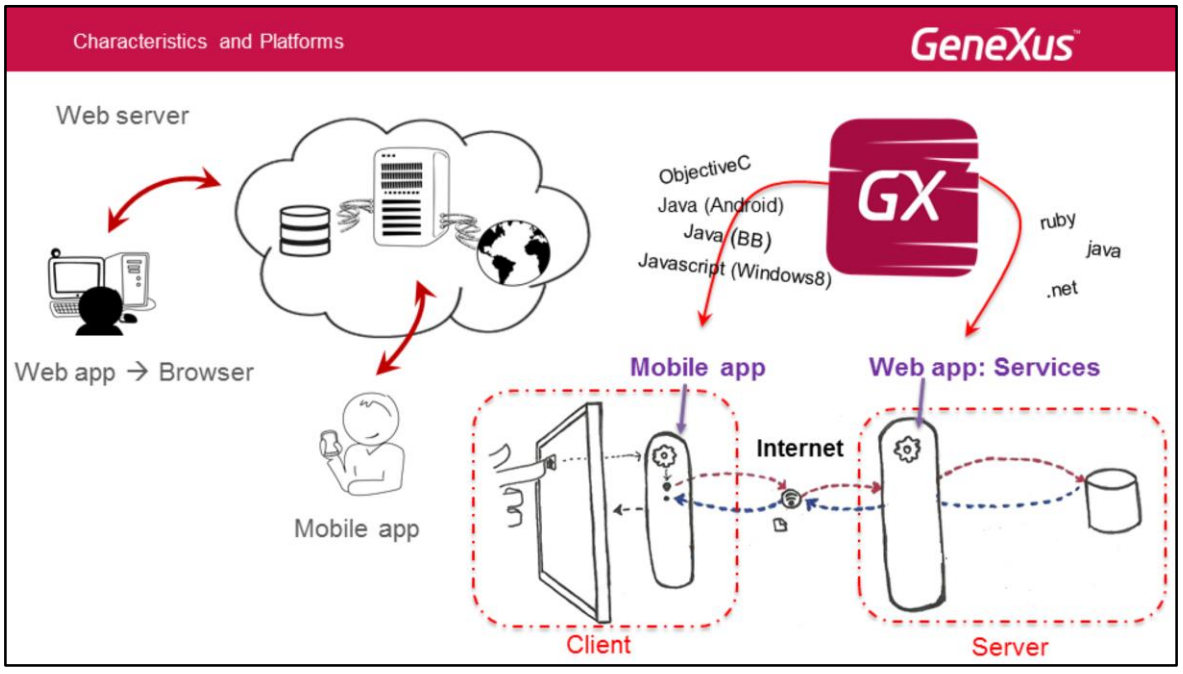
Lo que hacemos en GeneXus es crear las transacciones que definirán las tablas de la base de datos, y los web panels, y demás objetos GeneXus que accederán a esas tablas para recuperar los datos. Luego le pedimos a GeneXus que genere esos objetos en ruby, java, o .net, construyendo así la aplicación, que se pondrá en producción en el servidor web que corresponda.



Ahora queremos desarrollar la parte móvil de la aplicación. Para ello, los objetos que permiten hacer operaciones sobre la base de datos, así como recuperar su información, —esto es, los bussiness components, los data providers y ciertos procedimientos—, se exponen como servicios que serán consumidos por la aplicación nativa.

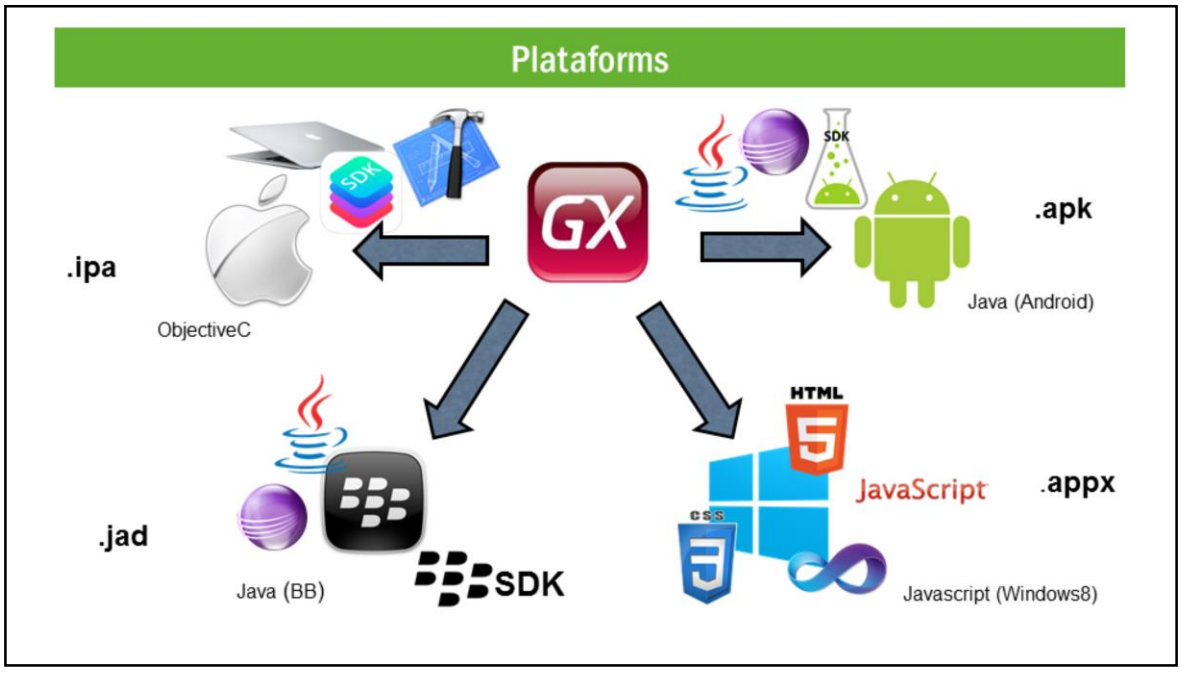


Ahora del lado del cliente estará el compilado correspondiente de acuerdo a la plataforma, que consumirá los servicios expuestos por la aplicación web en el server, para recuperar los datos y trabajar sobre ellos.

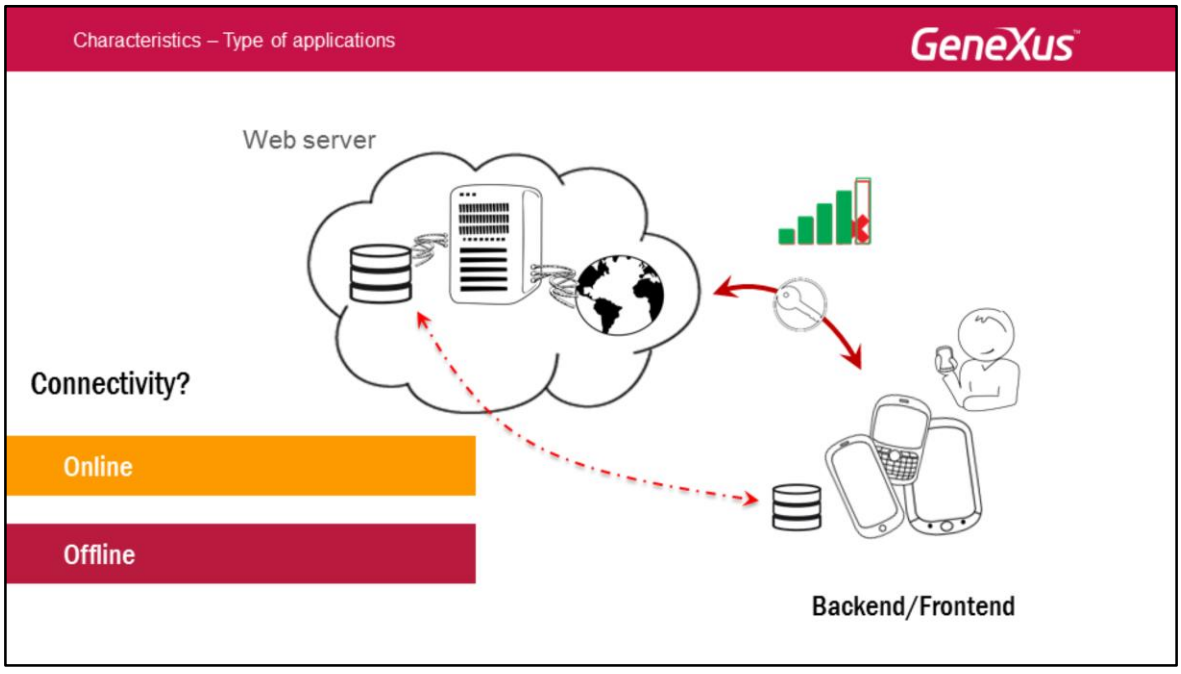


Así como la parte web se desarrollaba en GeneXus y éste luego la implementaba en alguno de los lenguajes disponibles (ruby, java o .net), lo que significaba que GeneXus era multiplataforma, la parte móvil se desarrollará análogamente, siguiendo la lógica de GeneXus, quien luego la implementará en el lenguaje de programación de la plataforma elegida, y siguiendo los estándares de interfaz y comportamiento de la misma. Por tanto, GeneXus sigue siendo multiplataforma, también en lo que hace a las aplicaciones para dispositivos inteligentes. El binario generado se instalará en el dispositivo y accederá, con conexión a internet, a los servicios del servidor web para recuperar y actualizar los datos.

Para simplificarnos la tarea de prototipación, GeneXus nos ofrece la posibilidad de desarrollar la parte web en una nube, de modo que siempre que tengamos acceso a internet podremos estar probando la aplicación, desde cualquier lugar, y todo el software (programas y base de datos) estarán hosteados allí sin que tengamos que preocuparnos por las infraestructura. Esto se hace a través de la propiedad **Deploy to cloud**, del generador web.



Si no tuviéramos GeneXus, para desarrollar aplicaciones en esos dispositivos deberíamos contar con múltiples herramientas y conocimientos.
En el caso de Apple se necesita el SDK que viene incluido dentro del xCode y trae un emulador para poder probar, y además una mac para poder compilar las aplicaciones, para poder subir el binario, que tiene la extensión ipa.
Luego en Android necesitamos el SDK que también trae el emulador para poder probarlo.
En Blackberry hay un SDK también y un emulador.
En Windows no hay un SDK específico. La programación es una mezcla de html 5, javascript y css.

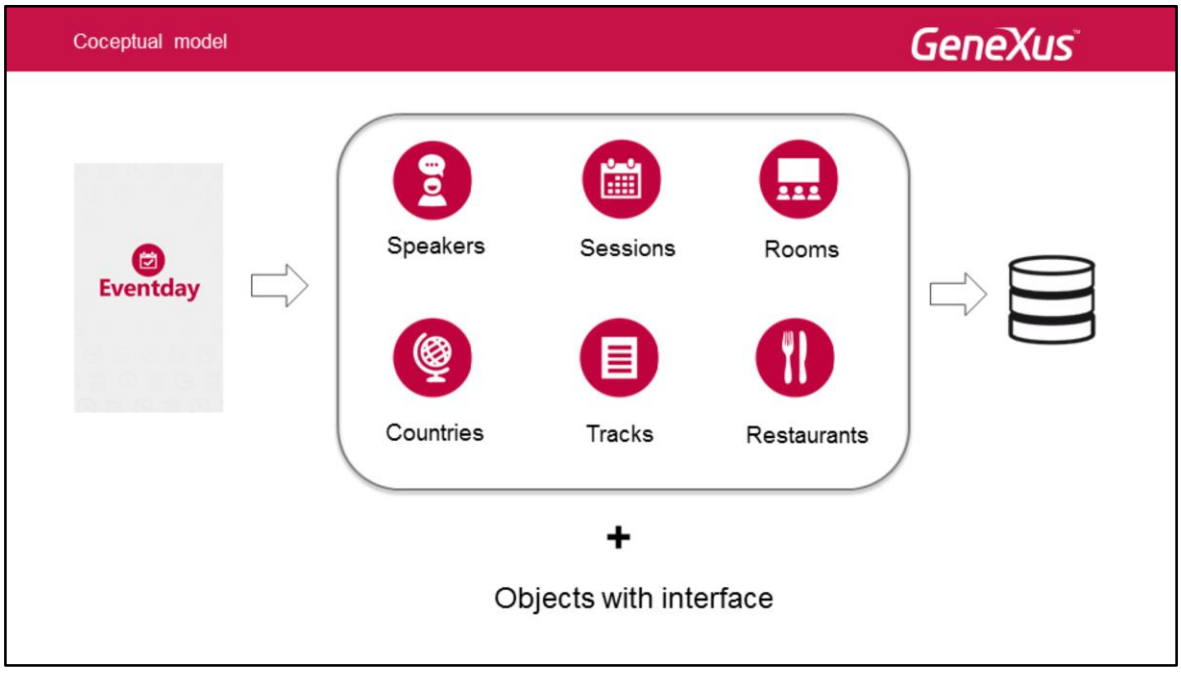


Un escenario importante para los dispositivos inteligentes es permitir que la aplicación, o parte de ella, se siga ejecutando cuando se encuentra desconectada de internet. En el caso de nuestra aplicación, vamos a querer que el usuario pueda seguir viendo toda la agenda de conferencias, y toda la información relacionada, incluso cuando pierde la conexión. Luego, cuando ésta se reestablece, automáticamente la aplicación actualizará sus datos (que estarán en una base de datos local en el dispositivo), sincronizándose con el server (receive). Como el usuario podrá haber marcado algunas conferencias como favoritas, también enviará esa información al server al sincronizarse (send), actualizando la base de datos central.

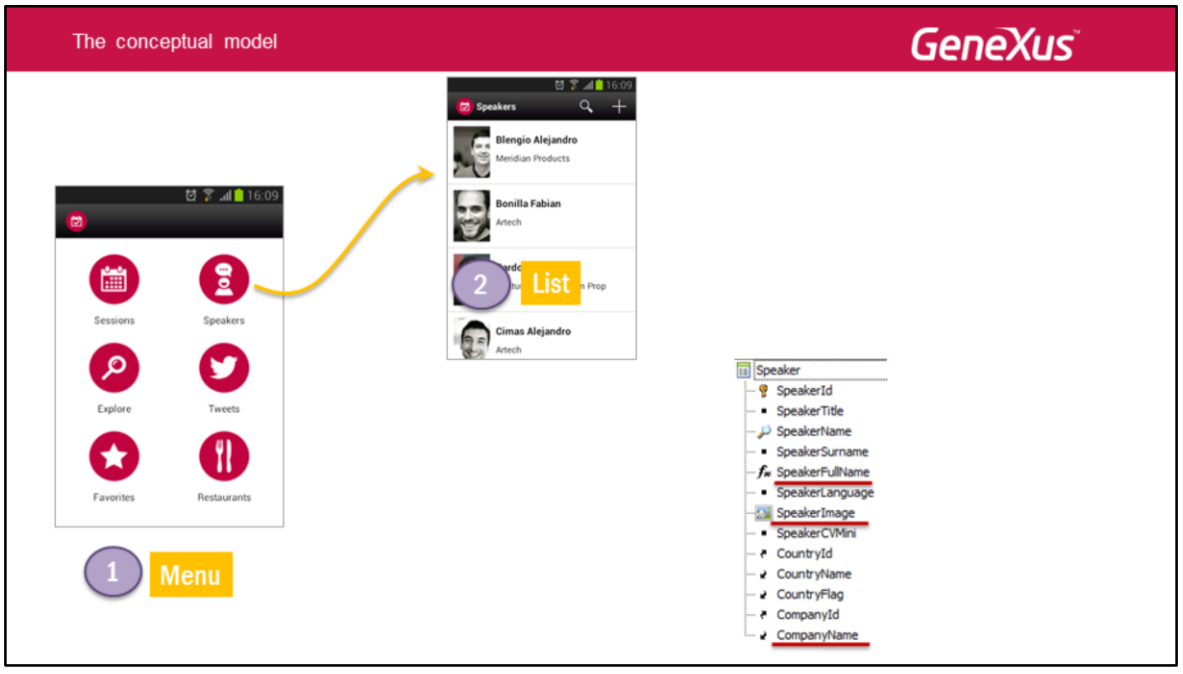
Sin embargo, habrá tareas que requerirán necesariamente el acceso al servidor web, ya sea por su sensibilidad (tareas que deben ser validadas en la base de datos centralizada, como en un escenario bancario), como por lo rápido que cambian los datos. Estas tareas deberán ejecutarse online.
En nuestro caso, el login deberá ser con conexión porque por seguridad no es conveniente replicar la tabla de usuarios en los dispositivos y el panel que muestra los tweets es deseable que también lo sea, por la velocidad en que se actualiza esa información.

Por tanto, podremos elegir qué objetos de la aplicación pueden ejecutarse offline y cuáles no.

Ahora nos introduciremos en el modelo conceptual correspondiente a estas aplicaciones. Veremos los cuatro tipos de pantallas que se necesitan para desarrollar una aplicación para Smart Devices, y los objetos que las implementan, y sus relaciones.



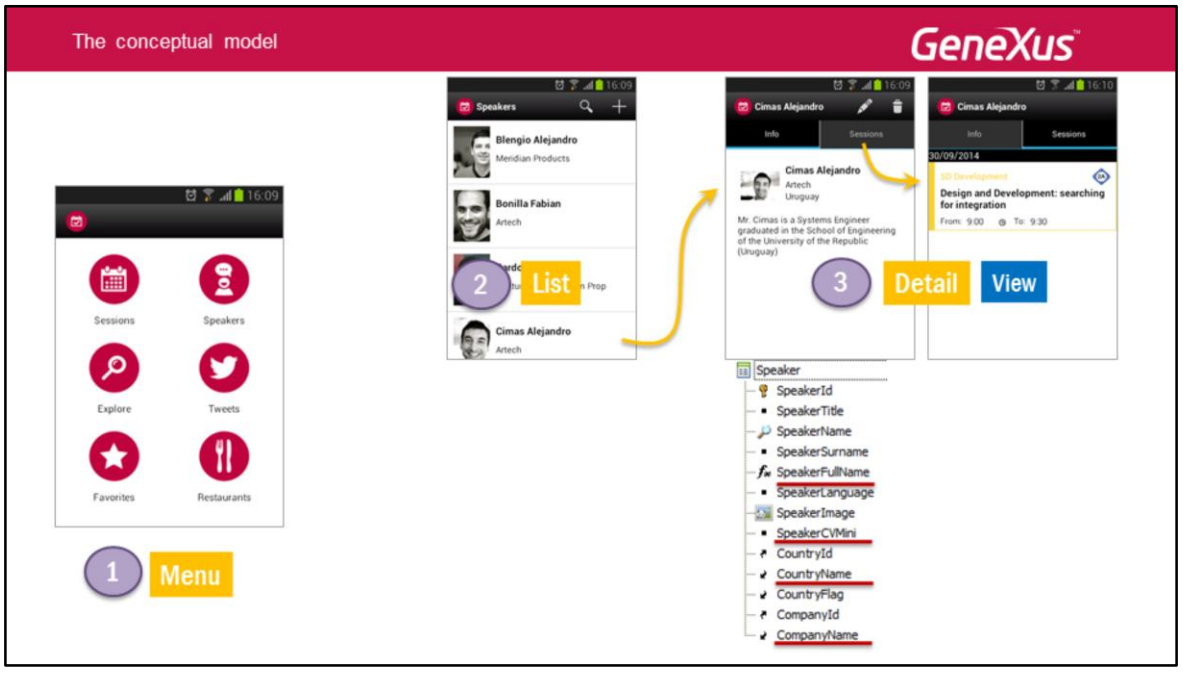
Para implementar nuestra aplicación EventDay partiremos de las transacciones que definen la estructura de datos y su lógica de trabajo a través de business components, y luego necesitaremos implementar ciertas pantallas, a través de objetos específicos.



Veremos los cuatro tipos de pantallas que se necesitan para desarrollar una aplicación para Smart Devices.

1. **Menú**: esencialmente contiene ítems que corresponden a las opciones que se le brindan al usuario. Puede mostrarse tanto como vemos en la imagen, como una tabla (es el default de Android), como tabs (default de iphone) o como una lista (default de iPad). Lo veremos luego.

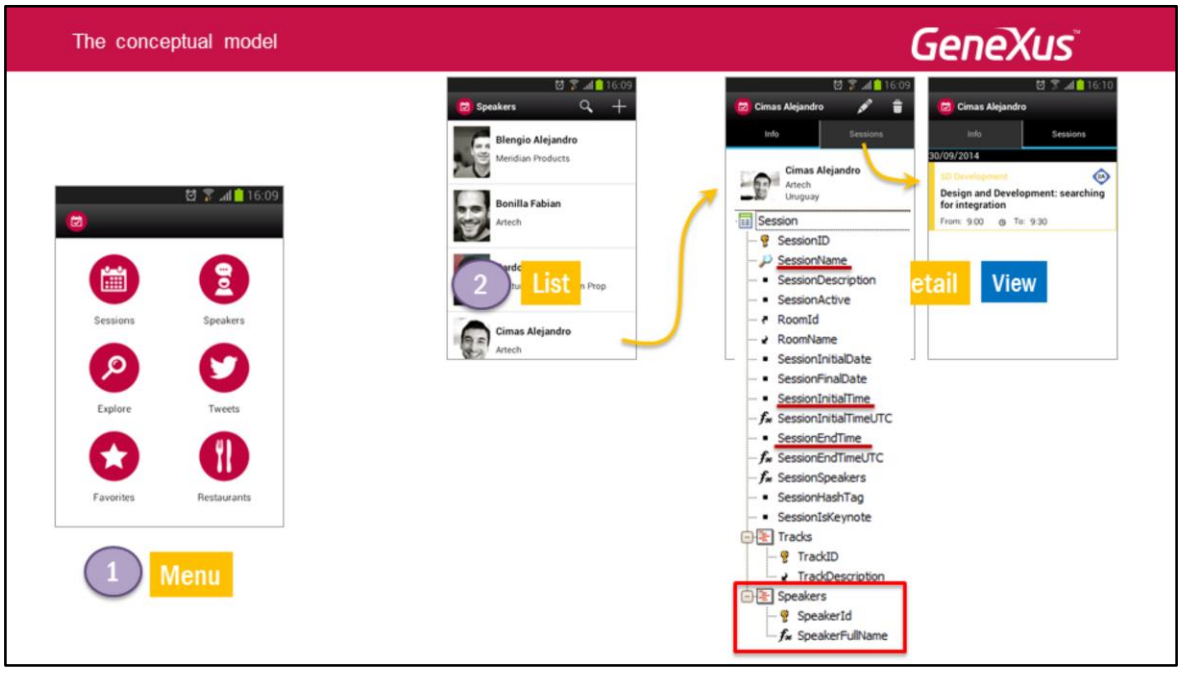
2. **Lista** de elementos con los que trabajar: suele corresponder al conocido "Trabajar con"… los datos de una transacción. La manera de presentar la información de la lista es configurable (así como lo es para el menú). En el ejemplo estamos viendo la pantalla de List de Speakers, donde se está mostrando el Full Name, la Image y Company Name de cada orador. Lo esencial es que esta pantalla, en forma predeterminada, establece que al hacer tap sobre un elemento de la lista, automáticamente se llame en modo View a la pantalla de Detail (la del punto que sigue). Igualmente, incluirá una acción para llamar a la pantalla de Detail (la del punto siguiente), pero esta vez en modo Edit, para insertar.



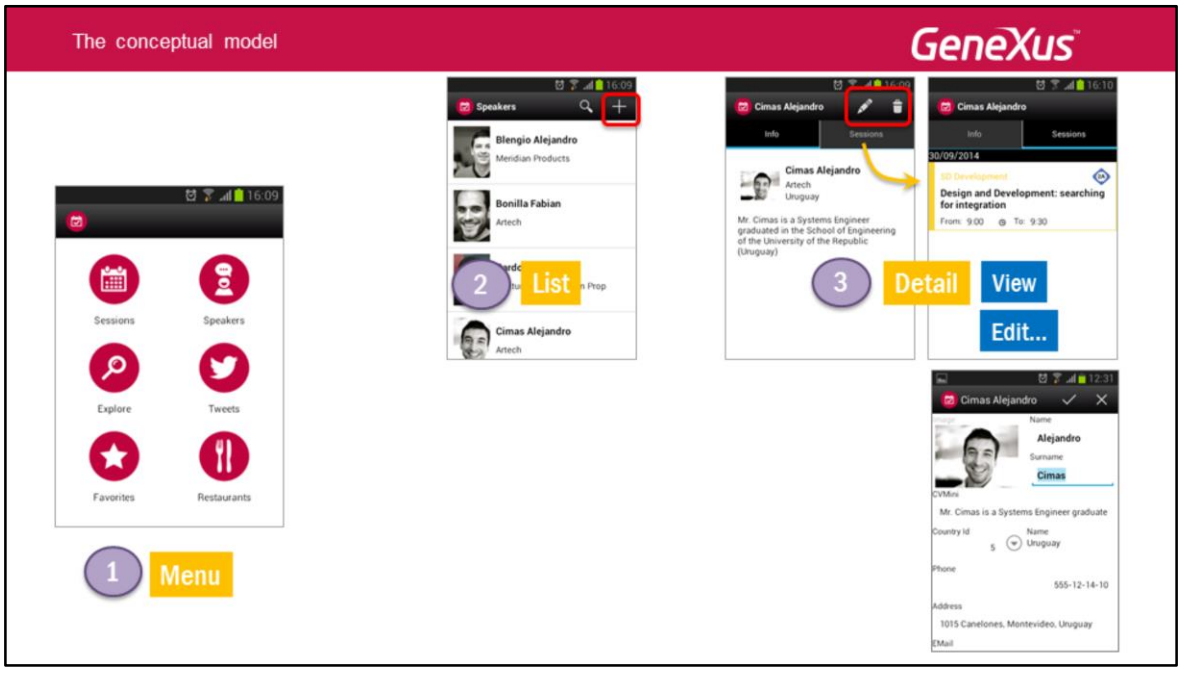
La pantalla de List, en forma predeterminada, establece que al hacer tap sobre un elemento de la lista, automáticamente se llame en modo View a la pantalla de Detail (la del punto que sigue). Igualmente, incluirá una acción para llamar a la pantalla de Detail (la del punto siguiente), pero esta vez en modo Edit, para insertar un nuevo elemento (en nuestro caso, un nuevo Speaker).

3. Visualización del **Detalle** (Detail) de un elemento: esta pantalla permite visualizar la información detallada de un elemento de información usualmente seleccionado por el usuario, de una lista (pantalla del punto 2).

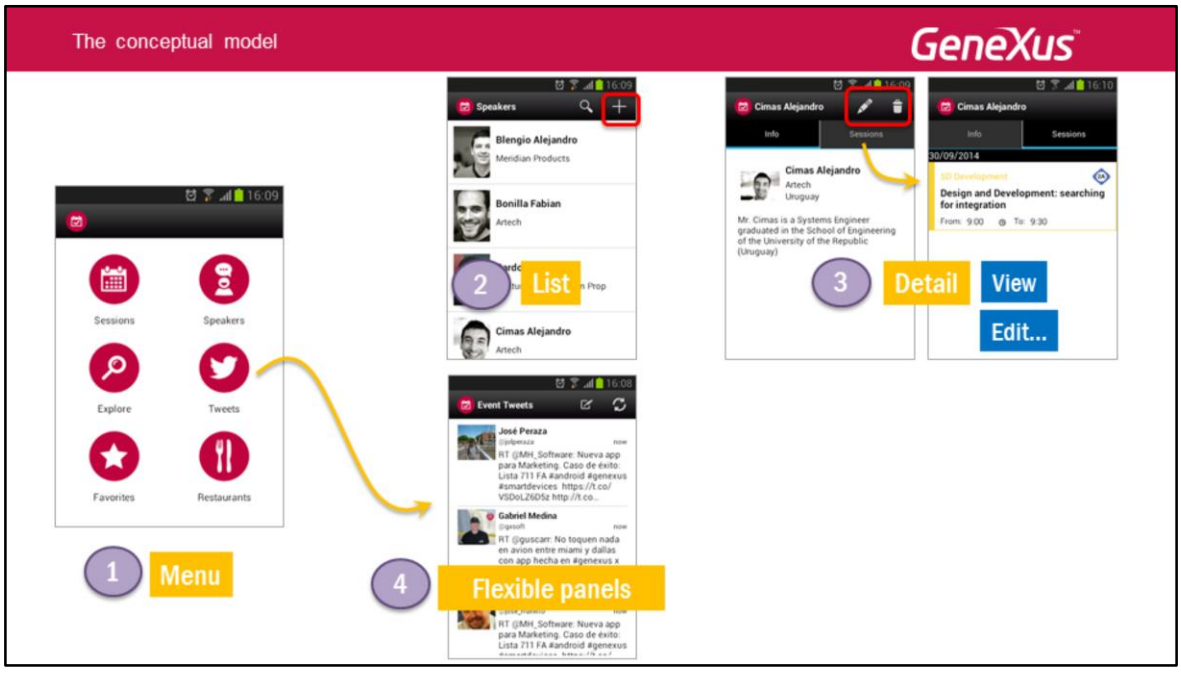
Podemos ver cómo por defecto en Android la pantalla de detalle se compone de dos pestañas: la primera que contiene la información general del elemento (en nuestro caso, el FullName, CVMini, Country Name y Company Name del speaker) y la segunda que contendrá la información relacionada…



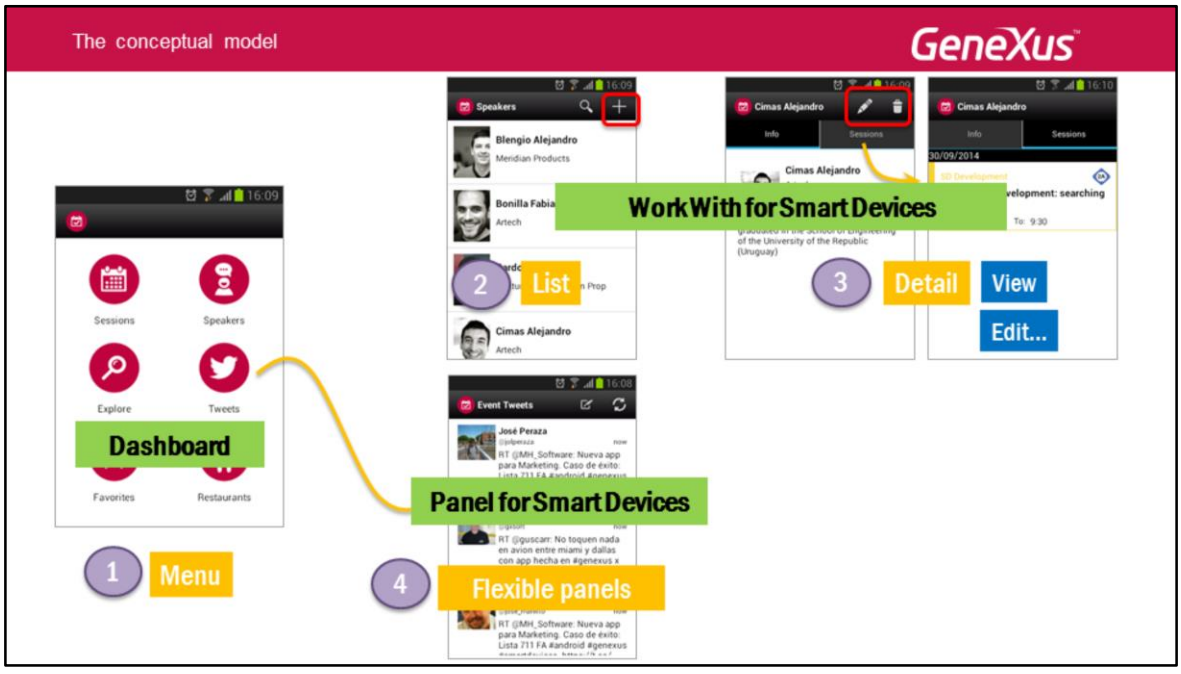
En nuestro caso, como los oradores estarán relacionados con las sessions, la segunda pestaña mostrará la lista de sessions del orador.



Tanto desde el List, como desde la pantalla de Detail en modo View, es posible acceder a otra pantalla de Detail, pero esta vez en modo Edit, para que el usuario pueda realizar las operaciones de CRUD (Create/Update/Delete) sobre la información. Con lo que el usuario ingrese en esta pantalla, se llamará al Business component que realizará la operación sobre la base de datos.

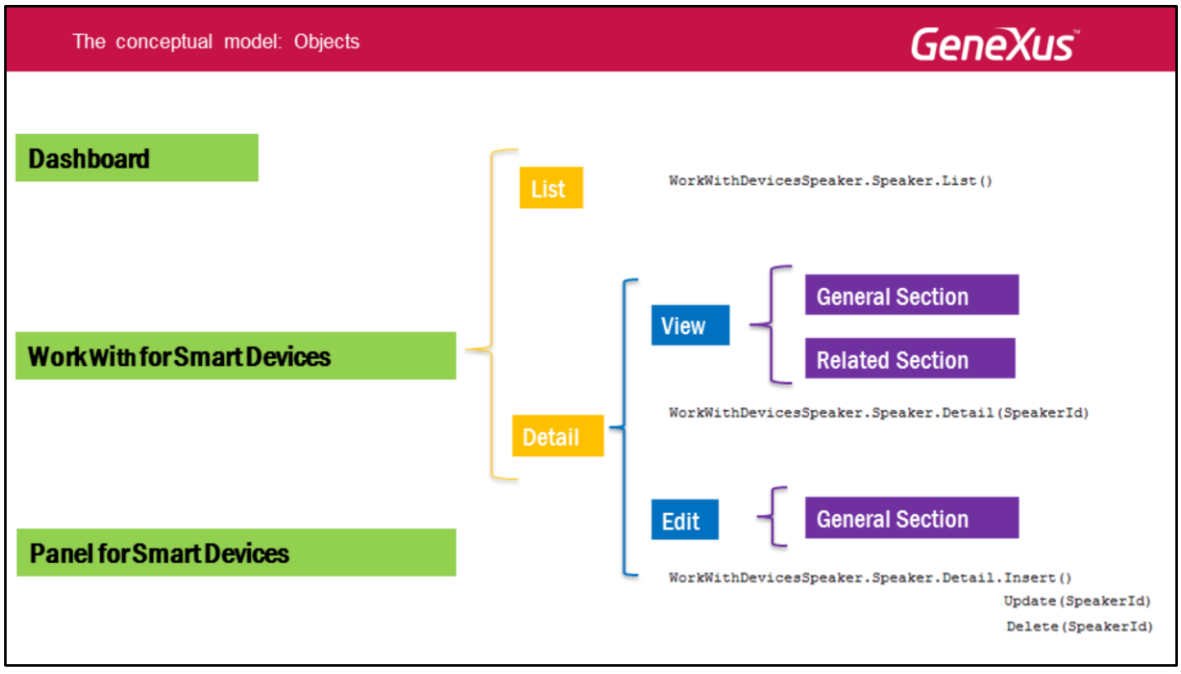


4. Por último, tenemos paneles flexibles que proveen toda la libertad para diseñar el layout sin estructuración preestablecida. Todo lo implementa el desarrollador desde cero. Por ejemplo, la pantalla de tweets, cuyos datos serán obtenidos a través de un web service.



Para implementar estas pantallas y sus funcionalidades, tenemos tres objetos:

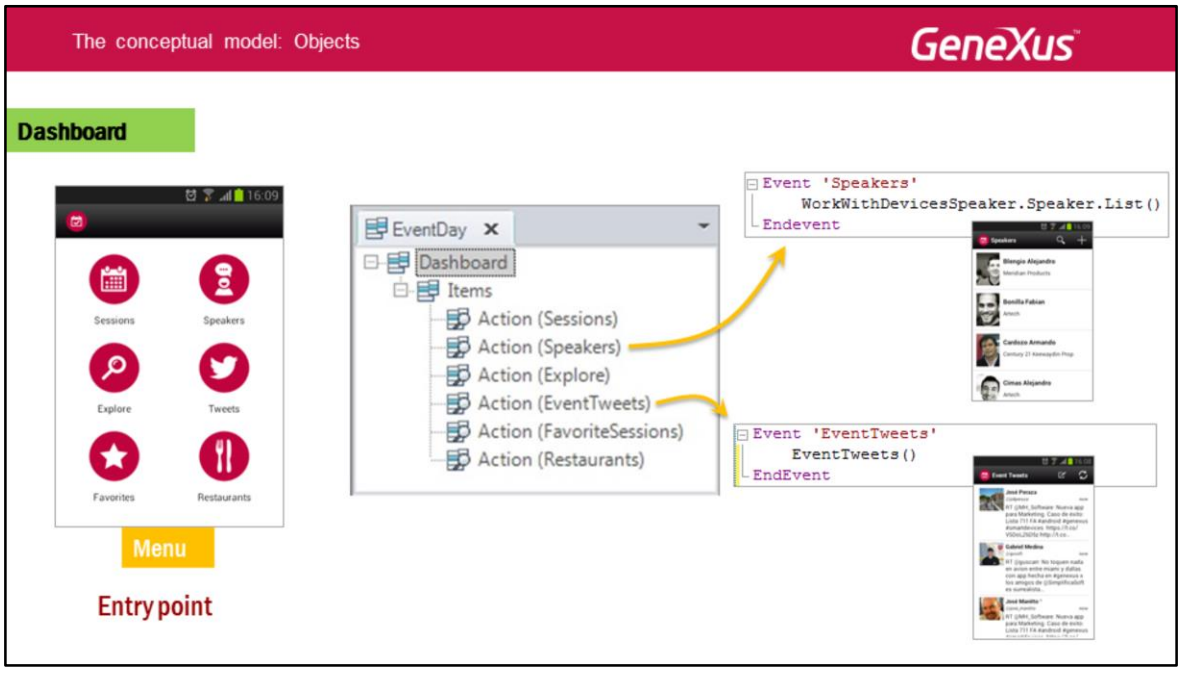
1. **Dashboard**: implementa el menú.
2. **Work With for Smart Devices**: es un objeto y un pattern, dos en uno. A diferencia del Work With for Web, no es un archivo de instancia a partir del cual se crean objetos que implementan el típico "trabajar con" los datos de una transacción, sino que es un objeto que incluye en su interior esas especies de sub-objetos: los que implementan las pantallas de List, y de Detail –con todas sus secciones (tabs)–. Generalmente este objeto estará asociado a una transacción, pero el hecho de que sea un objeto que puede crearse desde el diálogo de creación de objetos, le da la flexibilidad de poder crear todas sus pantallas (List y Detail) de cero, cargando la información de donde se desee.
3. **Panel for Smart Devices:** son análogos a los Web panels. También son análogos a cualquiera de las pantallas de List o de cada sección del Detail del Work With for Smart Devices. Se crean de cero, por lo que el desarrollador tiene la absoluta libertad de colocar y quitar lo que desee, y cargar los datos de la base de datos o de cualquier otro lugar.



Aquí indicamos cómo el objeto Work With for Smart Devices encapsula el list y el detail, y a nivel de este último, las pantallas de visualización de la información general y relacionada, y la pantalla de edición (CRUD) de la información general.

Para invocar a cada uno de los tres (list, detail en modo de visualización, detail en modo de edición) la sintaxis es la que se muestra, tomando como ejemplo el pattern Work With for Smart Devices aplicado a la transacción Speaker, de un nivel. Para el caso de edición, la invocación siempre es indicando la operación (Insert, Update, Delete).

Veamos algunos detalles de cada uno de los objetos.



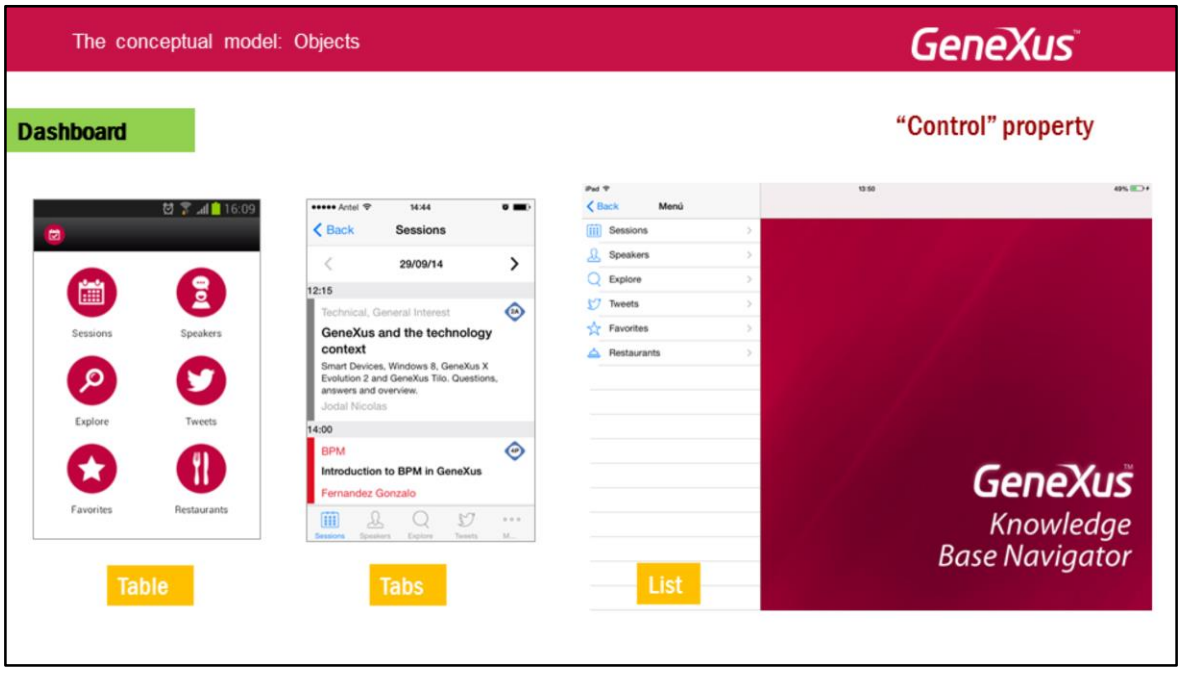
El objeto Dashboard consistirá esencialmente en ítems que especifican las acciones que serán ofrecidas al usuario. Cada una de ellas corresponderá a un evento donde se programa a quién invocar como consecuencia de elegir el ítem.

Al insertar cada acción, automáticamente se abrirá la ventana Select Objects para que elijamos el objeto al que queremos invocar como resultado de esa acción. Automáticamente quedará programada la invocación en los eventos, que, por supuesto, podemos modificar.

Puede haber muchos objetos dashboard dentro de la aplicación (todas las veces que se necesite utilizar un menú).

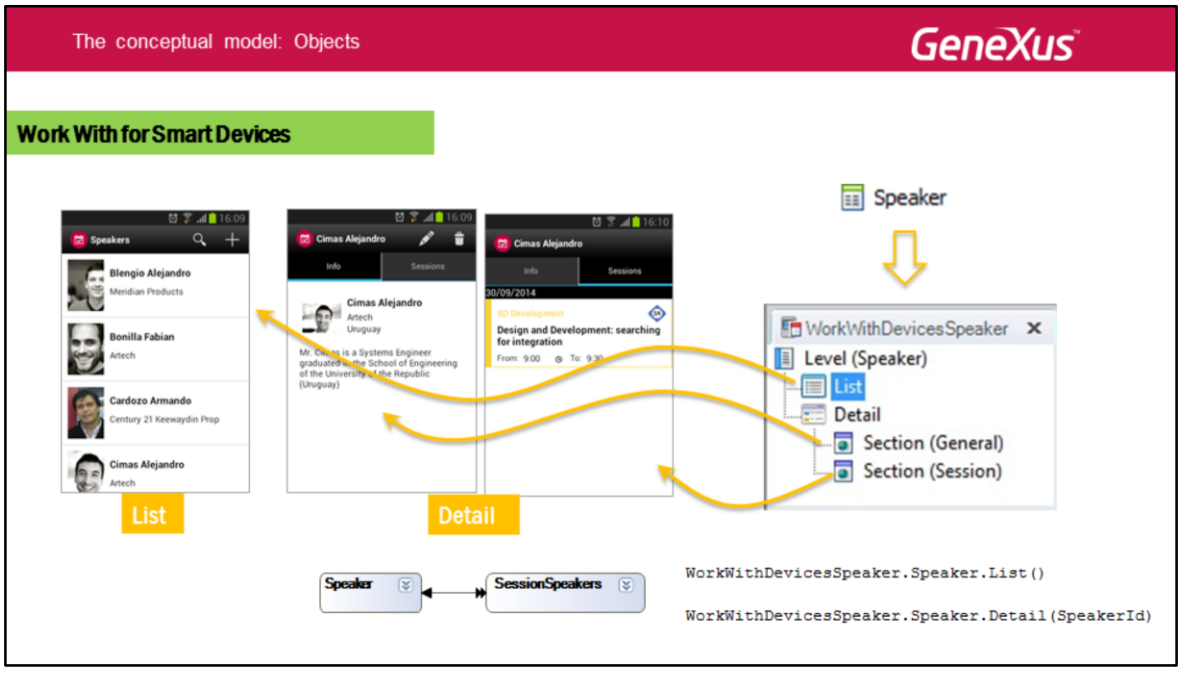
A diferencia de las aplicaciones Web, que permiten múltiples puntos de entrada (a partir de URLs), las aplicaciones para Smart Devices, que estarán instaladas en el dispositivo, tienen un único punto de entrada. El punto de entrada suele ser un menú, por tanto un objeto Dashboard.

Es por ello que los dashboard, por defecto, son objetos Main.



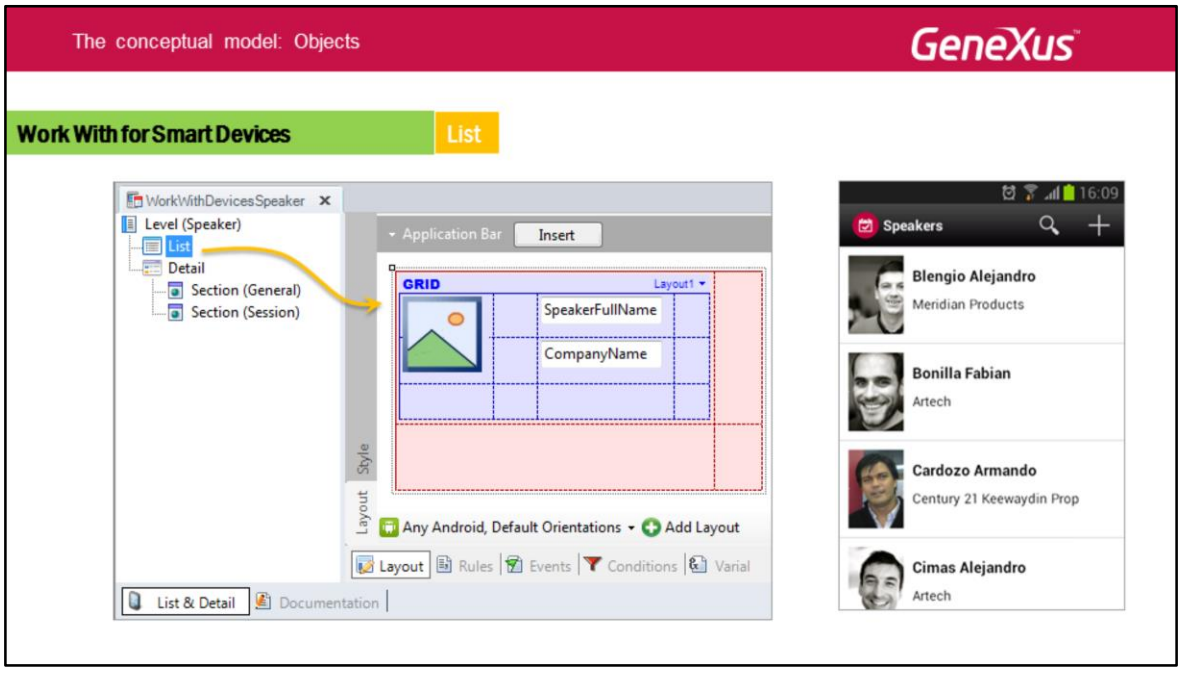
Si bien cada plataforma tiene su estándar, podemos elegir que los ítems del menú se muestren en forma de tabla (el default de los Android phones), en forma de tabs (el default de iPhones) donde el área central aparece cargada con el objeto llamado por el primer ítem (en nuestro caso, Sessions), o en forma de List (el default de las tablets).

Para modificar el comportamiento default, existe una propiedad del dashboard que permite seleccionar alguno de estos tres valores: Table, Tabs o List: la propiedad **Control**.



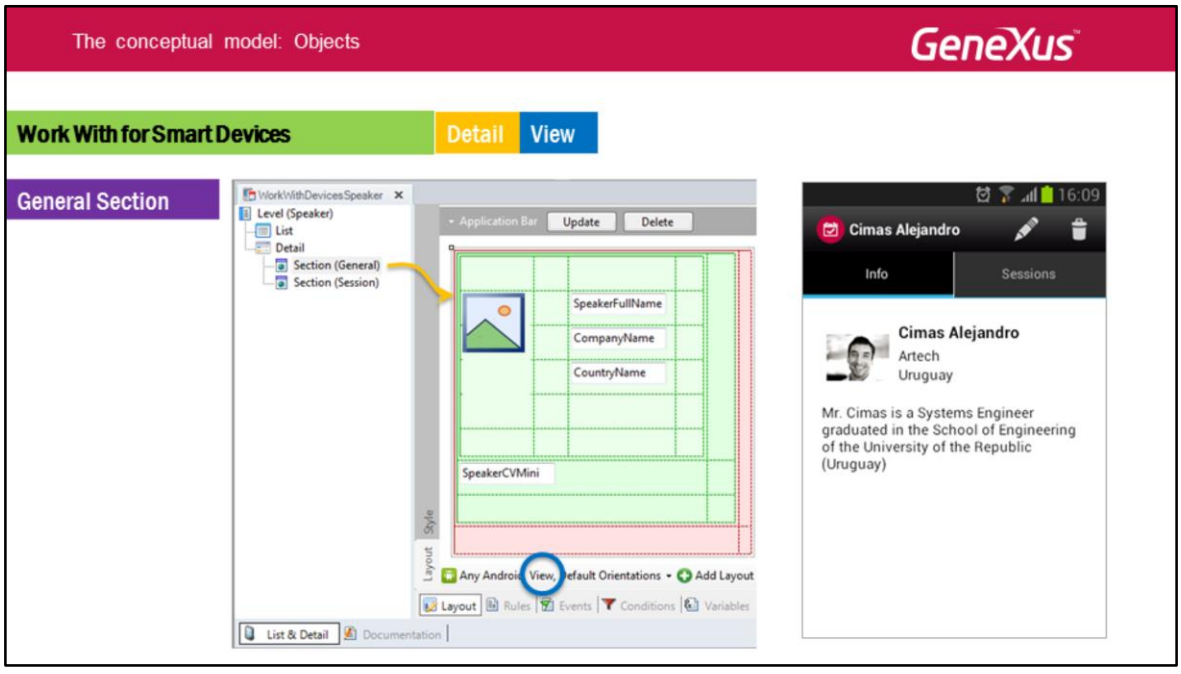
Partiendo de la transacción Speaker, aplicarle el pattern “Work With for Smart Devices” dará como resultado la creación de un objeto de ese tipo, que presentará una **estructura jerárquica** que implementa tanto el **List** como el **Detail**. El desarrollador podrá personalizarlo, trabajando directamente sobre sus pantallas –como podemos ver en las páginas siguientes–, agregando o quitando niveles, o, incluso, eliminando el List o el Detail, o quitando o agregando sections a este último.

Al aplicar el pattern a la transacción de un nivel, Speaker, podemos ver que aparece un nivel que corresponde a ese nivel de la transacción, con la mencionada estructura jerárquica. El **Detail** consiste de varias secciones: la **general**, correspondiente a la información de ese nivel de la transacción, y una sección por cada tabla de la base de datos con información directamente subordinada (en este caso, la de las conferencias en las que el speaker participa).



¿Dónde se encuentra diseñada la pantalla del List que vemos en ejecución? (pantalla de la derecha)

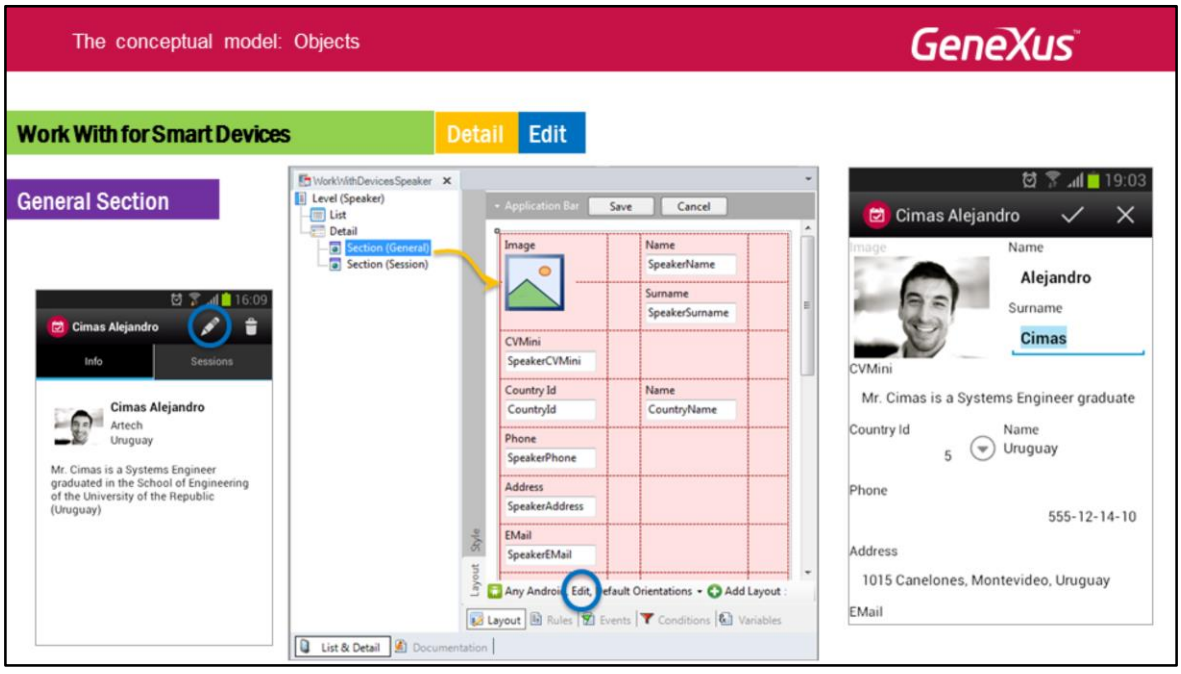
En el nodo List del “Work with”. Vemos que aparece un grid con los atributos SpeakerImage, SpeakerFullName y CountryName. Este layout puede personalizarse como cualquier otro layout. Obsérvese que asimismo se cuenta con una pestaña Events donde se programarán los eventos, por ejemplo, donde viene programado por defecto el código del evento asociado al botón “Insert” que podemos ver en la Application Bar.



¿Y dónde la pantalla de visualización del detalle de la información general de un speaker?

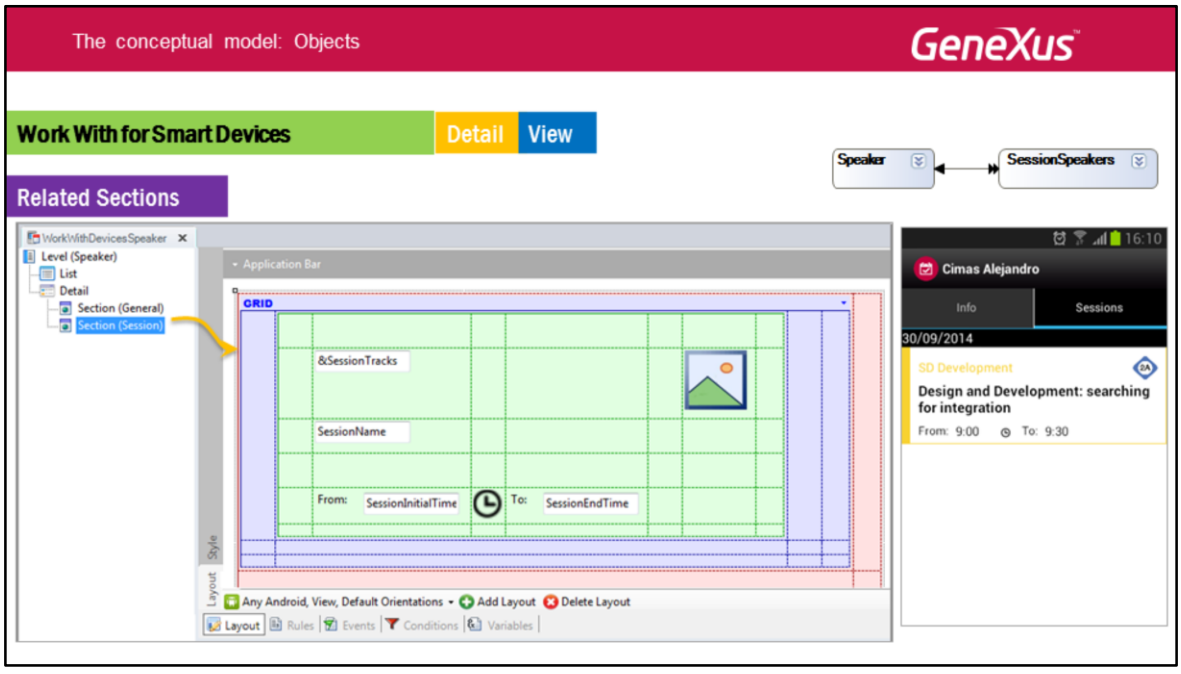
En el nodo Section(General) del nodo Detail en modo View. El combo que se señala abajo, donde vemos la opción “View” permite cambiarse de layout, al de Edit…

Aquí podemos ver una tabla con filas y columnas, dentro de la que se colocaron los controles asociados a los atributos.



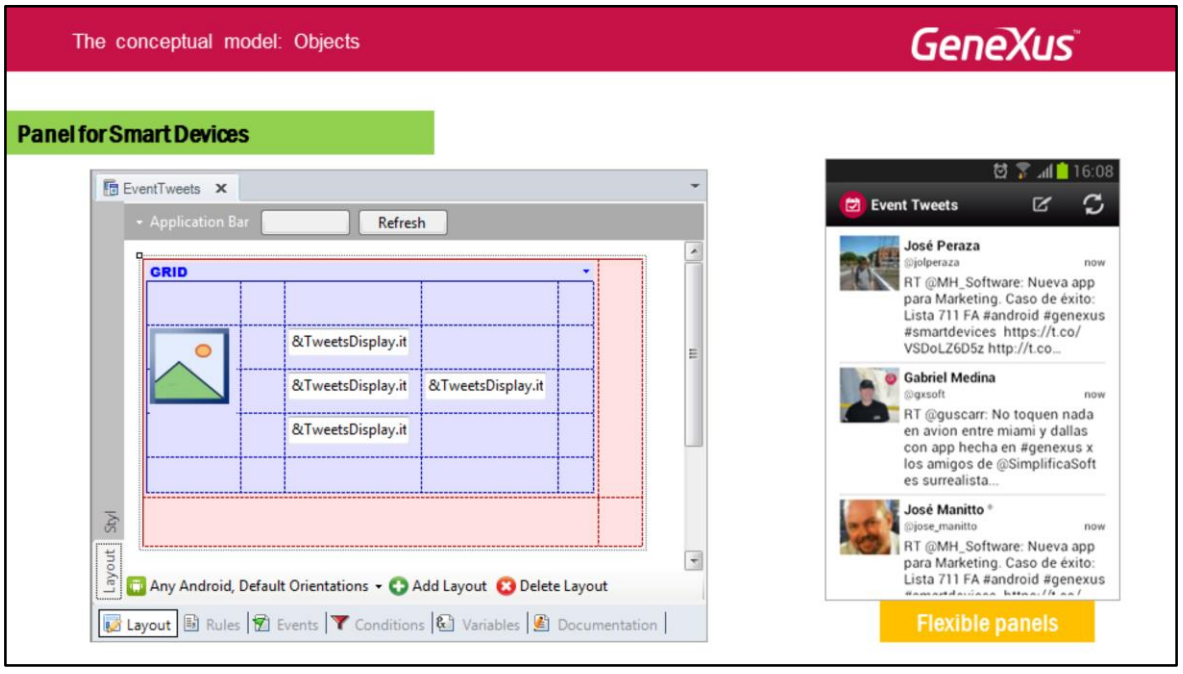
¿Dónde está implementada la pantalla que permite las operaciones de CRUD (Create/Update/Delete) –aquella a la que accedíamos, por ejemplo cuando hacíamos tap sobre el ícono de Update de la pantalla de view)?

Es también en el nodo Section(General), pero eligiendo la pantalla de Edit del combo inferior. Es decir, para la “Section (General)” tenemos dos layouts independientes (por más que sean inicializados con exactamente los mismos atributos): el layout de la pantalla de View, y el layout de la pantalla de Edit.



¿Y dónde está la pantalla correspondiente a las conferencias de ese speaker, que son visualizadas cuando se abre la pantalla de Detail en modo view?

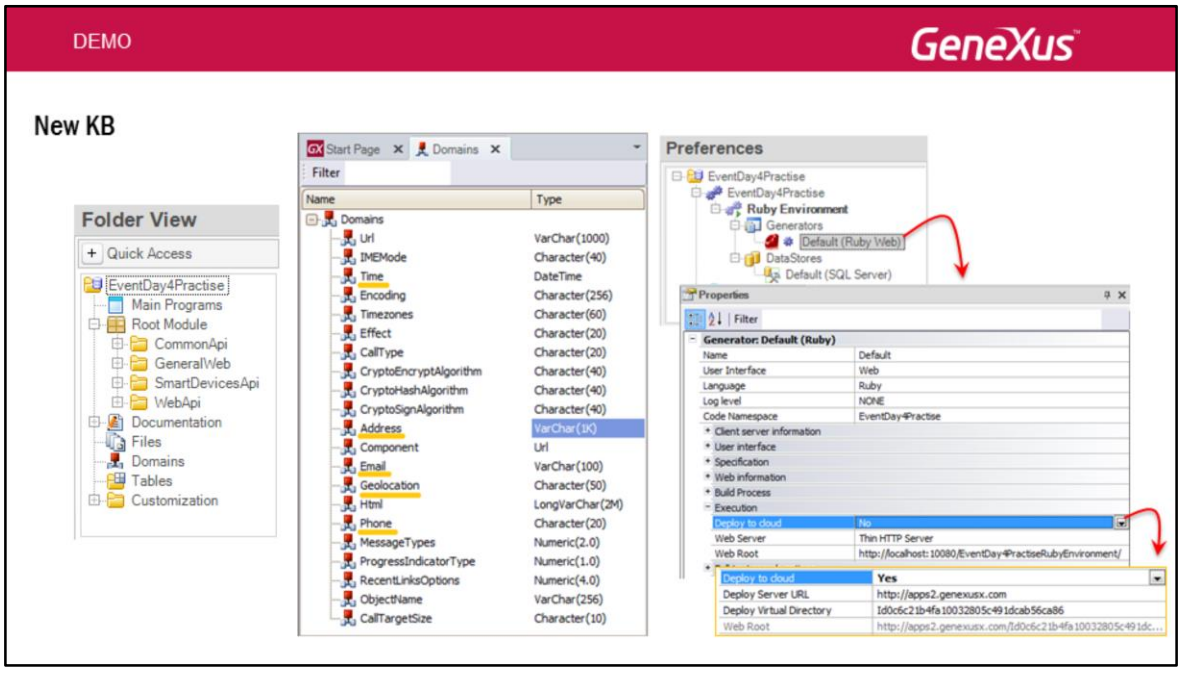
En el nodo **Section(Session)** correspondiente a datos de la tabla subordinada a speaker. También estará compuesto por un grid.



Para implementar un panel desde cero, como el de los tweets, tenemos un objeto que se parece en todo a cualquiera de los nodos del Work With. La diferencia respecto al work with aplicado a una transacción, es que mientras las pantallas del work with ya aparecen inicializadas, el panel estará por completo vacío. No tiene ningún tipo de lógica asociada.

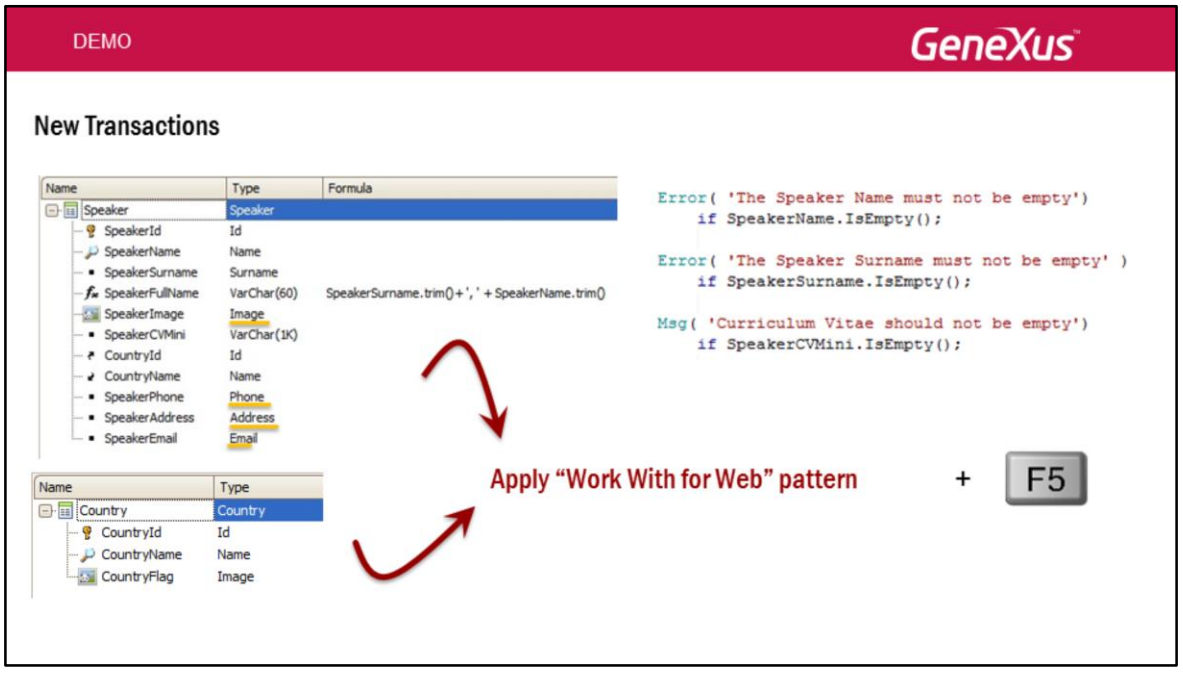
En este panel, en particular, podemos ver cómo el desarrollador ha insertado en el layout un grid asociado a un SDT colección, que será cargado en el evento Refresh llamando a un web service de Twitter. Para visualizar uno de los tweets, usamos la propiedad CurrentItem de la variable SDT, que devuelve el elemento de la colección del SDT que corresponde a la línea seleccionada.

Demo: starting to develop the application



Creamos una KB de cero. Observamos que aparecen los folders CommonApi, GeneralWeb, SmartDevicesApi y WebApi, y los dominios predefinidos (entre ellos, los dominios semánticos: Time, Address, Email, Geolocation, Phone, cuya semántica veremos luego en acción).

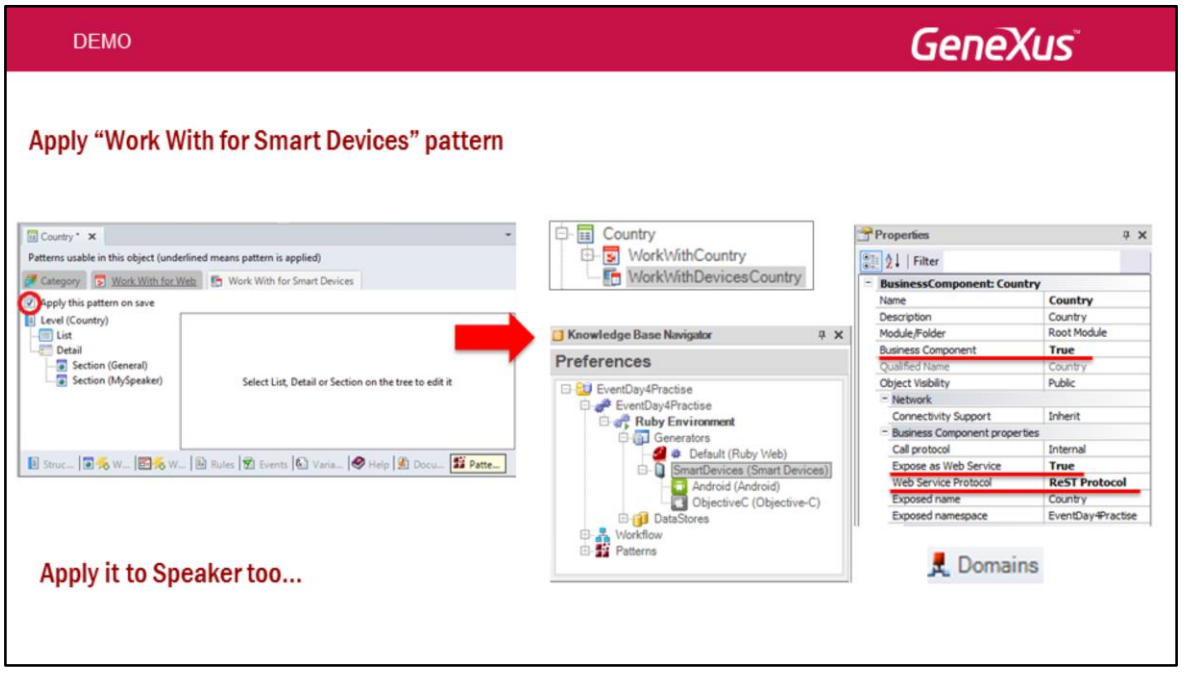
Vemos cómo por defecto la propiedad “Deploy to Cloud” del Generador web Default aparece en No. La prendemos y vemos cómo automáticamente se modifican las propiedades “Deploy Server URL” y “Deploy Virtrual Directory”, para apuntar a la nube.



Crear las dos transacciones que se indican y observar cómo se están utilizando los dominios semánticos (veremos luego en ejecución cómo reacciona la aplicación a esa semántica). Especificar las reglas que se muestran para la transacción Speaker.

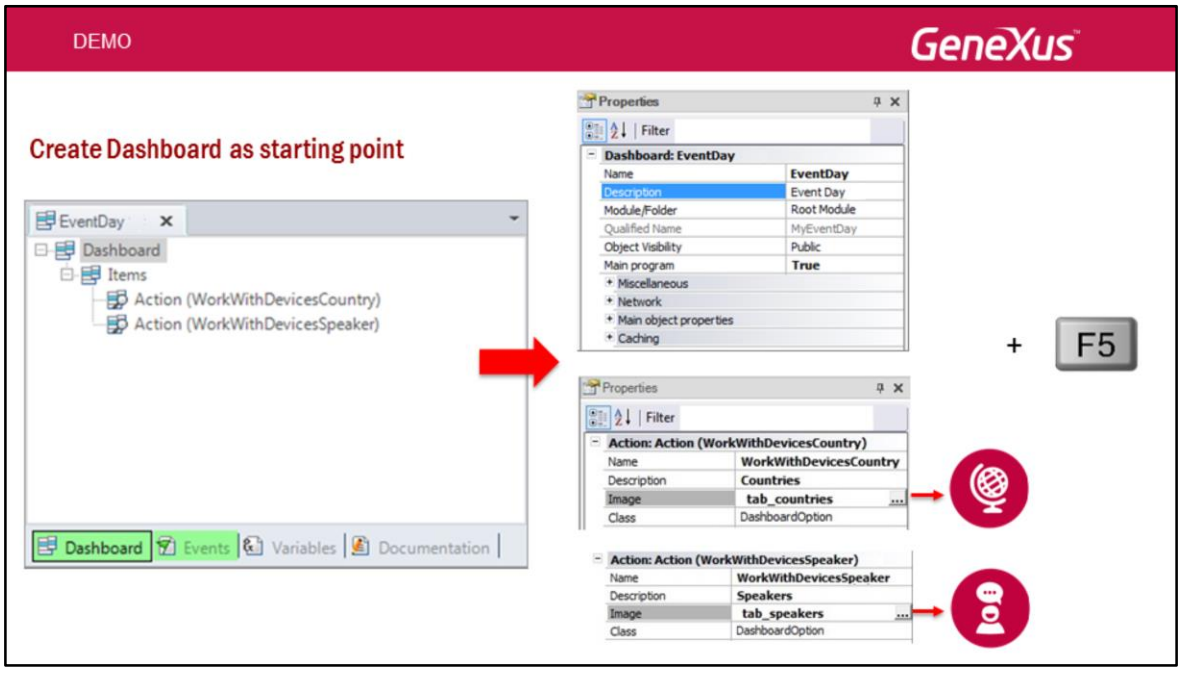
Aplicar los patterns "Work With for Web" a ambas transacciones y hacer F5.

Se abrirá el Developer Menu, con tres links: uno a Home, creado por el pattern, y otros dos a ambos WW.



Volver a GeneXus y aplicar ahora el pattern “Work With for Smart Devices” a la transacción Country. Alcanza con marcar el check box “Apply this pattern on save” y grabar.

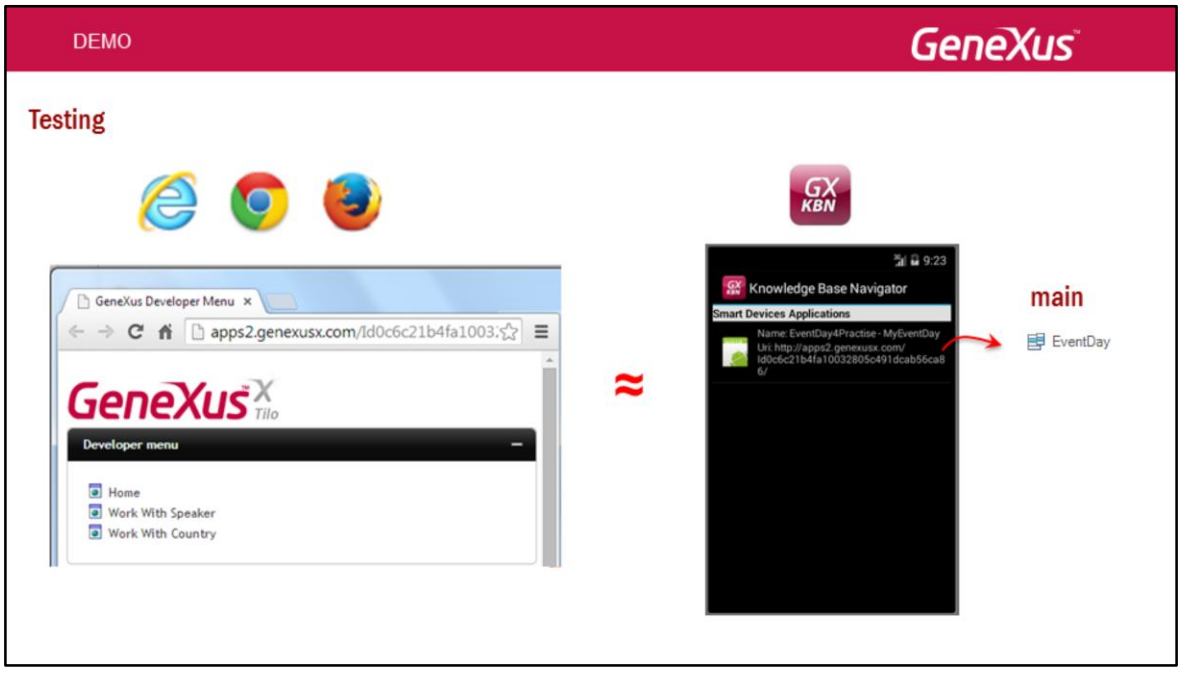
1. Así como al aplicar el pattern web aparecía la instancia en el Folder View, bajo la transacción, al aplicar el pattern para Smart Devices también aparecerá la instancia (pero como objeto): WorkWithDevicesCountry.
2. Se habrán modificado automáticamente ciertas propiedades de la transacción (que entenderemos cuando estudiemos la arquitectura de estas aplicaciones), en particular se prendió la propiedad “Business Component”, que se expondrá como Web Service.
3. Se habrá agregado automáticamente el generador para Smart Devices, que por default generará para Andoid y para iOS, siendo la plataforma principal Android (esto puede cambiarse). En el folder SmartDevicesApi de la KB se agregan una gran cantidad de objetos externos.
4. Se agregan más dominios, específicos para trabajar con Smart Devices.



Ahora debemos crear el dashboard, que será el punto de entrada de nuestra aplicación para Smart Devices.

Observar cómo automáticamente asume la propiedad Main program = True.

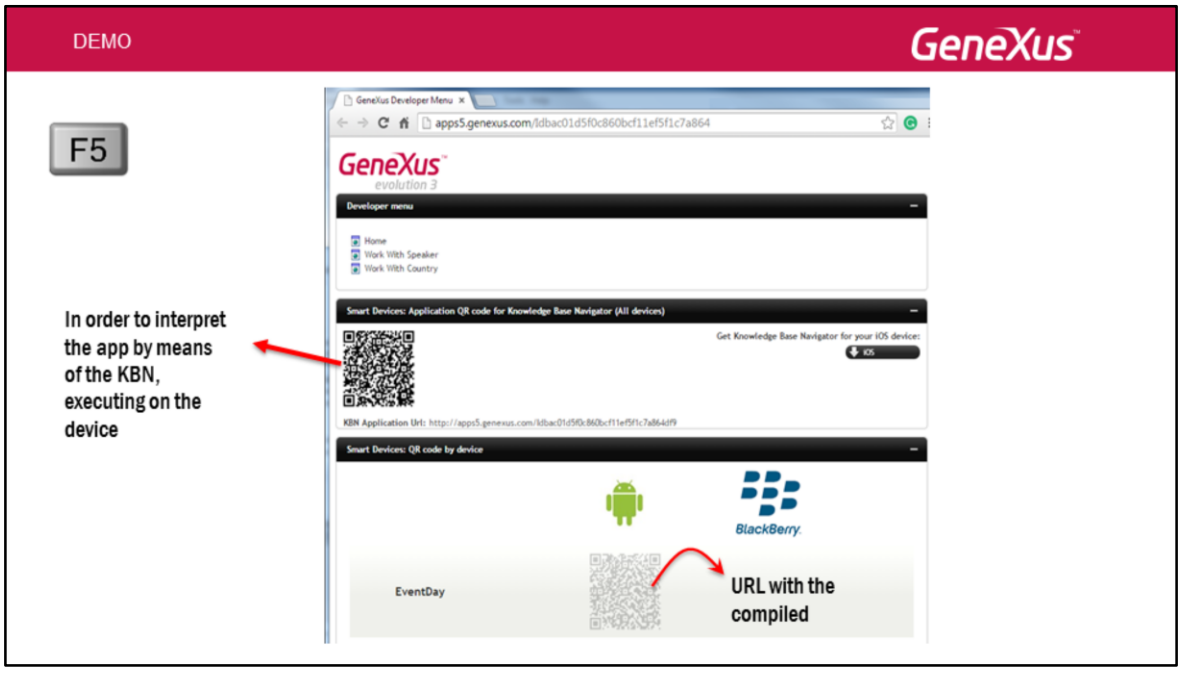
Ya estamos listos para ejecutar… ¡F5!



Mientras en Web tenemos, para facilitar la prototipación, un **Developer Menu** que permite ejecutar todo **objeto web** que no reciba parámetros, en Smart Devices tenemos un programa conocido como **KBN** (Knowledge Base Navigator), que en primera instancia podemos pensar como un navegador (intérprete) de **objetos smart devices**. Pero a diferencia de en web, donde los objetos no tienen por qué ser main para ser linkeados desde el Developer Menu, desde el KBN sí.

Por tanto, si tenemos como main únicamente al dashboard que creamos, EventDay, al dar F5 se levantará, además del browser default con el Developer Menu, el **emulador de Android** con el **KBN** ofreciendo los objetos main de Smart Devices (dado que por defecto la plataforma main de generación SD será Android, y no tenemos un dispositivo conectado a nuestra máquina. Para ver cómo prototipar directamente en el dispositivo, en http://training.genexus.com/gx-for-smart-devices-course-xev3?es busque el video de nombre “Prototipado y ejecución de aplicaciones móviles”.

Recordemos que una diferencia importante entre las aplicaciones web y las aplicaciones para Smart Devices, es que mientras las primeras no tienen un punto de entrada único, las segundas sí lo tienen.

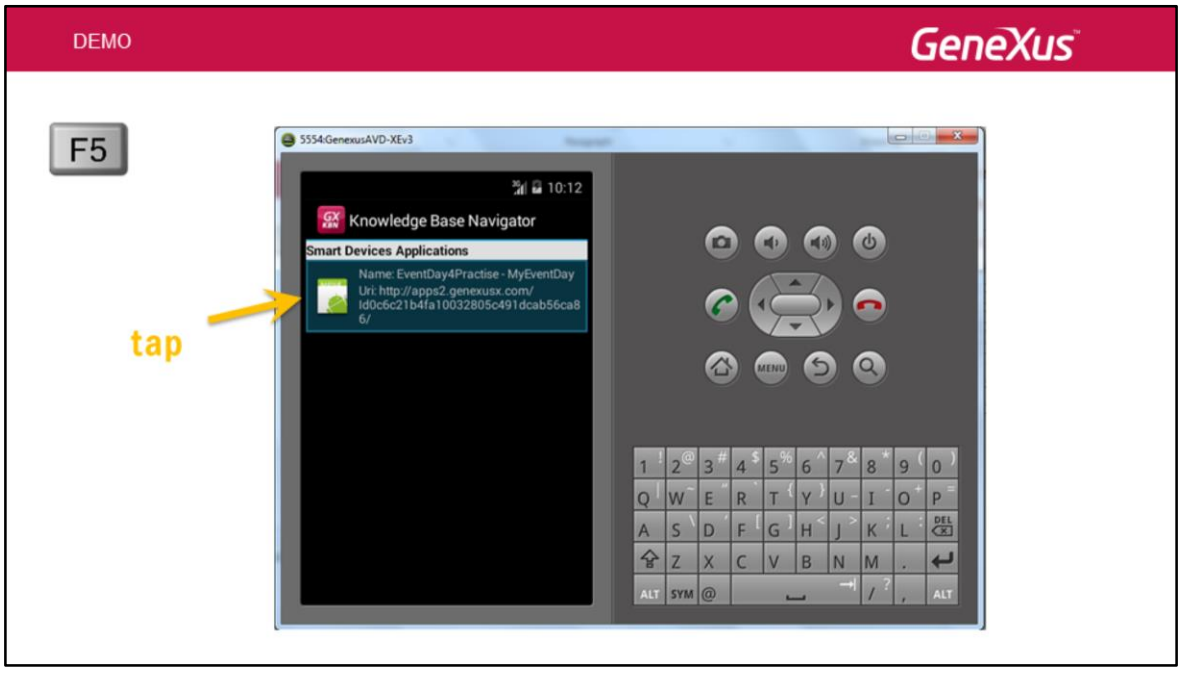


Teniendo, entonces, como único objeto SD main el dashborad, al hacer F5 veremos que:

1. Se generan todos los programas, y se abre el Developer Menu, que además de los links a los objetos web, contiene unos QR codes: uno que encapsula la URL para poder ejecutar desde el KBN en el dispositivo que corresponda (a la derecha aparece el link para descargar el KBN para iOS, dado que cada vez más se aconseja probar la aplicación compilada, ya que en el KBN las funcionalidades están reducidas; el de iOS se sigue disponibilizando puesto que se necesita una Mac para compilar, lo que restringe las posibilidades de test); los otros QR codes contendrán la URL para descargar la aplicación para Smart Devices compilada, tanto para Android como para Blackberry. Para Windows 8 o Phone, el compilado no puede descargarse directamente en un dispositivo, y tampoco se tiene KBN puesto que no crea metadata para la UI. La forma de prototipar allí es a través del emulador específico o enchufando el dispositivo a la computadora.

   No siempre se creará el archivo compilado. Dependerá de si se tiene "Startup object" configurado y de la modalidad de ejecución que se empleó (si F5, Run o Run with this only sobre un objeto main, etc.) Lo veremos cuando estudiemos la arquitectura. En nuestro caso aún **no tenemos el archivo compilado**, por lo que aparecen deshabilitados.

2. Se abre automáticamente…

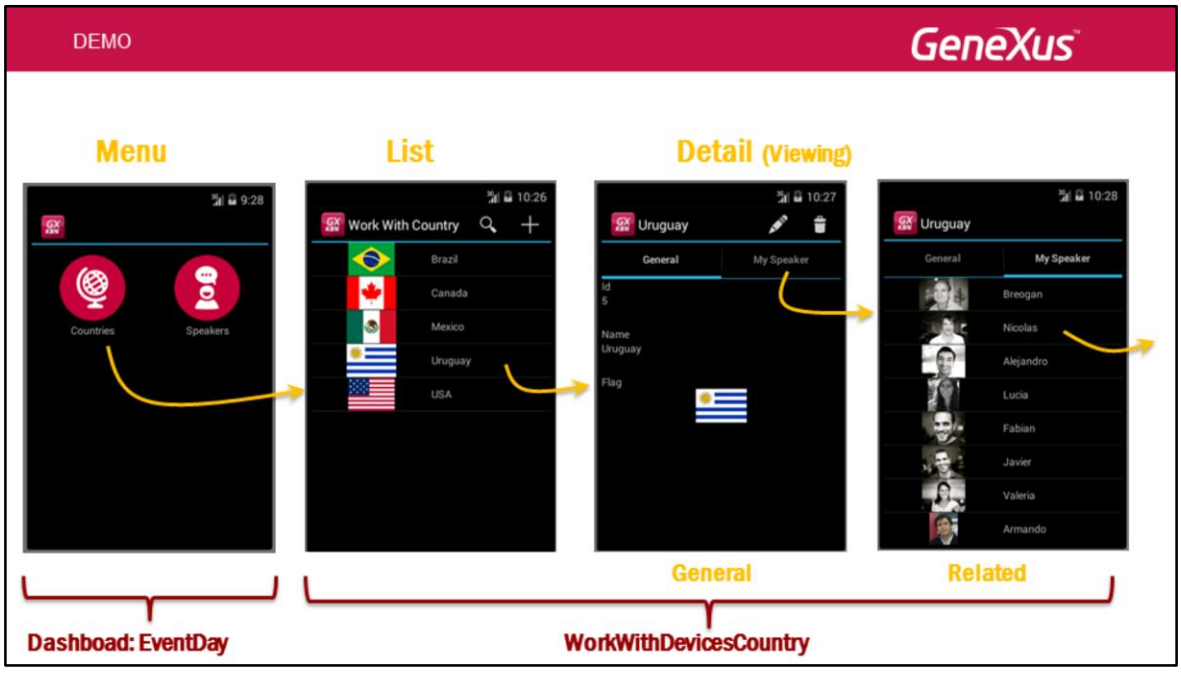


2. … el emulador de Android (a menos que usted tenga un dispositivo Android conectado al pc o notebook).

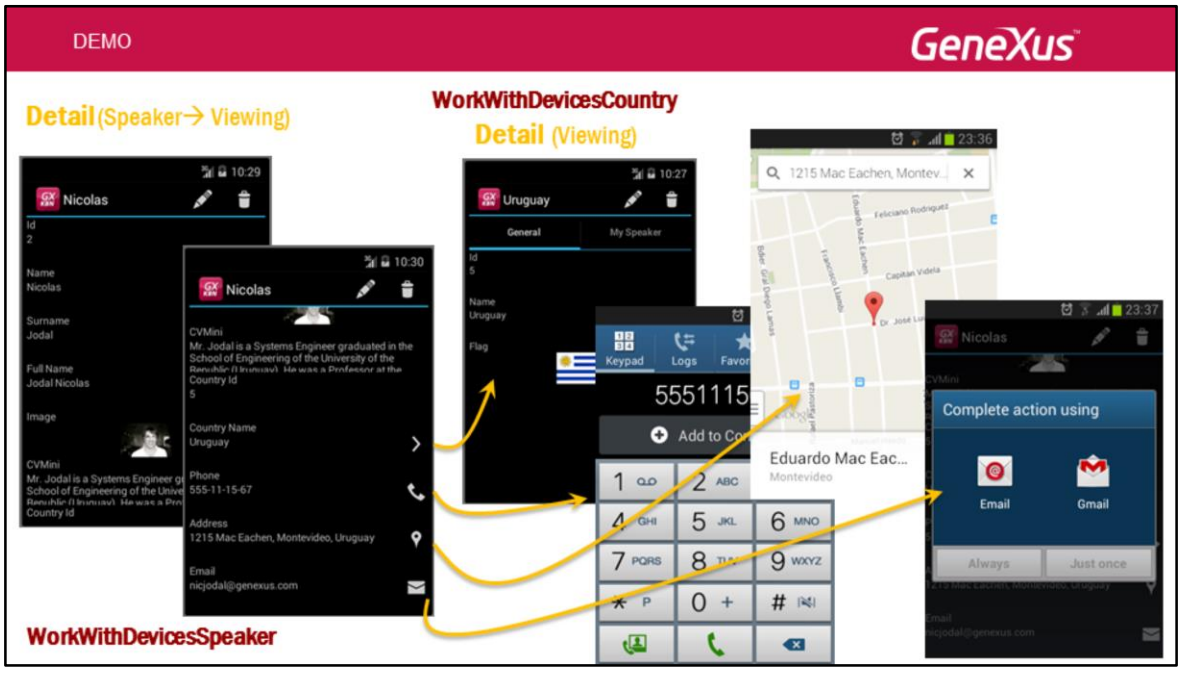
Recomendamos utilizar el emulador gratuito genymotion. Este emulador es bastante más veloz que el que viene con el SDK de Android. Le sugerimos descargárselo de genymotion.com y crearse los virtual devices en los que desee probar la aplicación. Para que GeneXus utilice este emulador en lugar del que viene con el SDK, bastará con que ejecute el virtual device antes del F5 y lo mantenga abierto luego.

Como vimos, mostrará las URLs de todos los objetos main para Smart Devices que hayamos definido (aquí uno sólo: el dashboard).

Al ejecutar esa URL (haciendo “tap”, es decir, el gesto del dedo de tocar la pantalla en un lugar preciso)…



Se abre el dashboard. Desde el menú accedemos al List de países. Desde allí, haciendo tap sobre Uruguay, vamos al Detail de ese país, viendo tanto su sección general, como la sección correspondiente a los speakers. Si sobre uno de éstos hacemos tap…

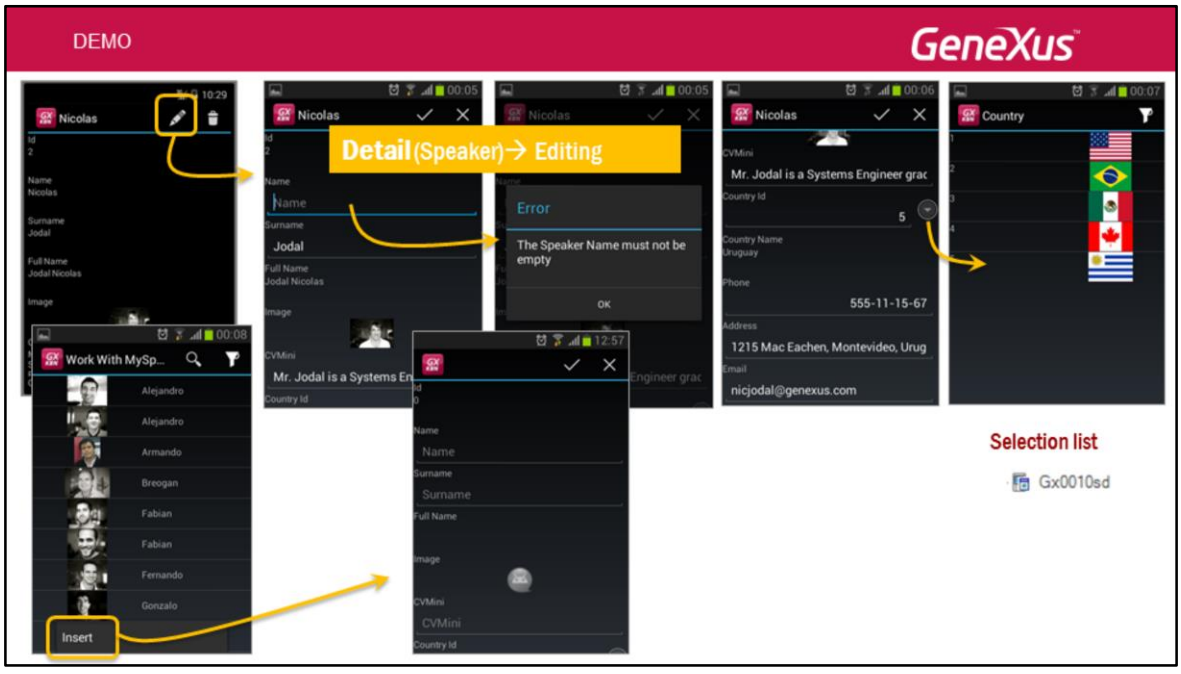


Nos lleva directamente al Detail del Work With de Speakers, para mostrar (en modo view) la información detallada de ese speaker elegido. Haciendo scroll por la pantalla, vemos que al lado de los campos correspondientes al país, al teléfono, a la dirección y al email, aparecen ciertos íconos.

Si hacemos tap sobre el del país, vemos que nos lleva al Detail del Trabajar con países, para mostrarnos la información detallada de ese país.

Si hacemos tap sobre el ícono del teléfono, nos abre la aplicación del dispositivo para realizar llamadas telefónicas. Aquí vemos en funcionamiento el dominio semántico Phone. Nos permite la integración con esa aplicación nativa.

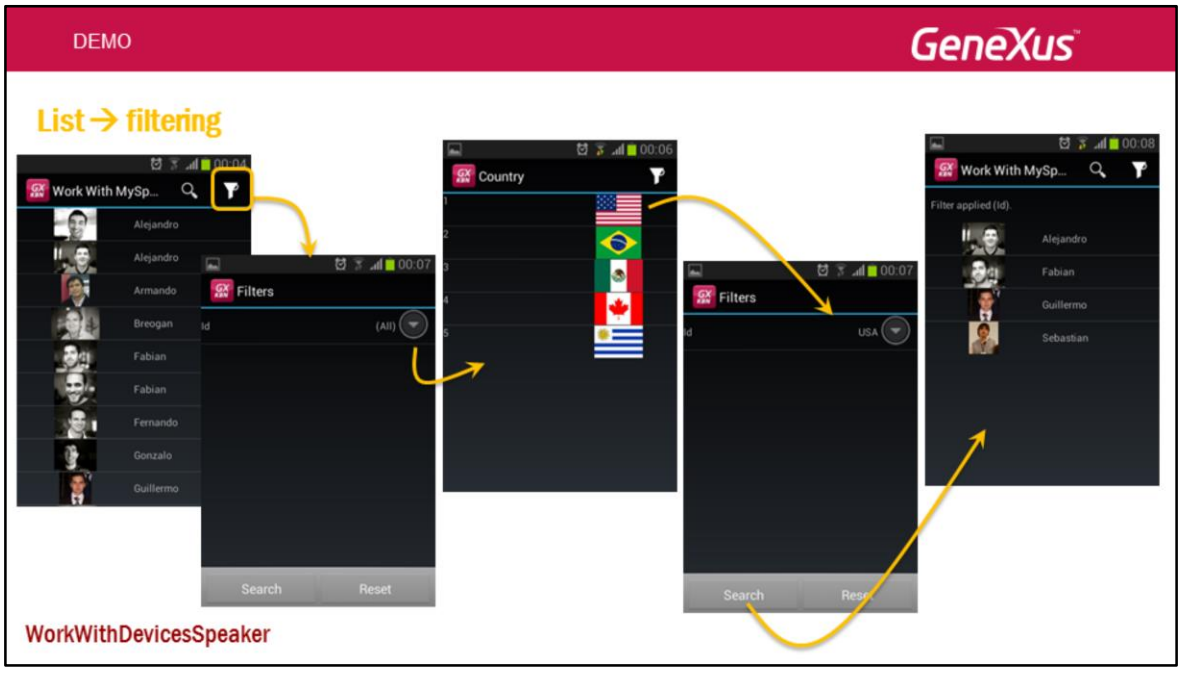
Igualmente, si hacemos tap sobre la dirección, vemos que se nos abre el mapa de google que tenemos instalado en el dispositivo, y si el tap lo hacemos sobre el mail, se nos pregunta cuál de las aplicaciones de correo instaladas en el dispositivo queremos utilizar para enviarle un email a esa dirección. Aquí vemos, entonces, la semántica incorporada en los dominios correspondientes.



Estando en la pantalla de Detail de un Speaker, podríamos desear modificar sus datos. Para ello, haciendo tap sobre el botón que aparece arriba, accedemos a la pantalla de Edit, de la Section(General).

A esa misma pantalla también accedemos si desde el List de speakers elegimos el botón de Insert. En un caso se estará llamando en modo Update, y en el otro, en modo Insert, a la misma pantalla. Supongamos que en el primer caso, eliminamos el nombre del speaker e intentamos grabar. Nos mostrará un error. ¿Dónde está especificado ese error y ese mensaje? ¡En la transacción Speaker! ¿Qué es lo que está sucediendo? Para actualizar la información se está ejecutando el business component.

Observemos, por otro lado, que dada la clave foránea CountryId, aparece un ícono a su lado, que nos permite llamar a una **lista de selección** para elegir el país que deseamos. Esta lista de selección será análoga a la que se crea para web. Será un objeto de tipo “Panel for Smart Devices”



Ahora, estando en el List de speakers (WorkWithDevicesSpeaker→List), queremos filtrar los oradores por país. Por ejemplo, queremos ver únicamente los oradores de Estados Unidos.
Vemos que se ha incluido automáticamente ese filtro, por clave foránea.
Dependiendo de los atributos de la transacción, se incorporan más o menos filtros. Si hubiera algún atributo de tipo Date, se habría incorporado un filtro por fechas.
También podemos hacer Searchs por distintos atributos.

Observar también que la información de la lista está saliendo ordenada por SpeakerName.

Sugerimos ir a GeneXus y sobre el grid del List, observar en las propiedades, aquellas bajo el grupo Data.

Todo es personalizable.

GeneXus

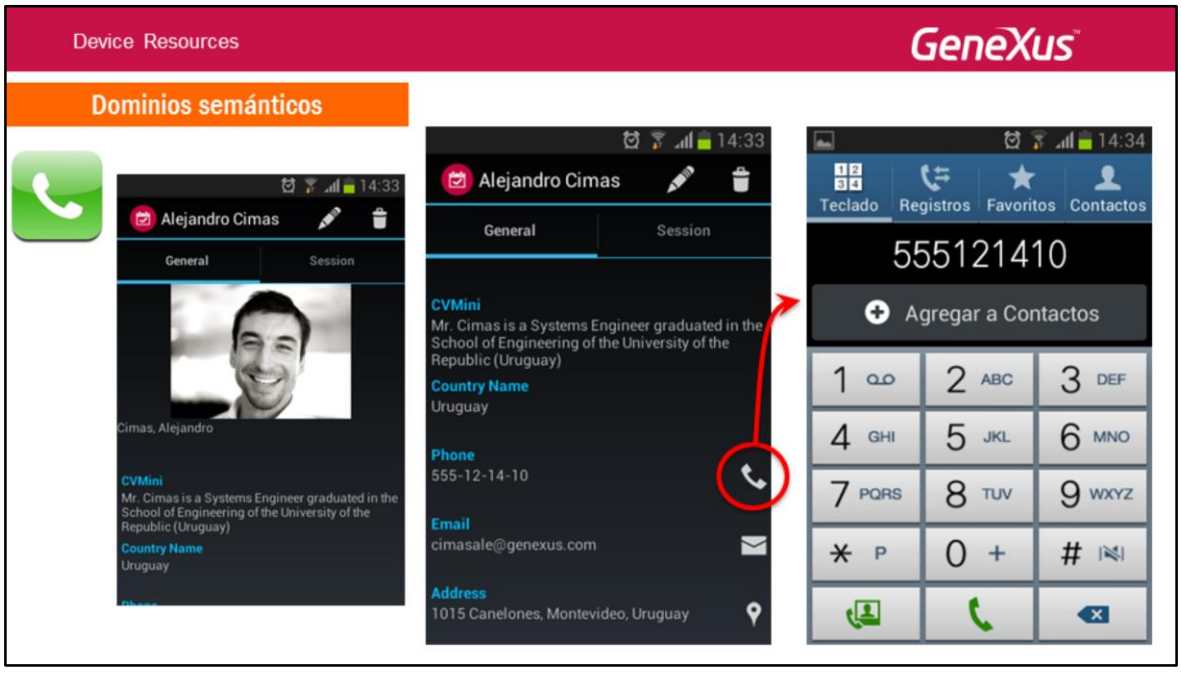
Características y plataformas

Interfaz y modelo conceptual

Integración con recursos del dispositivo

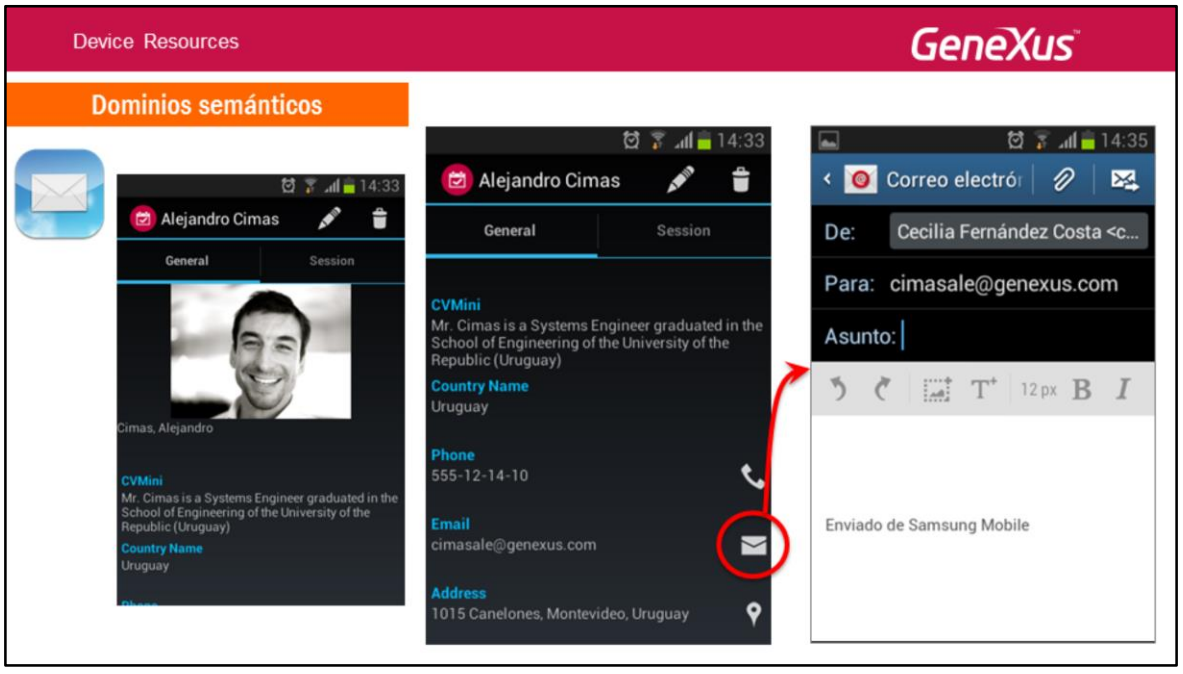
Veremos ahora cómo se integran las aplicaciones para Smart Devices con los recursos del dispositivo, como el software/hardware para realizar llamadas telefónicas, enviar mensajes (tanto chat como email), manejar la libreta de direcciones y contactos, utilizar la cámara, las aplicaciones de mapas o interactuar con Facebook y Twitter, entre otras cosas.

Esto se logrará utilizando los dominios semánticos, ciertos tipos de control, y las apis provistas por GeneXus.

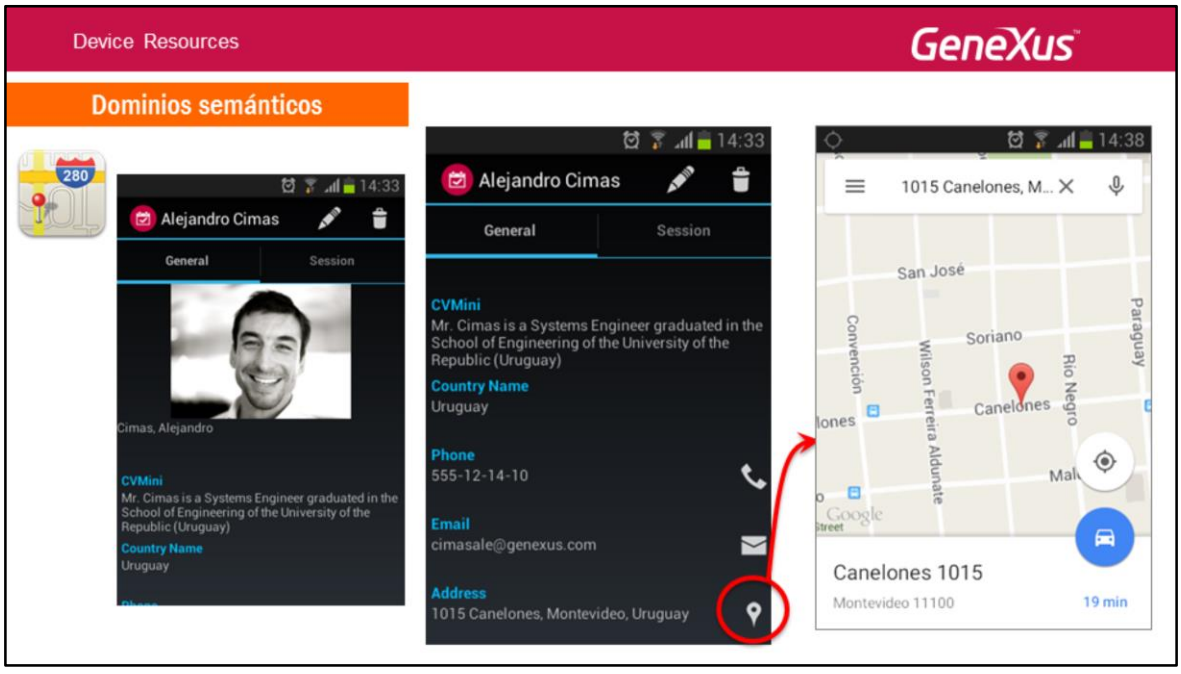


Aquí repasamos algunos ejemplos de integración a partir de la utilización de los dominios semánticos que vienen con GeneXus.

El atributo SpeakerPhone es del dominio semántico Phone, y esto hace que automáticamente se incorpore la funcionalidad que permite al usuario realizar la llamada telefónica a ese número, a través del programa nativo que viene con el dispositivo.



SpeakerEmail es del domino Email. De esta manera, el usuario puede enviarle un email al Speaker simplemente haciendo tap sobre el campo. Se abrirá el programa de correo electrónico instalado en el dispositivo.

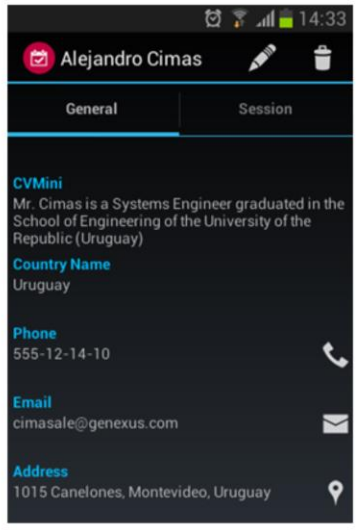


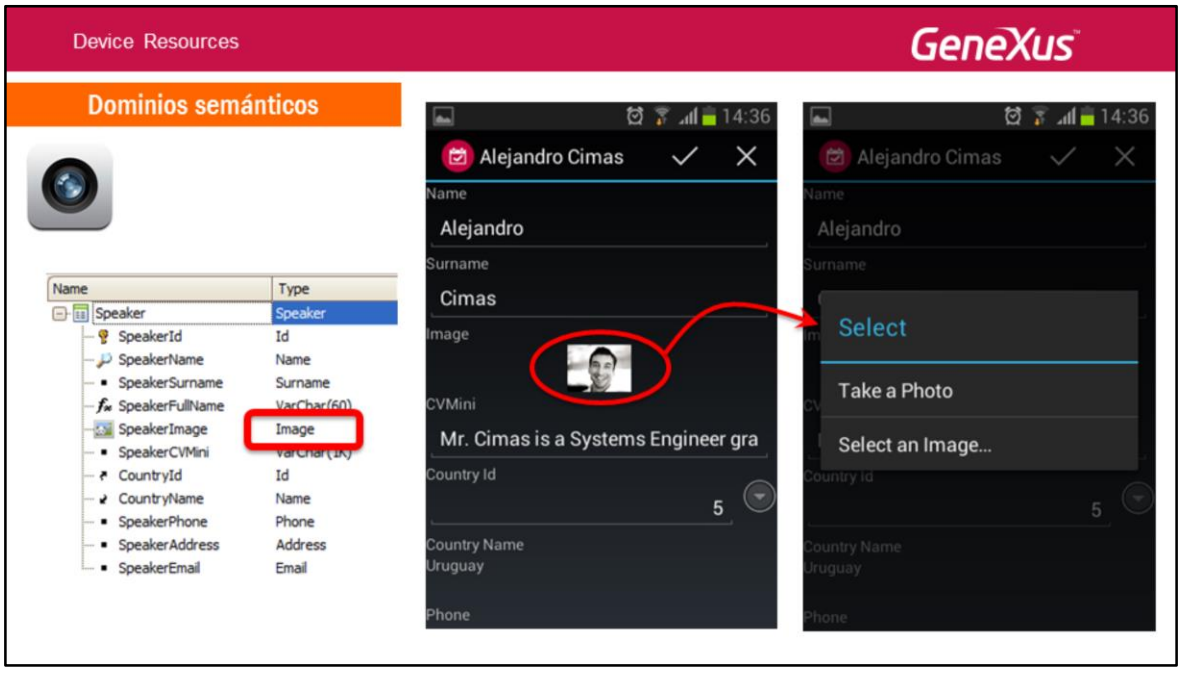
SpeakerAddress es del dominio semántico Address, lo que permitirá que el usuario pueda ver la dirección en un mapa, siempre que el dispositivo tenga instalada una aplicación de mapas.

GeneXus

Dominios semánticos

Name	Type
Speaker	Speaker
SpeakerId	Id
SpeakerName	Name
SpeakerSurname	Surname
SpeakerFullName	VarChar(60)
SpeakerImage	Image
SpeakerCVMini	VarChar(1K)
CountryId	Id
CountryName	Name
SpeakerPhone	Phone
SpeakerAddress	Address
SpeakerEmail	Email

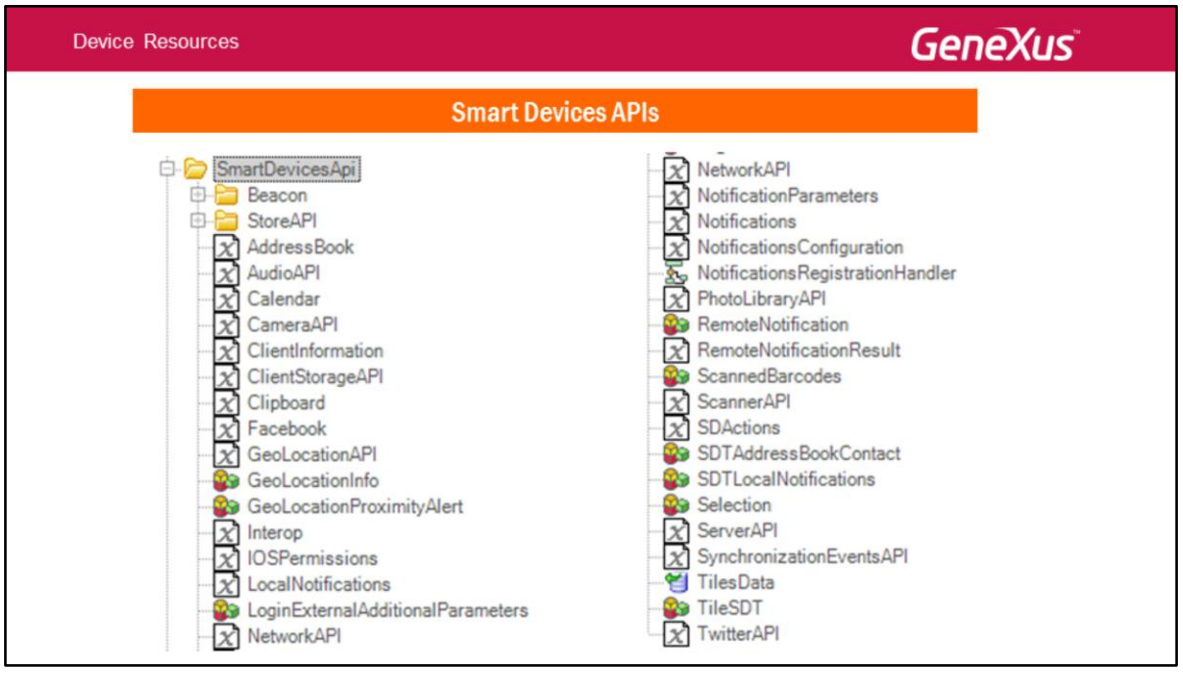




Por otro lado, SpeakerImage es del tipo de datos Image. Esto hará que cuando el usuario está en la pantalla de Edit, si hace tap sobre el campo se le ofrecerá tomar una foto y subirla como foto del Speaker, o elegir una de la galería de imágenes del dispositivo. De elegir la primera opción, estaremos interactuando con la cámara fotográfica del dispositivo.

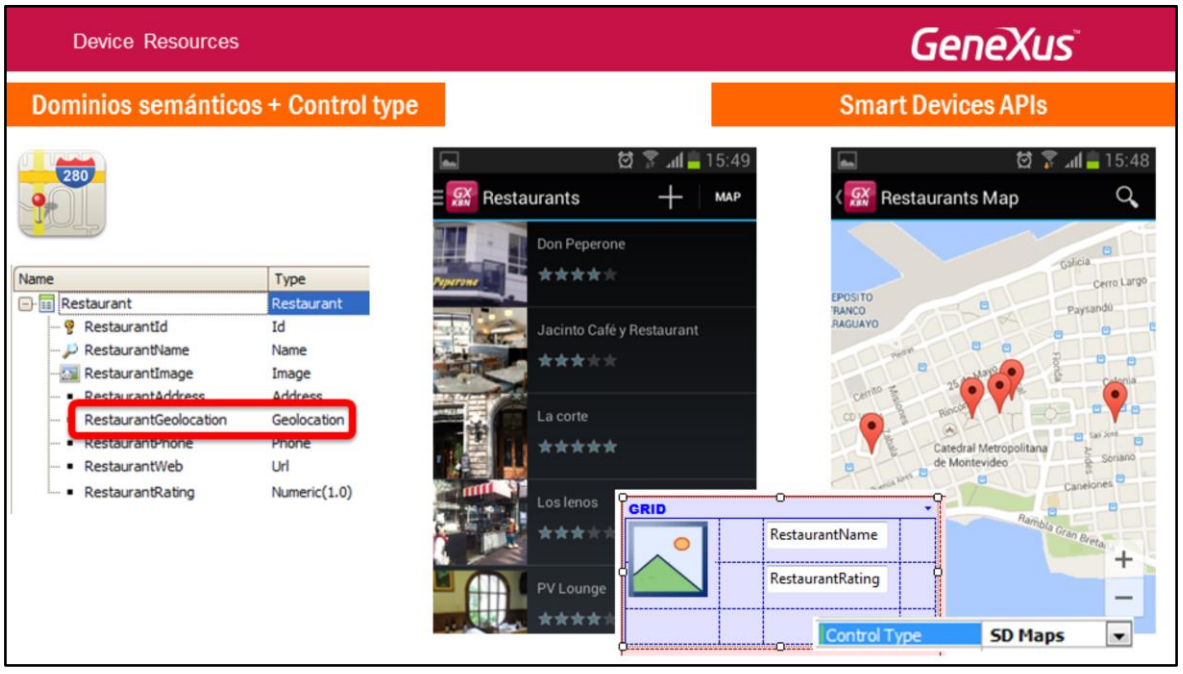
También puede utilizarse la cámara en conjunción con un programa instalado en el dispositivo de lectura de código de barras o QR.
Por ejemplo, podemos ver una aplicación para un sistema de supermercados donde se ofrece escanear el código de barras o QR code de un producto, para poder mostrar toda su información.

Una alternativa para ello es, teniendo una variable Varchar, &code, cambiarle su Control Type en el layout por **SD Scanner**. Con ello, cuando la variable no es ReadOnly se agregará un botón a su lado, Scan, que abrirá el programa lector de códigos de barras o QR.
La otra alternativa es utilizar la API ScannerAPI, que ofrece dos métodos: uno para escanear solamente un código, y el otro para escanear una serie de códigos.
En el ejemplo, para el search del producto se ha colocado una imagen en el layout, que se asocia a un evento de usuario ScanAndSearch, que utiliza la Api para escanear el código y devolver el varchar correspondiente en una variable que luego se pasa como parámetro al Panel for Smart Devices que muestra la información de ese producto.



Aquí vemos el contenido del folder SmartDevicesApi, correspondiente a GeneXus X Evolution 3, upgrade 2. Podemos ver todos los objetos externos que implementan las diferentes Apis que podrán utilizarse para interoperar. Este folder irá engordando con el tiempo, incorporando nuevas funcionalidades.

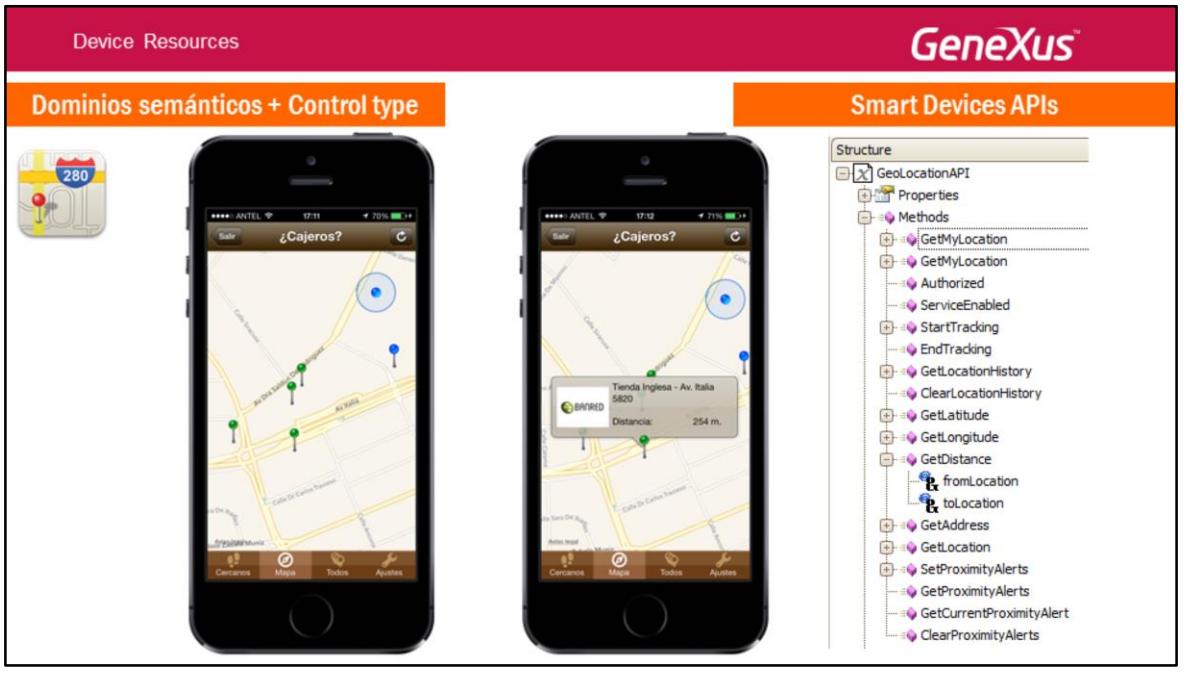
Podemos ver apis para integrar nuestra aplicación con la libreta de direcciones del dispositivo, con el calendario, con la aplicación nativa de Facebook y Twitter, con la cámara, con los mapas, para obtener información del cliente, del server o de la network, para enviar notificaciones, etcétera. Veremos algunas.



En la transacción Restaurant tenemos un atributo del dominio semántico **Geolocation**, que almacena las coordenadas geográficas **latitud/longitud**.
Teniendo un atributo de este tipo, podemos modificar la forma de mostrar la información del List correspondiente para que, en vez de mostrar cada Restaurant de la manera estándar, como una línea de información en un listado vertical, se lo muestre como su ubicación en un mapa, mediante un pin.

Para conseguirlo alcanza con modificar el Control Type del Grid, para que pase a tener el valor **SD Maps**.

Si desea ver una pequeña demo con algunos tipos de control, busque el video http://training.genexus.com/smart-devices/curso-para-aplicaciones-moviles-con-genexus-evolution-3?es#controles-tipos-de-control-en-el-diseño.



Aquí vemos un ejemplo similar, pero un poco más complejo. Necesitamos consumir un servicio externo que devuelva los datos de los cajeros automáticos para poder mostrar en un mapa únicamente los que están cercanos a la ubicación actual del dispositivo.

Para ello tendremos un grid con la información de los cajeros (entre esa información, su geolocation), al que habremos configurado el Controy Type como SD Maps. Pero debemos filtrar la información del grid, para que solamente muestre aquellos ítems para los cuales la distancia entre su localización y la del dispositivo sea menor a 500 metros, por ejemplo.

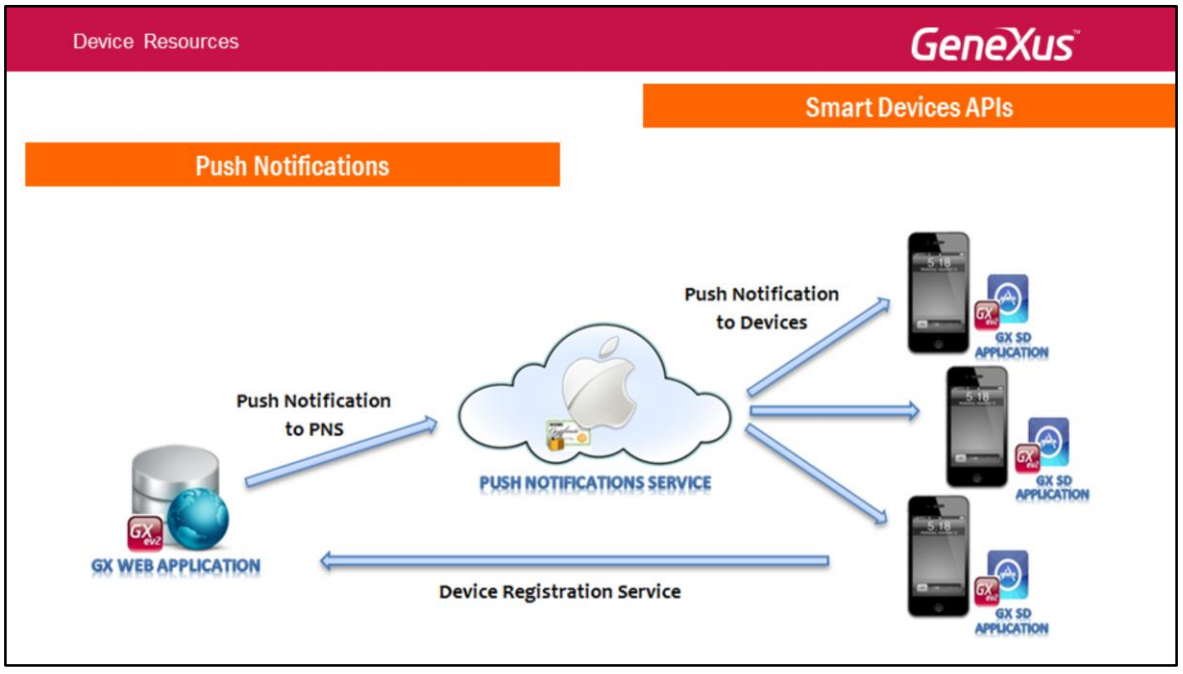
Utilizaremos para realizar ese filtro la GeoLocationAPI. Observemos que entre sus métodos tenemos uno que es GetMyLocation que devuelve la geolocalización actual del dispositivo, utilizando el GPS. Y otro que es GetDistance, que recibe dos parámetros geolocation.

Obsérvese que entre los métodos de la api también se ofrece la posibilidad de registrar un tracking de los lugares por los que va pasando el dispositivo, configurar alertas de proximidad, etc. Una funcionalidad muy útil es poder convertir un dato geolocation a uno address y viceversa.

Aquí vemos el ejemplo del panel que utilizábamos para mostrar los tweets del hashtag que se definió. Se ofrece una acción, como vemos, para poder ingresar un tweet. La forma de programarla es utilizando el método Tweet de la TwitterAPI.

Veamos como último ejemplo el caso de una aplicación que desea enviar notificaciones a todos los dispositivos registrados.

Las push notifications permiten recibir alertas en Smart Devices desde nuestros servidores remotos (aplicaciones web) incluso cuando las aplicaciones móviles no están corriendo.



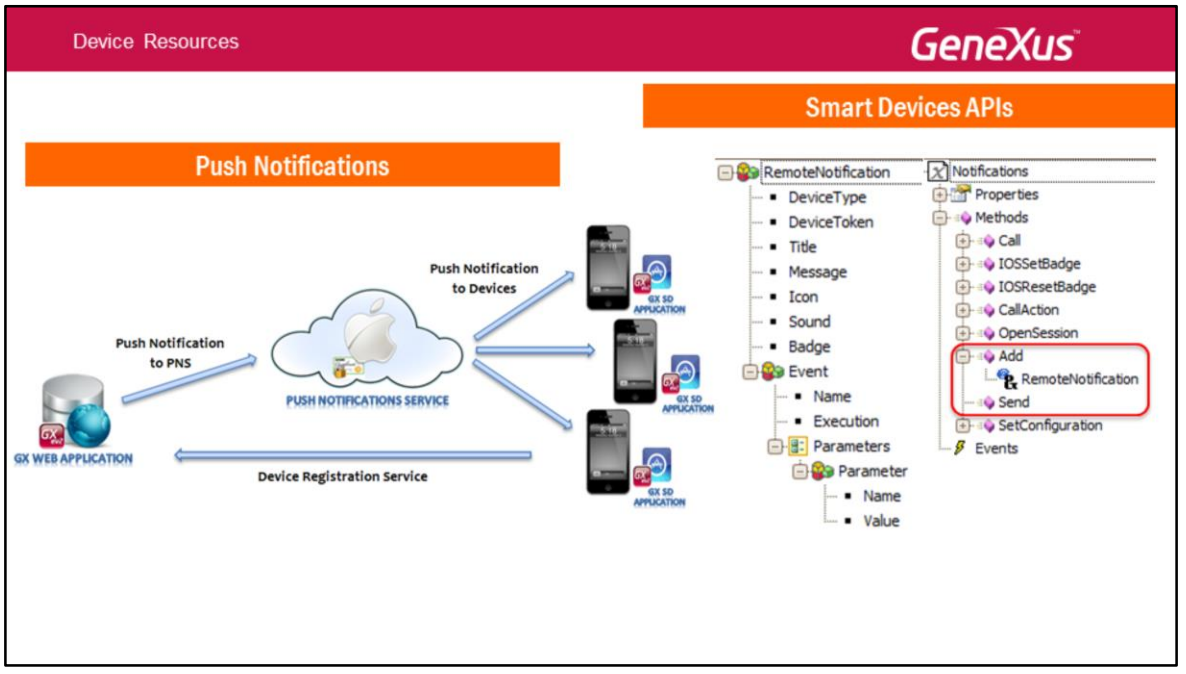
Las notificaciones se envían desde el servidor web a todos los dispositivos que se han registrado previamente en la base de datos de la aplicación.

Por lo que tenemos dos componentes:

- La aplicación SD, el client-side que será el receptor del servicio de Push notifications.
- La aplicación Web, el server-side que será quien envíe las Push notifications.

Cada proveedor de Smart Devices tiene su propia tecnología para implementar este mecanismo. Así, para:

- Apple (iOS): Apple Push Notifications Service (APNS)
- Android: Google Cloud Messaging (GCM)
- Blackberry: BlackBerry Push Service
- Windows: Windows Push Notification Services (WNS)



Para implementar las push notifications, primero que nada hay que habilitar el Push Notification Service de la plataforma, y obtener el Project number y una clave (certificados para poder utilizar el servicio).
Se configura en True la propiedad **Enable Notifications** del objeto main de la aplicación SD (por ejemplo, el dashboard), se ingresa la información de las credenciales obtenidas del PNS de las plataformas para las que estemos generando.
A partir de aquí, toda vez que se ejecute la aplicación, se ejecutará el Devices Registration Service para registrar y almacenar la información del dispositivo para ser utilizada en el futuro para enviar mensajes al dispositivo (http://wiki.genexus.com/commwiki/servlet/hwikibypageid?18149).
¿Cómo se envían notificaciones a los dispositivos registrados?

Para ello contamos con el SDT RemoteNotification que contendrá toda la información de cada mensaje a ser enviado, y la api Notifications que con su método Add permite armar la colección de RemoteNotifications a ser enviada, para luego, con el método Send, enviarla a los PNSs correspondientes, que son quienes se encargarán de hacer llegar los mensajes a los dispositivos.

Roadmap

GeneXus

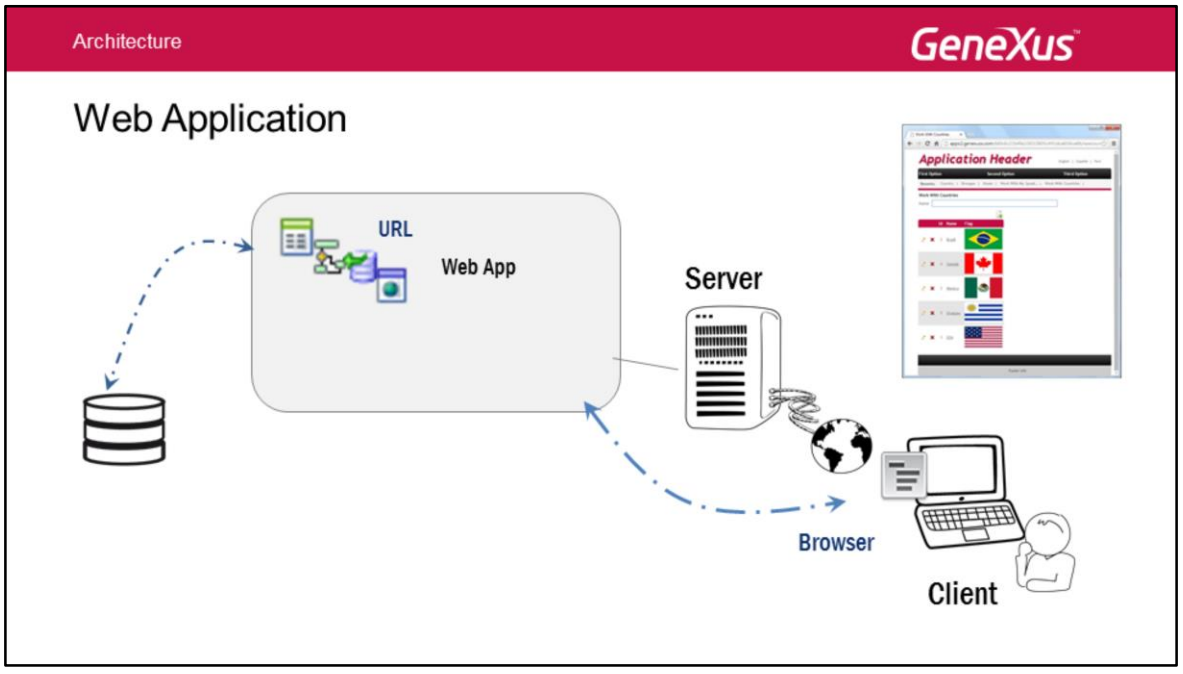
Características y plataformas

Interfaz y modelo conceptual

Integración con recursos del dispositivo

GeneXus

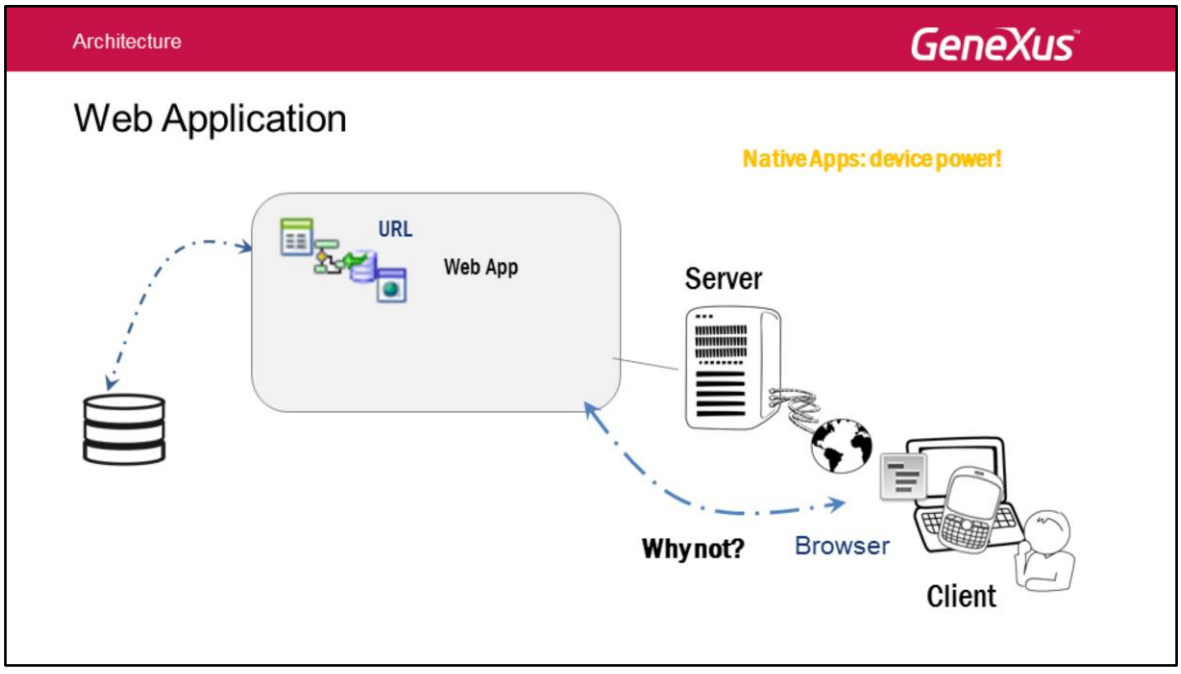
Architecture of online mobile applications



Para pensar la arquitectura subyacente en las soluciones para Smart Devices con GeneXus, partamos de lo conocido: las aplicaciones Web y hagamos la comparación que nos permitirá comprender mejor.

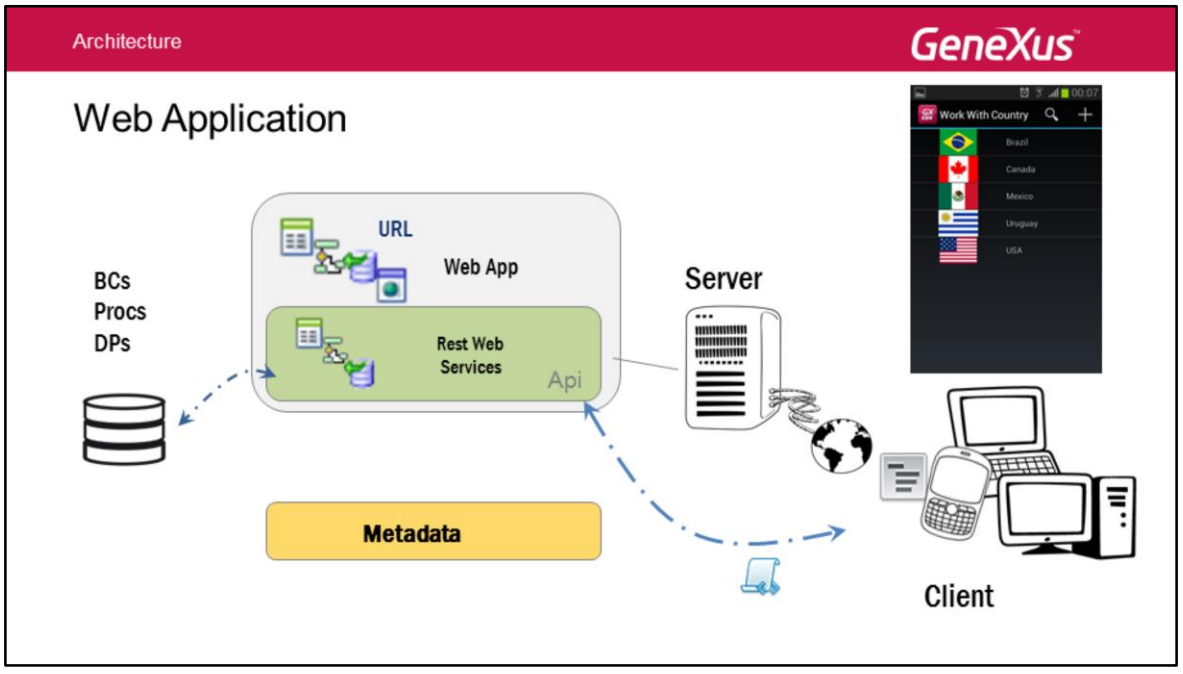
Tenemos por un lado un Servidor y por otro un Cliente. En el servidor tenemos la aplicación web y en el cliente un Browser.

Ejecutamos la aplicación Web a partir de una URL que ejecuta, por ejemplo el WorkWithCountries. Este objeto consulta la base de datos y devuelve la información al cliente, para que el Browser arme el layout que presentará al usuario (HTML) como respuesta a su pedido.



Si queremos la aplicación ejecutándose en un Smart Device, ¿por qué implementar una solución particular en vez de usar la web, a través del navegador del dispositivo?

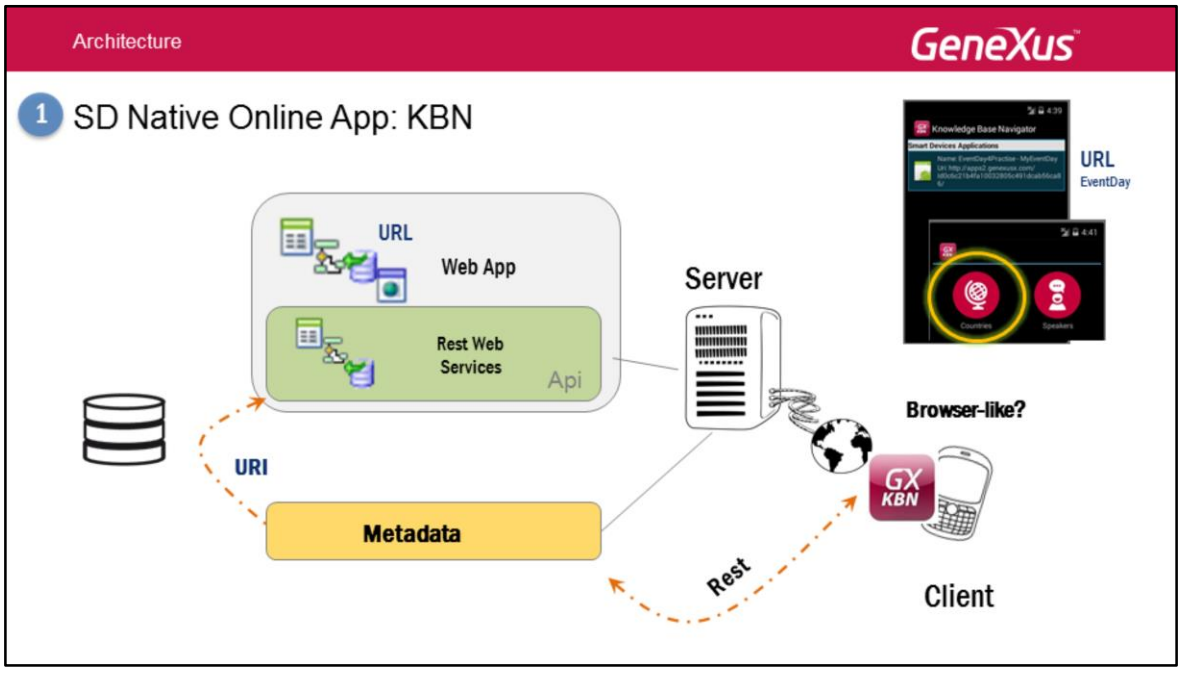
Es que queremos que la aplicación interactúe con las funcionalidades propias del dispositivo, como la agenda de contactos, el calendario, etc., y que tenga un look & feel similar al resto de las aplicaciones nativas.



Seguimos exclusivamente con la aplicación web. Generalizando: si deseamos que algunos de los objetos de la aplicación que procesan y devuelven datos estructurados (transacciones como business components, procedimientos y data providers), puedan ser consumidos por otros programas (no necesariamente implementados con GeneXus) a través de internet (tanto desde una notebook o pc, como de un smart device), una buena alternativa es exponerlos como **Rest Web Services** (serán, así, apis de la aplicación, conformando una **capa de servicios**). Con ello nos encontramos dentro de una arquitectura de diseño Rest, que piensa en esos programas como **recursos**.

De esta manera cualquier programa que acceda a internet, conociendo la URL de cualquiera de estos **web services**, podrá invocarlos a través del protocolo HTTP (con los métodos GET, POST, PUT, DELETE según corresponda). El **servicio** se ejecutará en el Server, accediendo a la base de datos (por ejemplo un data provider cuya salida sea una colección de bandera y nombre de los países) y devolviendo como response al Cliente la representación del recurso (con el formato JSON). El cliente deberá saber decodificar ese JSON, para hacer con su información lo que necesite.

Así, si el cliente se ejecuta en un dispositivo inteligente, y éste accede a una metadata (probablemente en el propio servidor) que contiene la info para armar la interfaz del work with (entre otras cosas), sabrá cómo armar el layout del Work With Country, a partir del Json recibido con los datos, mostrando la lista en el dispositivo. Por allí vendrá la solución que estamos buscando.

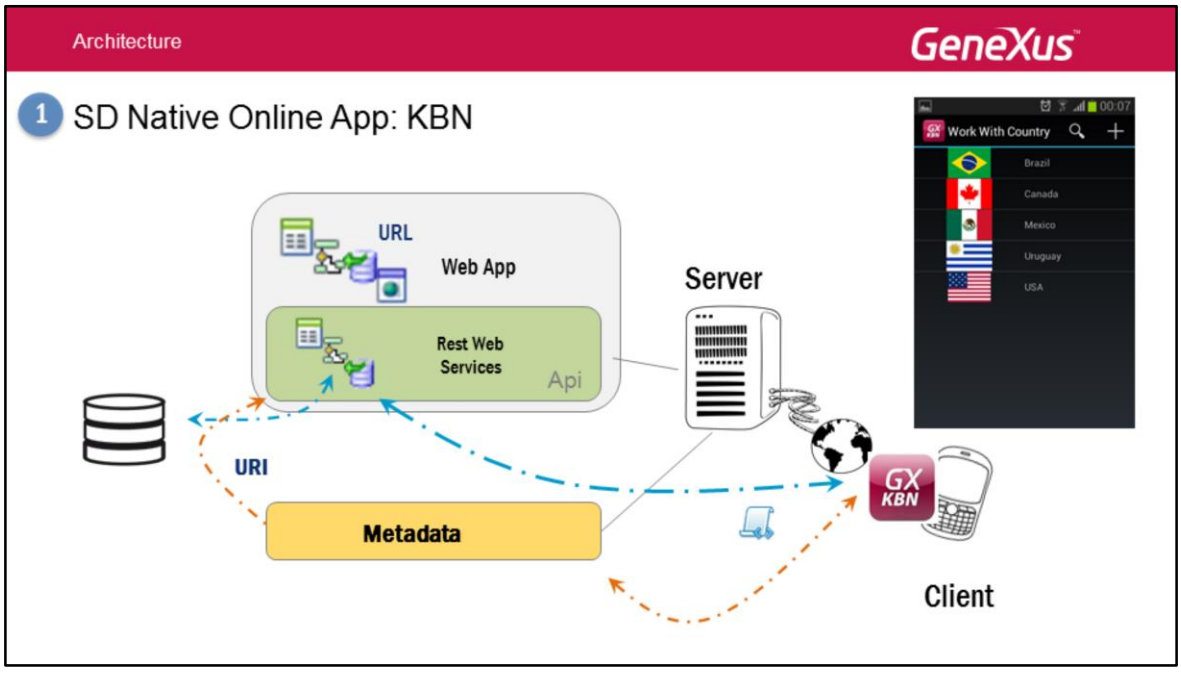


La **metadata** de la aplicación para Smart Devices contendrá toda la info de la aplicación (qué dashboards, work with for smart devices y panels implementa, y las URIs de los web services para obtener los datos necesarios de la base de datos) para poder armar la interfaz de la aplicación en el dispositivo y responder a las acciones que se ejectuan. Estará en el servidor web.
Tenemos dos opciones para prototipar: ejecutar una especie de navegador especial creado por Artech, el KBN, o instalar la aplicación compilada. Estudiaremos ambas.

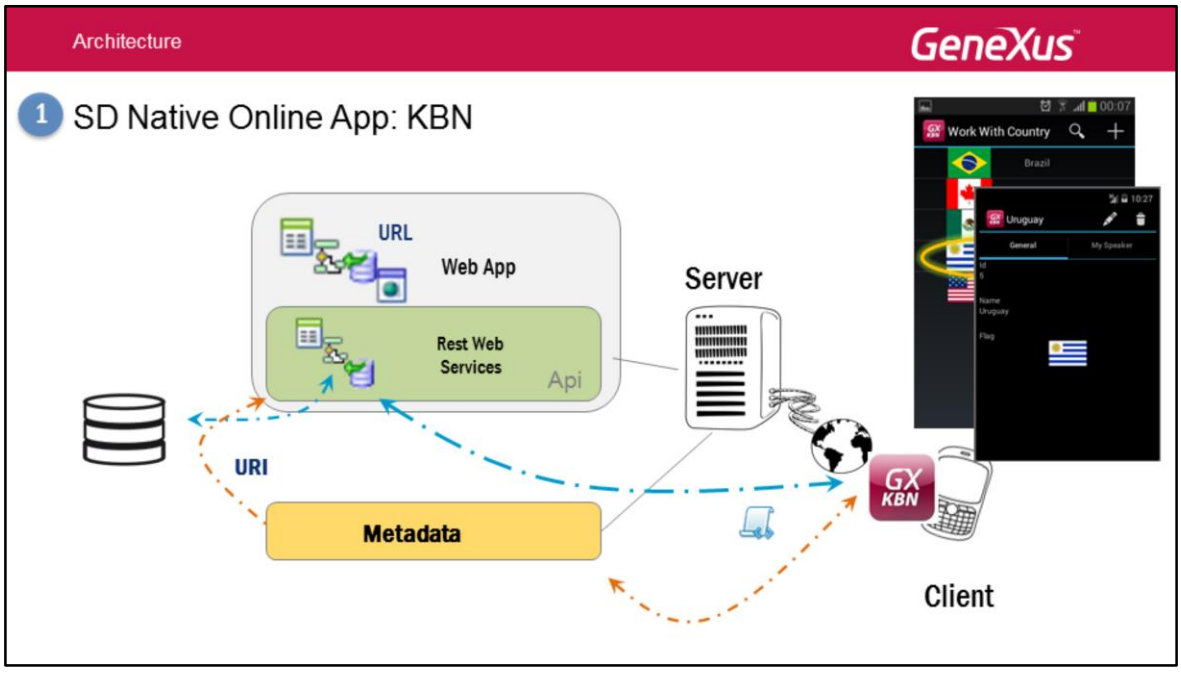
Opción 1: utilizar el Knowledge Base Navigator (KBN)

- Es una aplicación nativa (compilada en el lenguaje de la plataforma) creada por Artech.
- Permite navegar a través de las aplicaciones para Smart Devices creadas con GeneXus. Es como un Browser, que eligiendo una URL (correspondiente a un objeto main de la aplicación, consignado en la metadata), permite trabajar con las entidades y relaciones que conforman la parte de la aplicación GeneXus para Smart Devices que depende de ese main.
- Es un **intérprete** liviano, que tiene la lógica para leer la metadata de la URL correspondiente, así como las imágenes de la aplicación y decodificarla, invocando, de ser necesario, a los web services rest que requiera para obtener las respuestas con los datos, para armar la interfaz correspondiente en el dispositivo, que es la que visualiza el usuario.

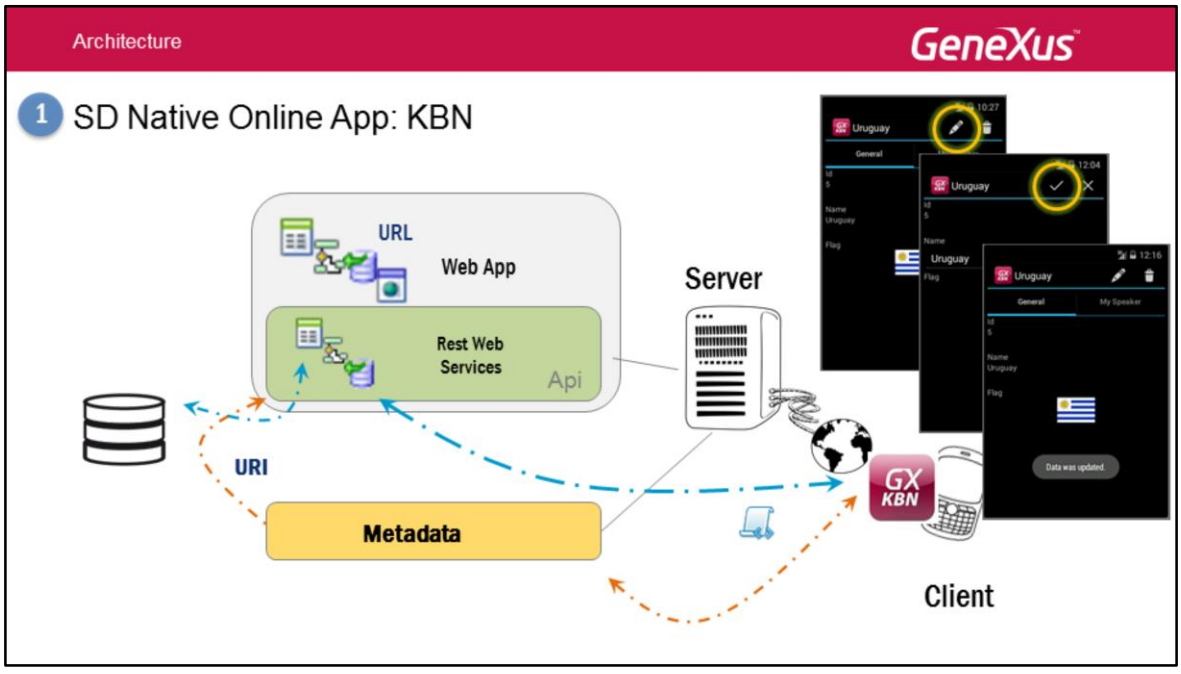
Por ejemplo, lee en la metadata que debe comenzar por el **dashboard** EventDay, que tiene tales imágenes, y tal interfaz (layout) y tales opciones. Arma la interfaz y la despliega en el dispositivo. Cuando el cliente hace tap sobre la opción, de la metadata obtiene la URI para ejecutar el recurso: en el ejemplo, Countries (un data provider que devuelve la lista de países).



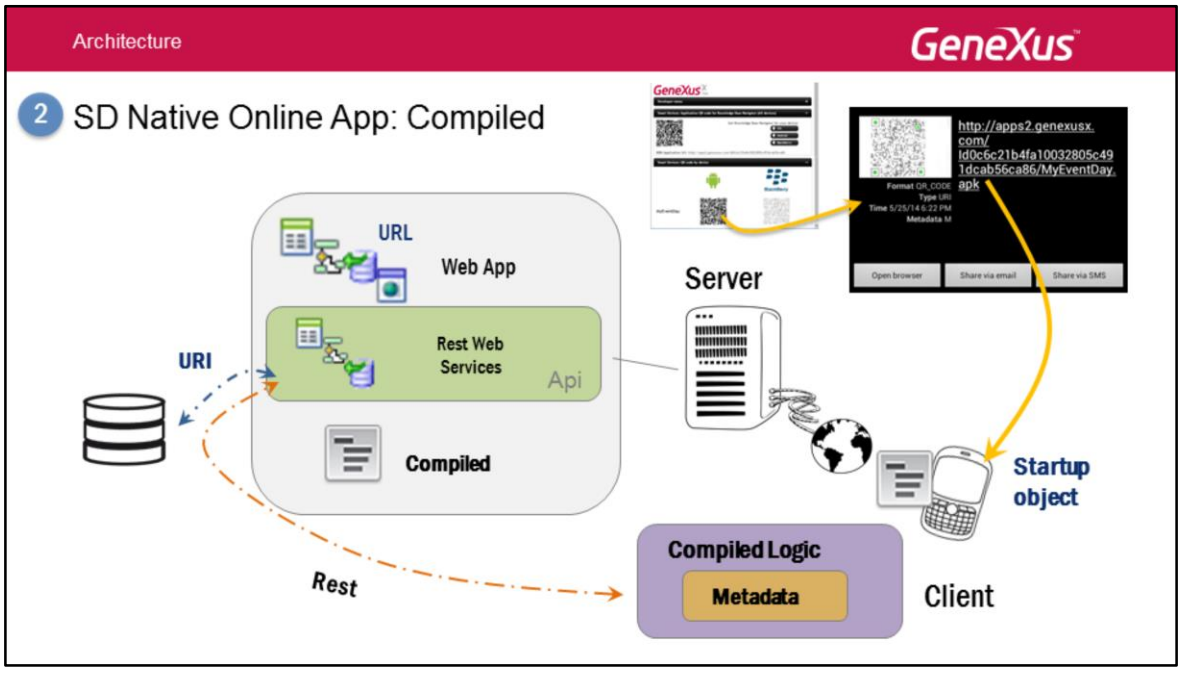
Por lo que el KBN lo ejecuta vía Http (rest), con lo cual el Data Provider accede a la base de datos para obtener la colección de países en un JSON, y este Json se le devuelve como respuesta al KBN, que habiendo accedido además a la metadata, tiene todo lo que necesita para armar la pantalla que se muestra al usuario en el dispositivo.



Análogamente, si se hace tap sobre un elemento de la lista se llamará al data provider que devuelve la información del país, para armar la pantalla del View.



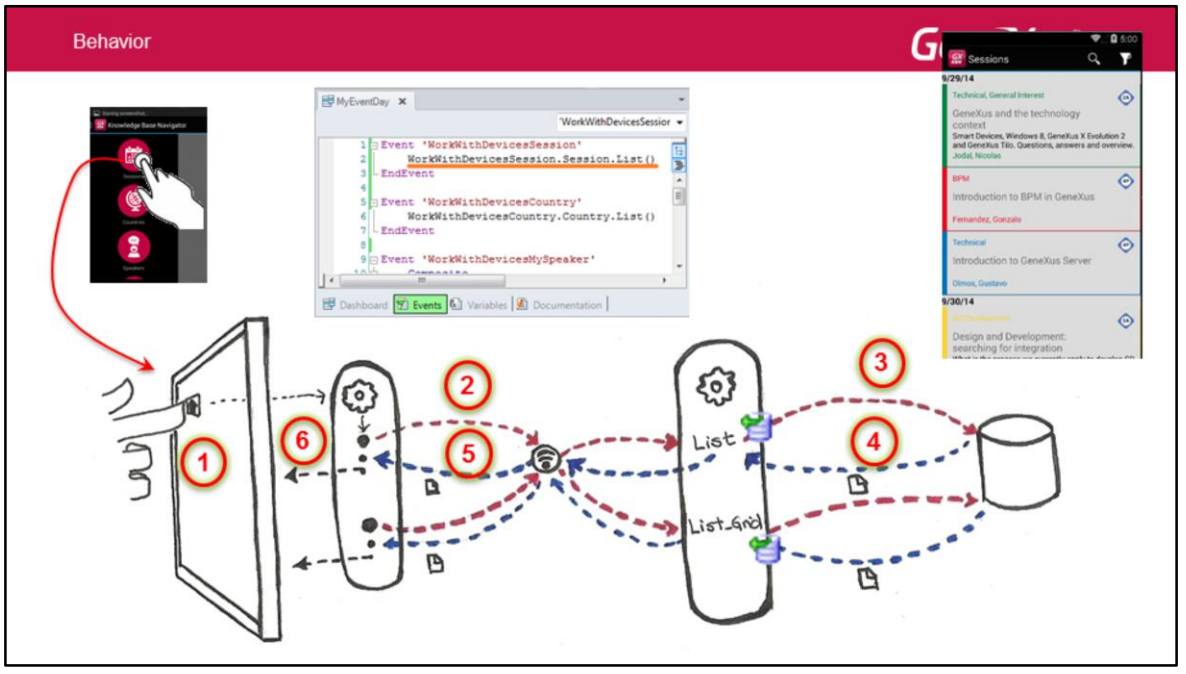
Luego, si se hace un update o un delete (o un insert desde el list), se dibuja la pantalla de Edit, y al intentar grabar, el servicio rest invocado será el Business Component, que intentará realizar la operación correspondiente sobre la base de datos, y devolverá al llamador el resultado de la operación (si falló, los mensajes de error ocurridos, para que se le muestren al usuario en la pantalla del dispositivo; si fue exitoso, un mensaje indicándolo).



Opción 2: Compilar la aplicación en el lenguaje del dispositivo, e instalarla en el mismo.

Cada plataforma de Smart Devices tiene su propio lenguaje y por tanto su propia extensión para el archivo compilado. Por ejemplo, para Android es .apk. Este archivo debe descargarse e instalarse en el dispositivo, y ya no se necesitará del intérprete KBN, dado que es como un KBN customizado (que ya tiene configurada la URL de la aplicación), y encapsulará toda la metadata y las imágenes.
Por tanto sólo deberá accederse al servidor para ejecutar los Rest Web Services que devolverán los datos actuales, procesados.

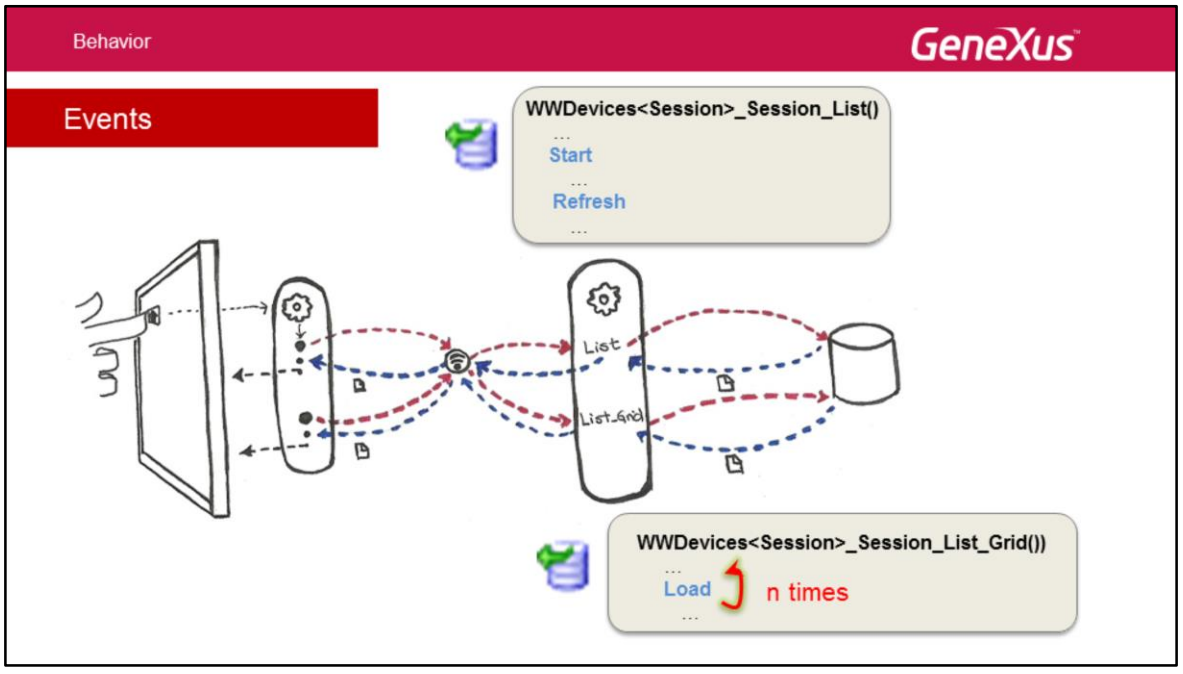
Para compilar la aplicación para Android, alcanza con indicar en las propiedades del Environment (e.g. Ruby Environment) el "**Startup object**" (por ejemplo, el dashboard EventDay). En este caso al hacer F5 no se generará la aplicación web, sino únicamente el compilado, apk, que se subirá al servidor. Por ello no se abrirá el Developer Menu web. De estar prototipando en la nube, podrá acceder al QR Code que encapsula la URL donde se encuentra el compilado para descargar en el dispositivo, yendo en GeneXus a View / Show QR Codes.



¿Qué sucede cuando desde el dashboard invocamos al List de Sessions? :

1. Usuario hace tap en el ícono del dashboard correspondiente a la lista de sessions, que ejecuta el Work With en el dispositivo.
2. App en el dispositivo (Work with) hace un call externo al servicio Rest (Data provider correspondiente a la carga de la parte fija del WW: "WorkWithDevices<Transaction>_Level_List()")
3. El DP se ejecuta (dentro de su lógica interna se ejecutan los eventos **Start** y **Refesh**). Se accede a la base de datos, si se requiere.
4. Si fue requerida, la BD encuentra los registros y los devuelve al DP.
5. El servicio Rest (DP) devuelve en su response la info a la parte de la app que corre en el dispositivo (el Work With que lo llamó).
6. La app en el dispositivo busca las imágenes de la app y con la response recibida, dibuja la UI (user interface) correspondiente a la **parte fija** del Work With.

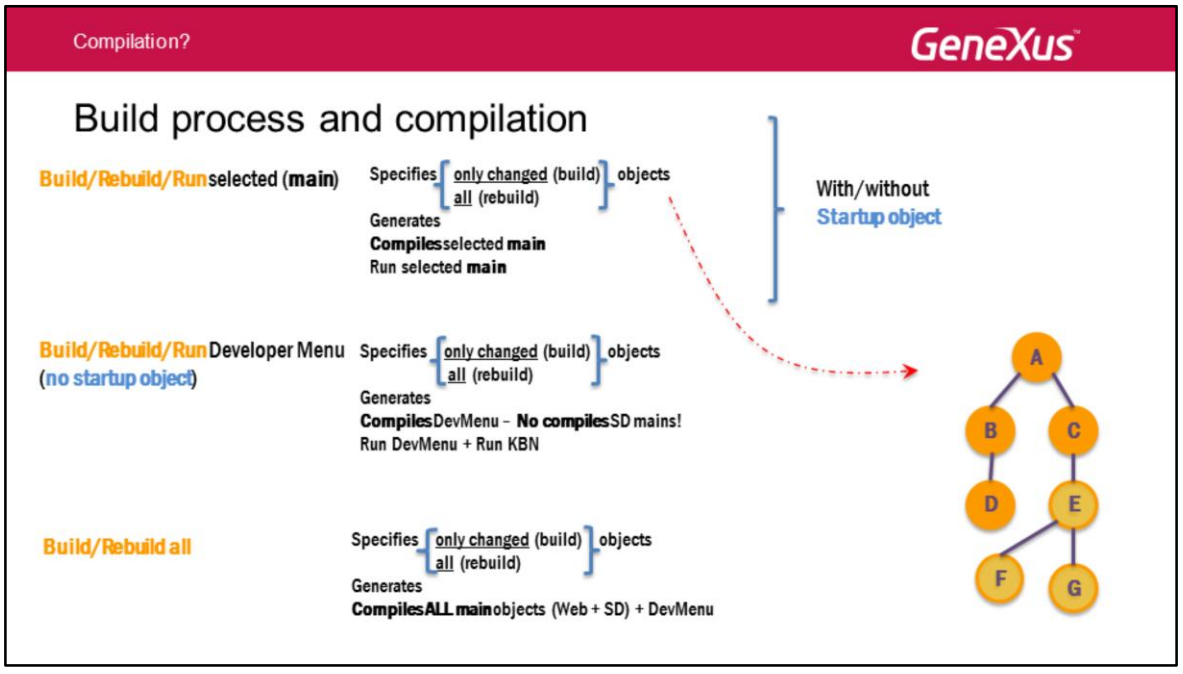
7. Se repiten los pasos 2 al 6, pero ahora llamando al Data Provider que devuelve los datos del grid: WorkWithDevices<Transaction>_List_Grid(). Internamente este DP ejecuta el evento Load por cada línea del grid a ser cargada. Una vez la app en el dispositivo recibe la response con los datos del grid, dibuja la UI correspondiente **al grid**.



En definitiva, del lado del cliente (el dispositivo), tenemos en el evento del dashboard que activamos al hacer tap sobre la imagen, una invocación al list del work with de Sessions. El código correspondiente es el que se ejecuta en el cliente, que comienza a ejecutar, entonces, el WorkWith correspondiente al List. En su lógica interna, este Work With llama al servidor para que éste le devuelva los datos a cargar en la parte fija a través de un servicio Rest, y luego le pide al servidor que ejecute el otro servicio rest, el que devuelve los datos del grid. Con los datos devueltos en dos responses, arma la pantalla (por un lado la parte fija, y por otro el grid).

Nota: estamos suponiendo que no hay caching, para simplificar la explicación. Tampoco estamos poniendo el foco en el paginado del grid, que también se realiza (el DP que ejecuta el Load, lo hace paginando, es decir: sólo devuelve x registros por vez).

El evento Start se dispara solamente la primera vez. No se vuelve a ejecutar a menos que se salga del panel y se vuelva a entrar.
Sobre todo esto puede profundizar en http://training.genexus.com/smart-devices/curso-para-aplicaciones-moviles-con-genexus-evolution-3?es#eventos-en-aplicaciones-móviles.



Recordemos que el F5 cuando no hay Startup Object equivale al Run del Developer Menu. Cuando hay Startup Object, equivale al Run de ese objeto.

Nota: Si vamos a Tools/Options/Build y cambiamos la propiedad **Call tree for Build** de su valor por defecto "Stop on Main Objects" por "Full", entonces cuando se elija hacer Build A, siendo A un objeto main, se especificarán todos los objetos que hayan cambiado del árbol de raíz A, sean o no main (si E es main, no se cortará la especificación en el árbol cuya raíz es E). Lo mismo vale para los otros builds (build all, build Developer Menu, build Startup Object).

No se ha listado la opción **Build (Run) with this only**. La idea de esta opción es especificar y generar solamente el objeto seleccionado. Si este objeto es main, entonces compilarlo y ejecutarlo. En caso contrario, compilar y ejecutar el startup object si existe, y si no existe, el Developer Menu si "this" es un objeto web o el KBN si "this" es un objeto Smart Device.
Recordar que en GeneXus X Evolution 3 existe también la opción **Run Without Building**.


GeneXus™