Desarrollo de software para dispositivos móviles

Obligatorio 2

Datamyne Mobile

Trade Profiles

2012

**Rodrigo Demicheli, Nro. est. 103036**

**Mauricio Carballo, Nro. est. 169199**

Contenido

Especificación de requerimientos 4

Identificación del sistema 4

Funciones principales 4

Características de los usuarios 5

Glosario 5

Requerimientos funcionales 6

Requerimientos no funcionales 7

Alcance 8

Diagrama de componentes 9

Prototipos de interfaz grafica 10

Análisis de riesgos tecnológicos 18

Comunicación con servidor remoto a través de webservices REST 18

Incorporar una librería de graficas de terceros 18

Persistencia en una base de datos local al dispositivo 18

Análisis luego de haber desarrollado la aplicación 19

Requerimientos funcionales 19

RF1: Login del usuario 19

RF4: Búsqueda por nombre de empresa 19

RF5: Acceso a información de contacto 19

RF6: Totales mensuales 20

RF7: Totales por dimensión 20

RF8: Graficas 20

Requerimientos no funcionales 21

RNF1: Extensibilidad 21

RNF2: Plataforma 21

RNF3: Comunicación remota 22

RNF4: GPS y Google Maps API 22

RNF5: Cacheo de datos 22

RNF6: Orientación de la pantalla 22

El diseño e implementación 23

Dispositivos soportados 23

Pantallas normales: 23

Pantallas grandes: 23

Flujo de Actividades 24

Tabletas en orientación landscape 24

Celulares u orientacion portrait 25

Comunicación 26

Accediendo a un profile 27

Generando la vista de un profile 27

Base de datos 29

Aspectos a mejorar 31

Capturas de pantalla y manual de usuario 32

Conclusiones 37

# Especificación de requerimientos

## Identificación del sistema

Rodrigo y yo trabajamos en Datamyne, una empresa que se dedica a vender servicios sobre información de comercio exterior. Casualmente había surgido la idea de la empresa de desarrollar una aplicación para dispositivos móviles cuando nos anotamos a este curso, por lo tanto decidimos combinar la necesidad nuestra y de la empresa para desarrollar esta aplicación.

Nuestra idea (negociada con la empresa) es construir la aplicación de manera que cumpla con los requerimientos y exigencia del curso y también sirva como base (o más) para la aplicación a desarrollar en nuestro trabajo.

La aplicación principal va a ser launcher al que se le agregarán módulos que son quienes realmente proveen funcionalidad.

Para este curso vamos a desarrollar el módulo llamado Trade Profiles, el cual provee información de empresas de USA y detalles de importaciones de las mismas, por ejemplo de que país, en que puerto y que producto importó.

La información de comercio exterior se presenta en forma de ranking, con gráficas para facilitar su análisis.

## Funciones principales

* Búsqueda de empresas por su nombre.
* Mostrar información de contacto de empresas (dirección, teléfono, ciudad, zipcode, etc.).
* Mostrar información de las importaciones de una empresa totalizada y rankeada por dimensión por mes.
* Graficar la información de importaciones para facilitar el análisis de los usuarios.

## Características de los usuarios

En general los usuarios van a ser personas que se dediquen al comercio exterior o que quieran tener conocimiento sobre las de importaciones de las empresas de USA para realizar negocios, inversiones o analizar la competencia de sus empresas.

Se presume además que este universo de personas está en contacto frecuente con la tecnología y por lo tanto maneja los conceptos (al menos básicos) del mundo de la telefonía e internet.

## Glosario

* TEUS: Twenty Foot Equivalent Units, 1 TEU indica la capacidad de carga de un contenedor estándar de 20 pies de largo.
* Totales mensuales: se refiere al total de importaciones de una empresa de USA para el periodo de un mes, medido en TEUS.
* Dimensión: código unificado que representa cierto elemento de la realidad, por ejemplo códigos ISO de país o de puerto.
* Totales por dimensión: se refiere al total de importaciones de una empresa de USA que pertenecen a cierta dimensión, medido en TEUS.
* Consignee (Comprador): empresa que importa productos en USA.
* Shipper (Proveedor): empresa a la que un Consignee le compra mercadería.

## Requerimientos funcionales

**RF1: Login del usuario**

*Descripción:* para acceder a los datos el usuario debe estar logueado.

*Prioridad:* Alta

**RF2: Guardar credenciales del usuario**

*Descripción:* se recordaran las credenciales del usuario para no preguntárselas la siguiente vez que use la aplicación.

*Prioridad:* Baja

**RF3: Niveles de acceso a datos por permisos**

*Descripción:* los datos mostrados dependen de los permisos del usuario logueado.

*Prioridad:* Baja

**RF4: Búsqueda por nombre de empresa**

*Descripción:* se debe poder buscar empresas por su nombre.

*Prioridad:* Alta

**RF5: Acceso a información de contacto**

*Descripción:* se debe mostrar información de contacto de empresas de USA.

*Prioridad:* Alta

**RF6: Totales mensuales**

*Descripción:* se debe mostrar los totales mensuales de la empresa para el período de un año.

*Prioridad:* Alta

**RF7: Totales por dimensión**

*Descripción:* se deben mostrar totales para las dimensiones País, Producto, Supplier y Puerto período de un año.

*Prioridad:* Alta

**RF8: Graficas**

*Descripción:* los totales mensuales y por dimensión deben ser graficados. En las gráficas de totales por dimensión, si el usuario hace clic en una de las dimensiones, se abrirá una nueva grafica (de barras) con la evolución anual para el ítem elegido.

*Prioridad:* Alta

**RF9: Funcionalidad del estilo Around Me**

*Descripción:* a partir de una búsqueda se deben mostrar las empresas encontradas en un mapa, junto con la ubicación actual del usuario.

*Prioridad:* Baja

**RF10: Notificaciones push**

*Descripción:* se debe poder recibir notificaciones de actualizaciones de los datos.

*Prioridad:* Baja

## Requerimientos no funcionales

**RNF1: Extensibilidad**

*Descripción:* La aplicación principal debe ser extensible por plugines o sub-aplicaciones.

*Prioridad:* Alta

**RNF2: Plataforma**

*Descripción:* La plataforma Android debe ser 3.x o superior.

*Prioridad:* Alta

**RNF3: Comunicación remota**

*Descripción:* La obtención de los datos debe ser a través de un webservice (REST) usando SSL/HTTP.

*Prioridad:* Alta

**RNF4: Motor de base de datos**

*Descripción:* se debe usar el motor SQLite.

*Prioridad:* Alta

**RNF4: GPS y Google Maps API**

*Descripción:* se debe usar el GPS del dispositivo y las API de Google Maps para Around Me

*Prioridad:* Baja

**RNF5: Cacheo de datos**

*Descripción:* cachear los datos obtenidos remotamente para uso off-line.

*Prioridad:* Baja

**RNF6: Orientación de la pantalla**

*Descripción:* la aplicación debe funcionar en la orientación vertical y horizontal

*Prioridad:* Alta

# Alcance

Según lo hablado con los profesores, estos requerimientos son los que se van a desarrollar para este curso:

Funcionales:

* **RF1**
* **RF4**
* **RF5**
* **RF6**
* **RF7**
* **RF8**

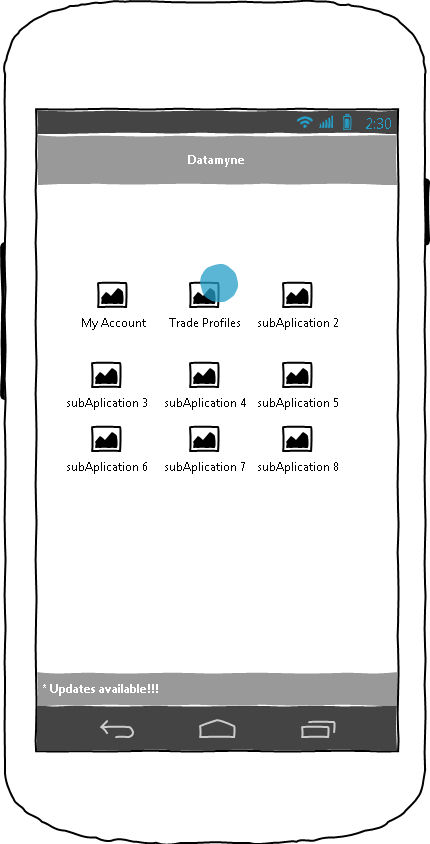
No Funcionales:

* **RNF1**
* **RNF2**
* **RNF3**
* **RNF4**
* **RNF6**

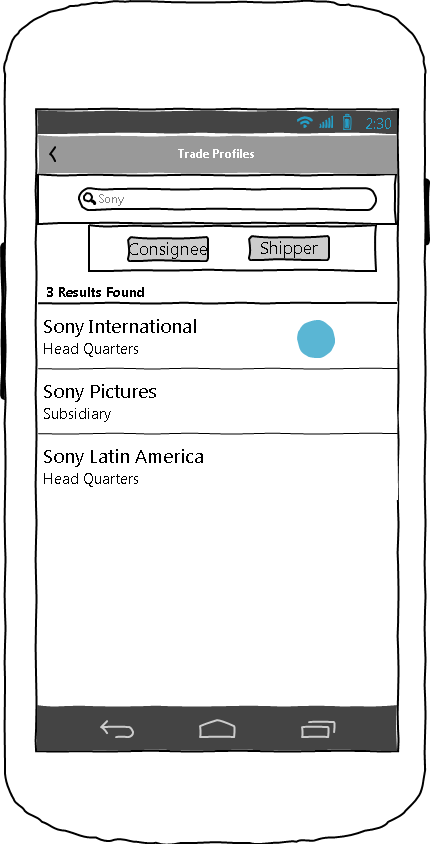
# Diagrama de componentes

* **Launcher:** es la aplicación “cascarón” que provee acceso a las extensiones y maneja el Login del usuario.
* **Base de datos:** configuración de la aplicación principal (por ejemplo credenciales de Login).
* **Trade Profiles:** modulo principal de este desarrollo, es la que va a implementar la mayoría de los requerimientos funcionales y no funcionales.
* **Configuración:** configuración de la aplicación Trade Profiles (conexión a base de datos, recursos externos, etc.).
* **Cache:** datos guardados para el uso offline de la aplicación.
* **Graficas:** librería de terceros para el manejo de gráficas.
* **Cliente REST:** es el encargado de comunicarse con el servidor de Trade Profiles para traer la información.
* **Servidor Trade Profiles:** expone servicios para ser consumidos por el Cliente REST, es quien tiene los datos propiamente dichos.

# Prototipos de interfaz grafica

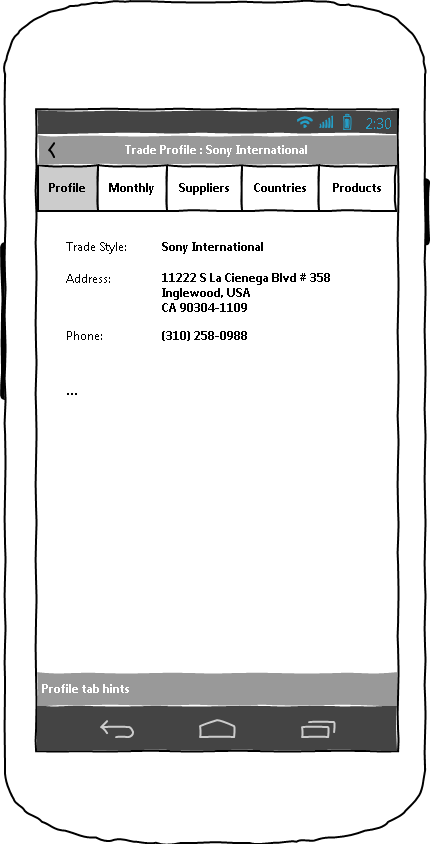
Al iniciar la aplicación se verá este dashboard:

Luego el usuario selecciona el icono de *Trade Profiles* y realiza la búsqueda de empresas por el texto “Sony” (se muestran 3 resultados de la búsqueda como ejemplo):

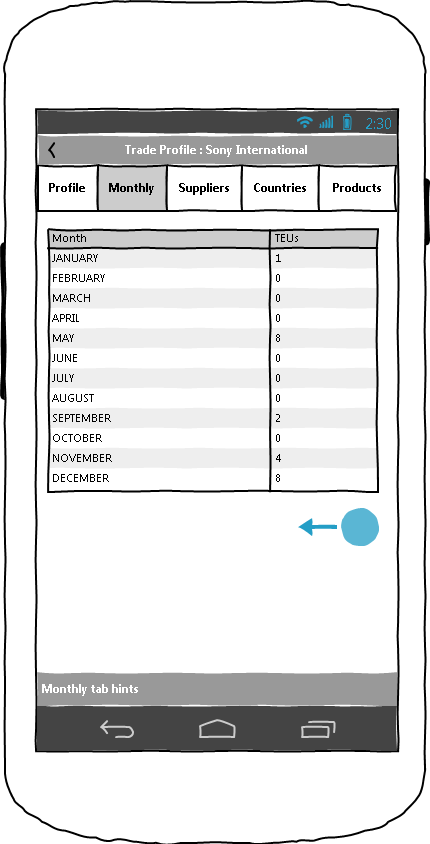


Luego el usuario selecciona uno de los resultados, en el ejemplo el primero de la ventana anterior.

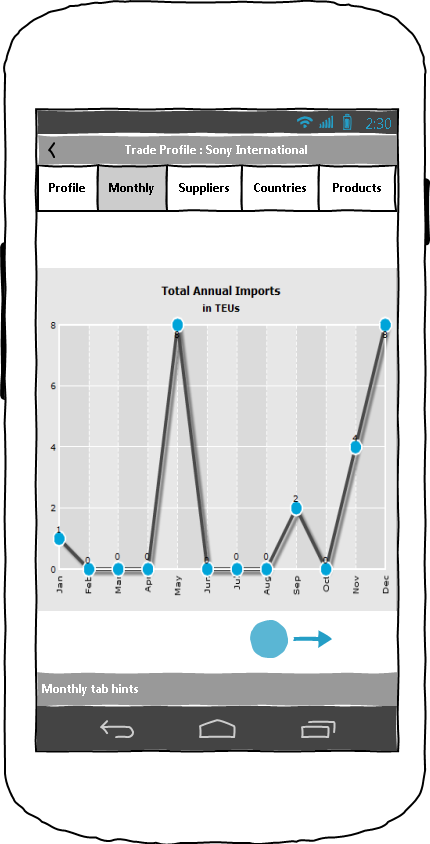
Se muestra la información de contacto de la empresa seleccionada:



Luego el usuario hace clic sobre la opción *Monthly,* se muestran los totales mensuales en TEUs de las importaciones de la empresa:

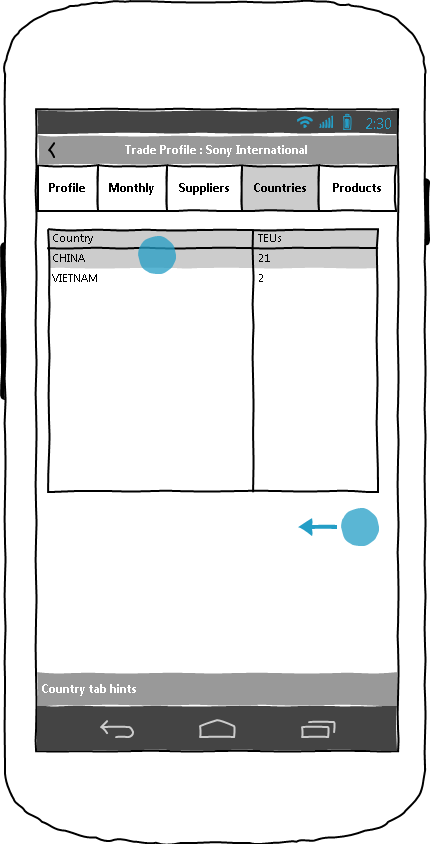


Deslizando el dedo hacia la izquierda en la pantalla anterior, se mostrara la gráfica:

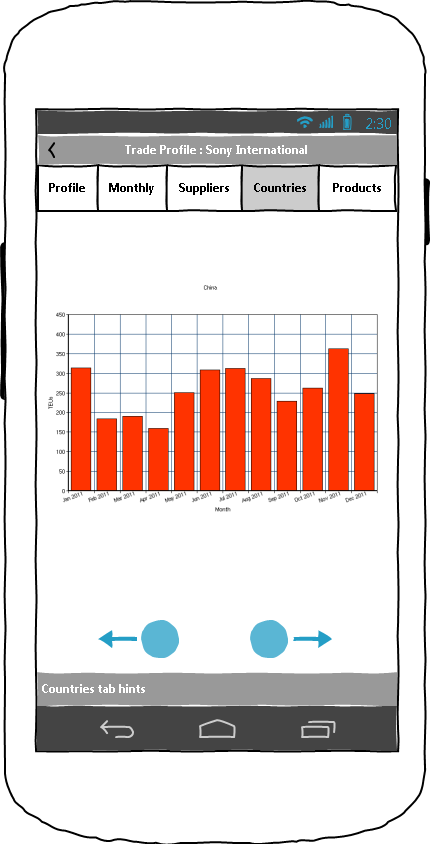


Deslizando el dedo a la derecha se vuelve a la pantalla anterior, ahí se selecciona la opción *Countries* (el comportamiento de las opciones *Suppliers y Products* es análogo).

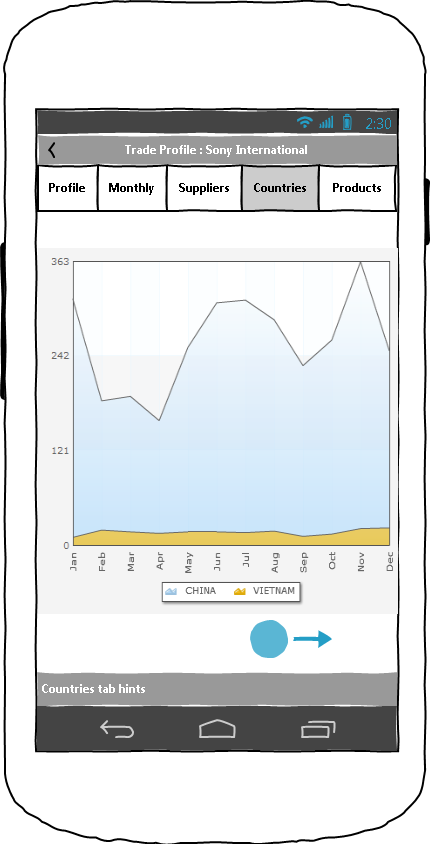
Se mostrará el ranking para de importaciones de la empresa para la dimensión *Countries:*



Si el usuario selecciona uno de los valores del ranking, se mostrara la gráfica de la evolución mensual para el periodo de un año:



Desde la ventana anterior avanzando la página se puede ver una gráfica con la evolución anual de todos los elementos del ranking:



# Análisis de riesgos tecnológicos

Se detectaron tres riesgos tecnológicos que nos parecieron que pueden ser bloqueantes para el desarrollo.

## Comunicación con servidor remoto a través de webservices REST

Para mitigar este riesgo se realizó la practica 5. En dicha práctica se encontró un problema con el webservice de Google que ya no permite la traducción a través de la URL provista, por lo que la cambiamos por la URL que utiliza la página de traducción de Google: http://translate.google.com/translate\_a/t?client=t&text=hola%20mundo&hl=es&sl=es&tl=en&multires=1 que retorna matriz multidimensional con los resultados.

El resultado de la traducción se encuentra en retorno [0][0][0], en formato string.

Se utilizó JsonArray para parsear el resultado y acceder al dato específico.

Se adjunta proyecto Eclipse.

## Incorporar una librería de graficas de terceros

Se encontraron varias librerías de gráficas, y se eligió TeeChart, ya que son las que están más cuidadas estéticamente.

Se está utilizando una versión de prueba por 50 días totalmente funcional que agrega una marca de agua en las gráficas.

En el prototipo se crea una Activity con una gráfica de barras.

Se adjunta proyecto Eclipse.

## Persistencia en una base de datos local al dispositivo

Para mitigar los riesgos de utilizar una base de datos en el dispositivo, se siguió un tutorial para crear una base de datos y realizar INSERTs.

Y luego se realiza un SELECT sobre la tabla y se escribe en la salida estándar.

Se adjunta proyecto Eclipse.

# Análisis luego de haber desarrollado la aplicación

## Requerimientos funcionales

### RF1: Login del usuario

Este requerimiento no pudo implementarse por falta de tiempo, tuvimos que priorizar otras funcionalidades del sistema.

Igualmente si bien le dimos prioridad alta, consideramos que es trivial y que no implicaría investigación o desarrollo complejo, por lo tanto no aportaría tanto a la aplicación.

### RF4: Búsqueda por nombre de empresa

Se implemento la búsqueda solo por Consignee, estaba planificado además hacer la búsqueda por Shipper.

Al tener que priorizar y ajustarse al tiempo disponible, decidimos hacer solo una de las búsquedas porque son análogas y a los efectos del curso no hay diferencia entre ellas, solo los clientes verdaderos de la aplicación pueden distinguir y necesitar ambas funcionalidades.

### RF5: Acceso a información de contacto

Se implemento según lo planificado, esta información se muestra en la primera página al elegir el resultado de una búsqueda.

Se muestran los datos más significativos de las empresas, lo que devuelve el servidor de Trade Profiles es un conjunto muy extenso de campos (alrededor de 160 campos diferentes).

Para mostrar la información se uso el componente WebView de Android, ya que fue la única manera de lograr un centrado y alineación aceptables.

### RF6: Totales mensuales

Se implemento según lo planificado, esta información se muestra en la segunda página al elegir el resultado de una búsqueda.

Se muestran los datos para el periodo de un año, así se configuro el servidor de Trade Profiles, en la aplicación real esto sería parametrizable.

La complejidad del desarrollo de esta página es notoriamente mayor al de la página de información de contacto, ya que hubo que investigar mucho para poder incorporar las graficas de la librería de terceros dentro de los layouts animados con paginación que usamos.

Si bien esto se probó en la etapa de análisis de riesgos, no se había probado exactamente como se iba a usar (porque fue cambiando sobre el desarrollo), y al fin de cuentas resultó ser mucho más complejo de lo que se pensaba.

### RF7: Totales por dimensión

Se implemento según lo planificado, esta información se muestra en desde la tercera página en adelante al elegir el resultado de una búsqueda.

Los comentarios sobre el desarrollo de este RF son análogos a los del RF6, lo único que cambia es la lógica para armar la grafica, pero eso no es un problema de la plataforma Android y es trivial.

### RF8: Graficas

Se implemento según lo planificado, se pueden ver desde la segunda hoja en adelante al elegir el resultado de una búsqueda.

Ya mencionamos algo de la complejidad de incorporar esta funcionalidad, incluso llegamos a pensar en resolverlas con HTML y no con la librería de terceros.

## Requerimientos no funcionales

### RNF1: Extensibilidad

Intentamos implementar este mecanismo mediante el uso de un Dashboard, donde cada opción del mismo es resuelto por una Activity diferente, de esta manera lograríamos extender la aplicación.

Sin embargo para lanzar una versión nueva los usuarios tendrían que bajarse la aplicación entera nueva, así que desde el punto de vista del cliente no se ve mucho la extensibilidad.

Se implemento por completo la opción de *Trade Profiles*, se tuvo que dejar de lado la opción de

*My Account* por temas de tiempo. Esa opción entre otras funcionalidades iba a permitirle al usuario gestionar sus datos personales, direcciones de correo, contraseña, tamaños del cache de *Profiles* descargados entre otras cosas.

Para completar el look del Dashboard agregamos dos iconos dummy, de esa manera se puede ver cómo sería una versión más completa de la aplicación.

### RNF2: Plataforma

Como nuestra aplicación apunta a tabletas principalmente (por la cantidad de datos que se muestran y para que las graficas puedan ser grandes), nuestra idea inicial era que el requerimiento mínimo de plataforma sea Android 3.x y superiores.

Pruebas realizadas con emulador v3.0 y 4.0.3 (normal y tablet) y dispositivos físicos v4.0.3.

Es algo para tener en cuenta cuando la aplicación tome el rumbo comercial real, pero para el alcance de este curso decidimos dejarlo así para no complicarnos el desarrollo y poder seguir probando tranquilamente.

### RNF3: Comunicación remota

Esta funcionalidad se implemento y de hecho es fundamental para la aplicación, sin ella el programa no sirve para nada.

### RNF4: GPS y Google Maps API

Este RNF fue planificado con prioridad baja ya que no aportaba mucho al core de la aplicación, y en la práctica se mantuvo así. Realmente no llegamos a plantearnos esta funcionalidad por falta de tiempo y priorizar lo realmente importante.

### RNF5: Cacheo de datos

Decidimos darle más importancia que la prioridad baja que se planifico al principio, dado que es bastante común que el usuario o no tenga acceso a datos por internet o simplemente en determinados momentos no quiera usarlos.

Implementamos un cacheo de datos donde cada *Profile* descargado durante el modo online es guardado en un archivo en la tarjeta SD del dispositivo si esta está disponible.

Adicionalmente se guarda en una base de datos cuales *Profiles* están cacheados para mostrarlos rápidamente al usuario, previo a esto se chequea que los *Profiles* que dice la base de datos están cacheados realmente existan en la tarjeta SD, de lo contrario la base se actualiza para evitar inconsistencias.

El usuario desde el Dashboard y accediendo a la ActionBar tiene una opción para pasarse al *Modo Offline*, de esta manera ya no se harán peticiones por internet y se mostraran solo los *Profiles* que fueron cacheados previamente.

Creemos que esta funcionalidad fue subestimada en un principio, ya que acceder a los datos cacheados permite una gran versatilidad a la aplicación.

### RNF6: Orientación de la pantalla

Esto es fundamental y le dimos la importancia necesaria generando layouts para cada orientación y rediseñando las interacciones para que aprovechen el formato en cada caso.

También tuvimos en cuenta las pantallas más chicas que tableta usando parámetros de los layouts para que se ajusten automáticamente según el tamaño de la pantalla.

Esto permitió que la aplicación sea apta para teléfonos como el Samsung Nexus S de Rodrigo que tiene una pantalla sensiblemente más pequeña que una tableta.

# El diseño e implementación

Al momento de pasar el prototipo a código, debimos tomar ciertas decisiones que no habíamos contemplado, por ejemplo habíamos realizado el prototipo en orientación vertical y no landscape.

Decisiones que tomamos:

## Dispositivos soportados

Inicialmente el desarrollo se iba a realizar solo para dispositivos de porte grande (tablets), pero decidimos dar soporte a dispositivos de menor porte como celulares (con ICS como mínimo).

Esto lo logramos haciendo uso de distintos layouts (layout y layout-land para pantallas de tamaño normal, ej. Celulares, y para tablets: layout-large y layout-large –land).

Las mayores diferencias entre los layouts para los distintos dispositivos son los siguientes (referidos a los Trade Profiles):

### Pantallas normales:

Para los celulares, las vistas para las orientaciones horizontal o vertical no se cambian en funcionamiento. Al entrar al trade profile se dispara la actividad TradeProfilesActivity, que en este caso muestra un searchview y la lista de resultados de la búsqueda debajo. Al seleccionar un resultado se dispara la actividad DetailsActivity, que despliega los datos del ítem seleccionado haciendo uso de una ViewPager para mostrar los distintos tabs.

### Pantallas grandes:

En tablets, en modo vertical se comporta como con pantallas pequeñas, el cambio lo notamos cuando se pasa a modo landscape, en donde la pantalla se divide en dos secciones, haciendo uso de fragmentos: uno a la izquierda que contiene un searchview y la lista de resultados de la búsqueda (ListFragment).

Al hacer click en un elemento de los resultados, se dispara un evento que realiza la búsqueda del profile seleccionado (atreves de un webservice de rest), y desplegando el resultado en un fragmento en la sección derecha de la pantalla (en caso de layout landscape, si la orientación es vertical se dispara una nueva actividad DetailsActivity para que ocupe toda la pantalla), en dicho fragmento o actividad se despliega un ViewPager que permite mostrar los distintos tabs en forma de pagina (estilo google play) permitiendo el scroll horizontal para su navegación.

## Flujo de Actividades

### Tabletas en orientación landscape

## Image_15

Al iniciar la aplicación se muestra el *Dashboard* (*HomeActivity*), desde ahí al acceder a la opción de *Trade Profiles* se crea la Activitycorrespondiente al modo de trabajo del usuario (offline u online).

Esa Activity tiene la barra de búsqueda y lista de resultados de búsqueda a la izquierda (*TitlesFragment*) y la vista de detalles a la derecha (*PagerView*). Finalmente al hacer *swipe* para moverse a la derecha se van creando las páginas con las tablas y graficas (*PageFragment*).

## 

### Celulares u orientación portrait

## Image_11

Al iniciar la aplicación se muestra el *Dashboard* (*HomeActivity*), desde ahí al acceder a la opción de *Trade Profiles* se crea la Activitycorrespondiente al modo de trabajo del usuario (offline u online). Esa Activity tiene la barra de búsqueda y lista de resultados de búsqueda (*TitlesFragment*).

Una vez ejecutada la búsqueda y elegido un resultado, se inicia la Activity DetailsActivity, esta contiene y la vista de paginas detalle (*PagerView*). Finalmente al hacer *swipe* para moverse a la derecha se van creando las páginas con las tablas y graficas (*PageFragment*).

## Comunicación

La comunicación con el servidor se realiza por medio de webservices rest.

Al momento del desarrollo de la aplicación móvil, el servidor que provee los datos se encontraba en el siguiente dominio beta, pero próximamente se pondrá en producción:

* Dominio Beta: [http://200.40.197.173:8082](http://200.40.197.173:8082/system)
* Dominio Producción: [http://www.datamyne20.com](http://www.datamyne20.com/)

Por lo que dejamos comentado en el código el dominio de producción (clase RestTradeProfileClient2, línea 19, constante BASE\_SERVER).

El dominio beta se encuentra disponible, y en breve estará disponible el dominio de producción, en cuyo caso les comunicaremos el cambio para que realicen la modificación en el código.

Los dos webservices usados son:

* /system/rest/fullTradeprofile/consignee/6795958  
  donde el ultimo segmento del path “6795958” representa el id de consignee a obtener.
* /system/rest/autocomplete?Base=usa\_hid12&idComponent=402&compositeid=402&targetTerm=dole  
  donde el parámetro “targetTerm” contiene el texto a buscar , en este caso “dole”.

Para probar que servicio está levantado acceder a:  
Dominio/system/rest/autocomplete?Base=usa\_hid12&idComponent=402&compositeid=402&targetTerm=dole  
Y si se tiene respuesta se puede usar dicho dominio.

Tanto la búsqueda por consignee como obtener un profile hacen uso de clases que extienden AsyncTask que corren en su propio thread al invocarle el método execute, HttpClientTask y DetailsAsyncTask respectivamente. Esto es necesario ya que queremos que la aplicación no aparezca como congelada mientras se están obteniendo los datos desde el servidor, y mostrar un dialogo de carga.

La clase AsyncTask provee métodos que sobrescribimos:

* doInBackground código a ejecutarse dentro del thread. No se puede modificar la UI (acceder a elementos dentro una activity) ya que los threads no pueden acceder al thread principal. Para poder acceder a la UI de forma segura nos brindan los siguiente métodos
* onPreExecute: permite ejecución de código antes de arrancar el thread.
* onPostExecute: permite el acceso a objetos en el thread principal y se ejecuta al finalizar el código del thread. Lo usamos para cerrar el dialogo de progreso y actualizar los elementos de la UI, con por ejemplo el contenido de una página del profile o los resultados de una búsqueda.
* onProgressUpdate este código es disparado de forma segura al invocar publishProgress dentro del thread. Permite actualizar la UI (ejemplo barra de progreso en un dialogo de progreso). Lo usamos para comunicar cuando hubo un error o excepción.
* onCancelled. Evento disparado cuando se invoca el método cancel (true), y también permite el acceso seguro a la UI. Nosotros lo usamos para comunicar errores o excepciones en el dialogo de progreso creado.

## Accediendo a un profile

La tarea asincrónica DetailsAsyncTask, tiene 3 responsabilidades claras encapsuladas:

* Obtener los datos cacheados en la SD ( en caso de estar disponible)
* Si el profile no está cacheado, debe obtenerlo contactando al servidor, y
* guardarlo la respuesta en la SD(si está disponible) y finalmente
* si se pudo salvar en la SD se da de alta en la base de datos, para marcar el profile como cacheado y poder usarlo en modo offline

Todas las clases involucradas con estas responsabilidades se encuentran en el paquete com.datamyne.mobile.providers:

* IDatabaseProfileProvider, DataBaseProfileProvider, ProfilesSQLiteHelper: la interface, la implementación y el helper encargados de manejar la base de datos
* IProfileProvider, ProfileProvider: la interface y la implementación encargada de crear y leer archivos desde la SD, y en caso de no encontrarse un archivo pedido, invocar cliente rest para obtener los datos desde el servidor.
* IRestTradeProfileClient, RestTradeProfileClient2 : la interface y la implementación encargada de acceder al servidor y retornar **los datos obtenidos**

## Generando la vista de un profile

Para la generación de la vista de un profile (datos mostrados en el ViewPager), como mencionamos antes usamos una DetailsAsyncTask para acceder a los datos, que en onPostExecute ejecuta el método writeData. Este método es el encargado de generar la página pedida por el adapter del pager.

El layout de una página esta descripto por “fragment\_trade\_profile\_detail\_pager.xml”, donde se tiene un contenedor (linearLayout vertical) con un textView (títulos), un contendor linearLayout (@+id/pageDataTable) que contendrá los datos de la tabla, y el último contenedor linearLayout (@+id/linearLayoutTChart) que contendrá la grafica.

El layout es el mismo para todas las paginas, pero dependiendo de la pagina se cargaran o no los contenedores con las graficas, tablas y títulos.

Se proveen dos interfaces para una misma implementación (ChartCreator) que tiene las siguientes responsabilidades:

* ITabTableCreator : Generar Tablas de datos
  + Generar un View que contenga los datos del profile   
    crearTablaTabProfile(JSONObject data). Retorna un webView.
  + Generar un View que contenga la tabla de datos de Totales anuales crearTablaTabMonthly(JSONObject data). Retorna un tableLayout
  + Generar un View que contenga la tabla (correspondiente a la dimensión pasado por parámetro) de datos para el resto de las dimensiones   
    crearTablaTabOther(JSONObject data). Retorna un tableLayout
* IChartsCreator: Generar Graficas
  + Generar la grafica para totales anuales  
    crearGraficaMonthly(JSONObject data). Retorna un TChart.
  + Generar el resto de las graficas dependiendo de los datos recibidos por parámetro crearGraficaMulti(JSONObject data). Retorna un TChart.

Estas clases e interfaces se encuentran en el paquete: com.datamyne.mobile.profile.utils

* HoneycombCompatibility: clase que provee compatibilidad con Honeycomb, ejecutando el método setLogo por reflection de manera de poder compilar el código tanto con SDK 11 o superior, y en caso de ser compilado con ICS (SDK >14) tener la funcionalidad de setear la imagen del logo de las actionBars.
* Item: clase auxiliar usada para almacenar los resultados de las búsquedas por empresa, se utiliza para pasar los resultados desde la asyncTask a la UI. Además implementa Parcelable, para poder guardar la lista de ítems encontrados en el estado de la actividad o fragment de manera de que cuando se rota la pantalla mantener los resultados sin tirar la búsqueda de nuevo.
* DetailsAsyncTask: clase que ya comentamos antes.
* ITabTableCreator, IChartsCreator, ChartCreator: Interfaces y clases encargadas de generar los datos de los detalles de los profiles, mencionados antes.
* DrawView: view que debimos implementar para poder generar un cuadrado rellenado de color dinámicamente que usamos como leyenda de las graficas de dimensiones. Para implementar esta clase debimos sobrescribir los métodos onDraw y onMeasure, en los que dibujamos en el canvas el cuadrado y lo rellenamos con el color pasado por parámetro al constructor. Se deber sobrescribir onMeasure, ya que define el alto y ancho que va a tener el DrawView creado, si no se sobrescribe con un valor (en nuestro caso 20x20

## Base de datos

Esta tecnología se probó en la etapa de análisis de riesgos y no presento complicaciones al incorporarla en la aplicación.

Se usa para almacenar cuales *Profiles* fueron descargados durante el modo online y están disponibles para el modo offline. La base se guarda en la SD con el nombre **DB\_profiles** y se crea en caso que no exista, consta de una sola tabla hasta el momento:

CREATE TABLE profiles (

\_id INTEGER PRIMARY KEY AUTOINCREMENT, --identificador autogenerado

type TEXT, --tipo de profile, por ahora solo “consignee”

code INTEGER, --código interno de empresa

name TEXT, --nombre de empresa

file\_path TEXT --ruta absoluta al archivo donde se guardo el profile

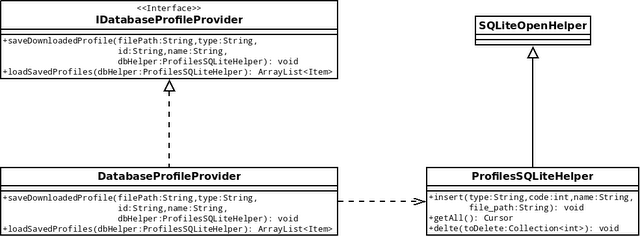
);

Cada vez que se pide un *Profile* en el modo online y este debe descargarse, se ejecuta el método que guarda los datos de ese *Profile* en la base.

Cuando se trabaja en modo offline y al ingresar en la Activity de *Trade Profiles*, primero se ejecuta un método para eliminar de la base los *Profiles* cuyo archivo físico no exista, para evitar inconsistencias entre la base y los archivos guardados en la tarjeta SD. Luego se hace una consulta para obtener todos los *Profiles* almacenados en la base y mostrarlos al usuario.

Todo este comportamiento se encapsulo en las clases:

* SQLiteOpenHelper: creación de la base, tabla, inserción y borrado de registros.
* IDatabaseProfileProvider: interfaz para la interacción con la aplicación define los métodos de inserción y obtención de *Profiles.*
* DatabaseProfileProvider: implementa la interfaz IDatabaseProfileProvider.



# Aspectos a mejorar

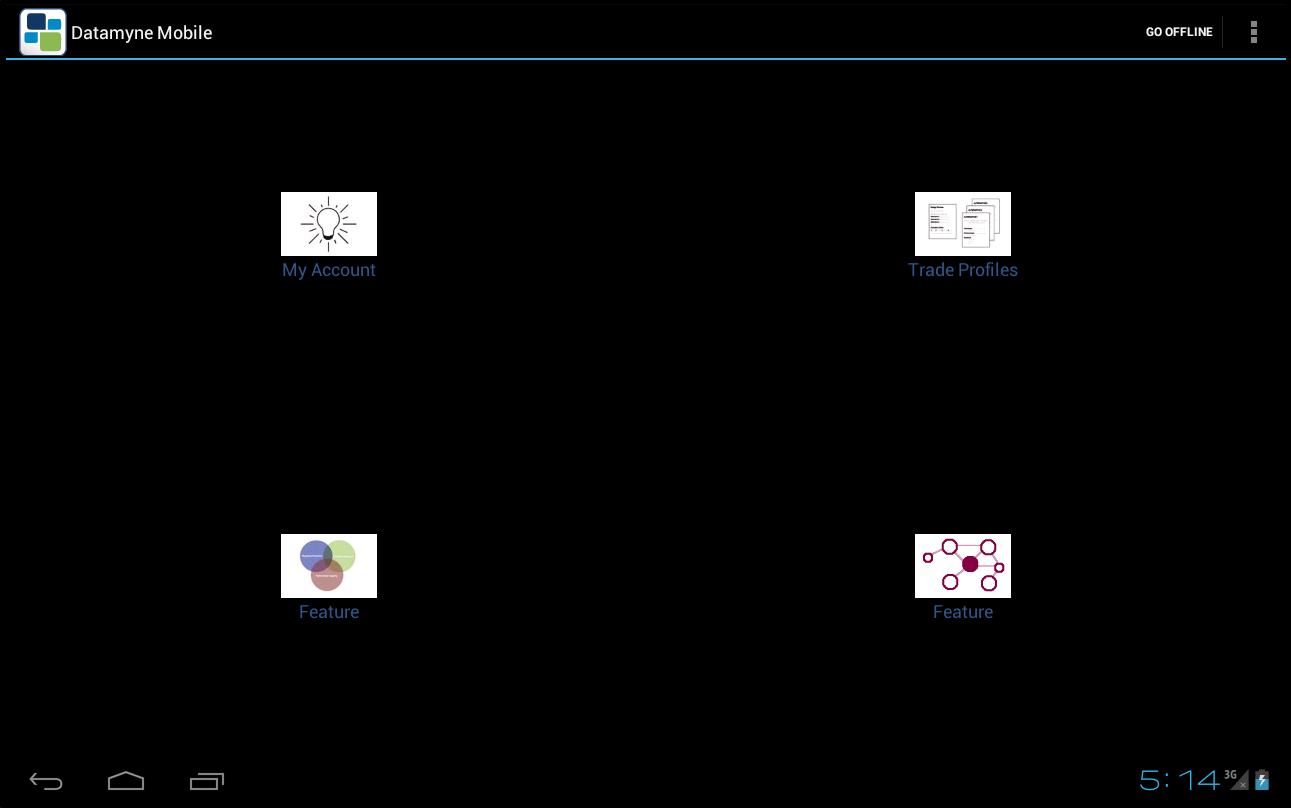
* El código fuente no está prolijo, hace falta hacer varios refactor para mejorar la mantenibilidad.  
  En particular habría que generalizar las Activities que resuelven los *Trade Profiles* online y offline y extraer una clase abstracta. Actualmente hay duplicación de código producto del desarrollo basado en pruebas y prototipos.

Otro aspecto que no ayuda en este rubro es la inclusión de varias *inner clases*, dificultan el entendimiento del código y lo hacen feo. No sabemos si siempre es así en los proyectos de Android, pero nos vimos obligados en cierta forma a este tipo de soluciones dado que se necesitan tener los contextos de las Activities a mano para pasarlos entre ellas.

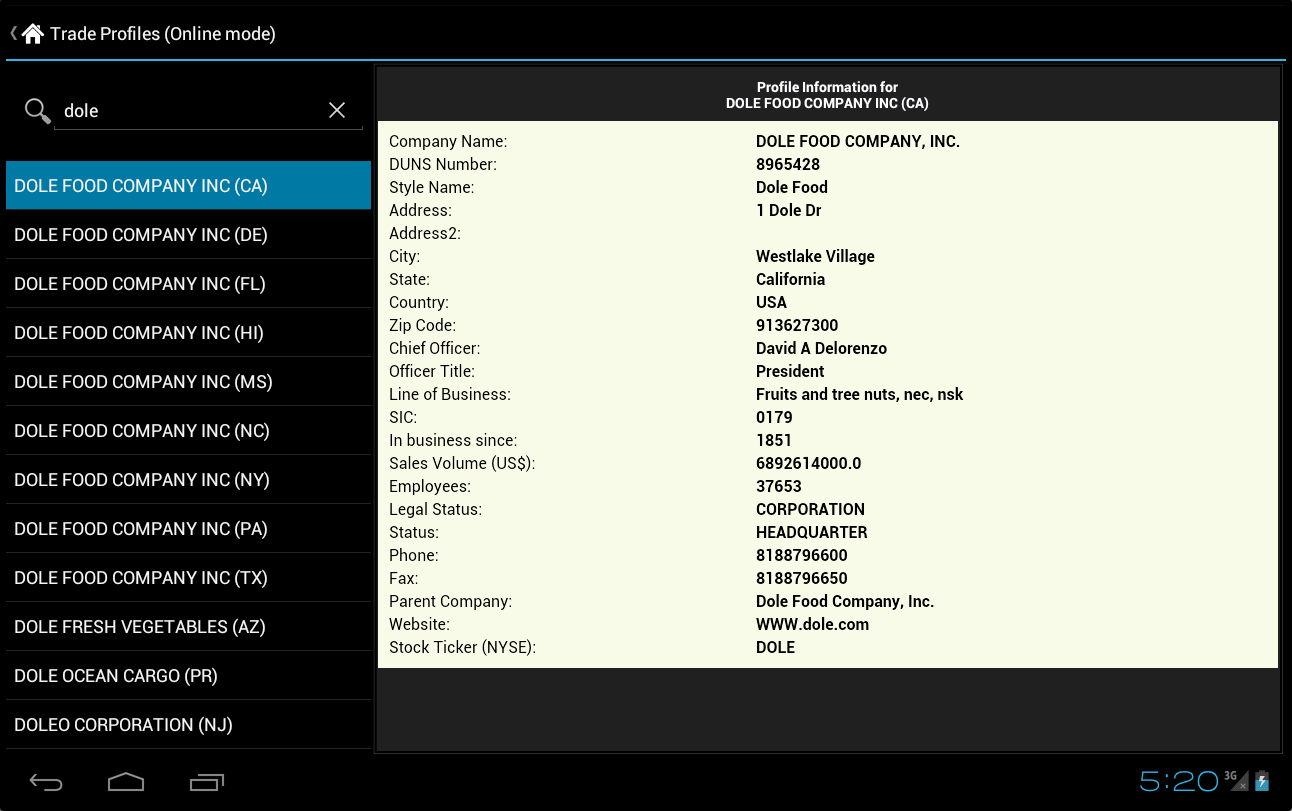
* *Look & Feel* de la aplicación, lleva mucho trabajo y atención en los detalles. Sabemos que estas cosas hacen la diferencia pero tuvimos que priorizar la funcionalidad.
* Mejorar el de los errores producidos por la falta de recursos externos, por ejemplo la tarjeta SD o conexión a internet, que aunque se capturan y muestran mensajes, en algunos casos se disparan dos diálogos cuando no se tiene SD (modo on the fly) y sin conexión al momento de pedir un profile (un dialogo por pagina, por cómo funciona el ViewPager.
* La aplicación debería detectar la no presencia de conexión a internet, cambiar de modo online/offline automáticamente (previo consentimiento del usuario) y notificar al usuario claramente en la GUI.
* En la vista de páginas del resultado de una búsqueda, faltan los tabs que indiquen que tipo de información se está mirando. Perdimos mucho tiempo en intentar hacer funcionar los tabs dentro de nuestros layouts sin éxito.
* Agregar seguridad y login, que no llegamos a implementar.
* Mejorar el layout del Dashboard.

# Capturas de pantalla y manual de usuario

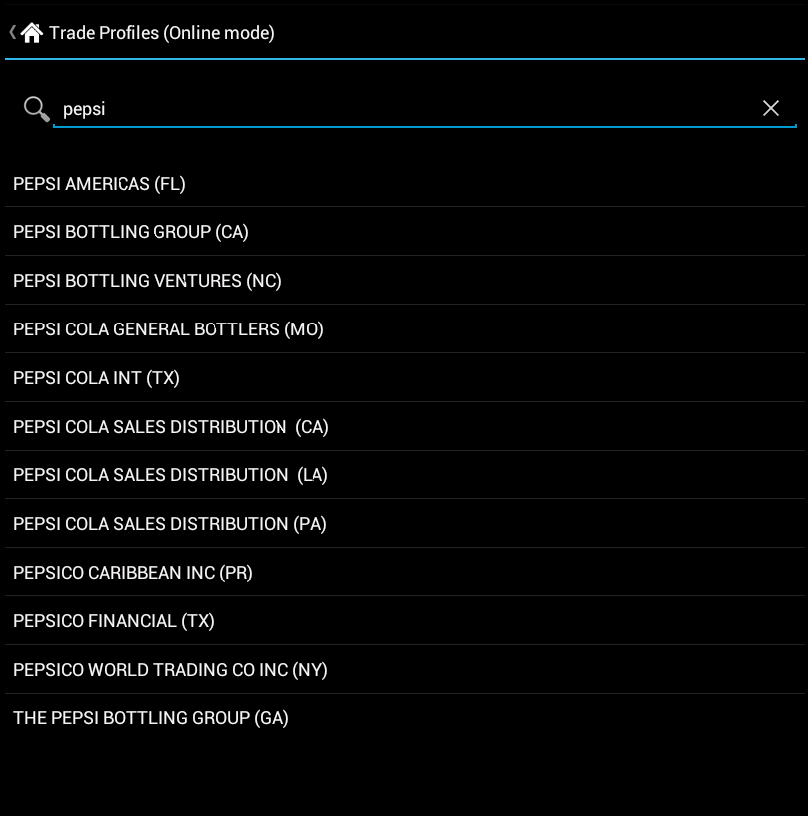
El Dashboard de la aplicación, es lo que se muestra al ejecutarla y desde él se acceden a las diferentes funcionalidades:

  
En el ActionBar se presenta el botón para cambio de modo a conectado/desconectado (offline solo se permiten ver los *Profiles* descargados anteriormente y en modo online busca y obtiene los *Profiles* desde nuestro servidor), y en el menú se dispone de la opción para cerrar la aplicación.

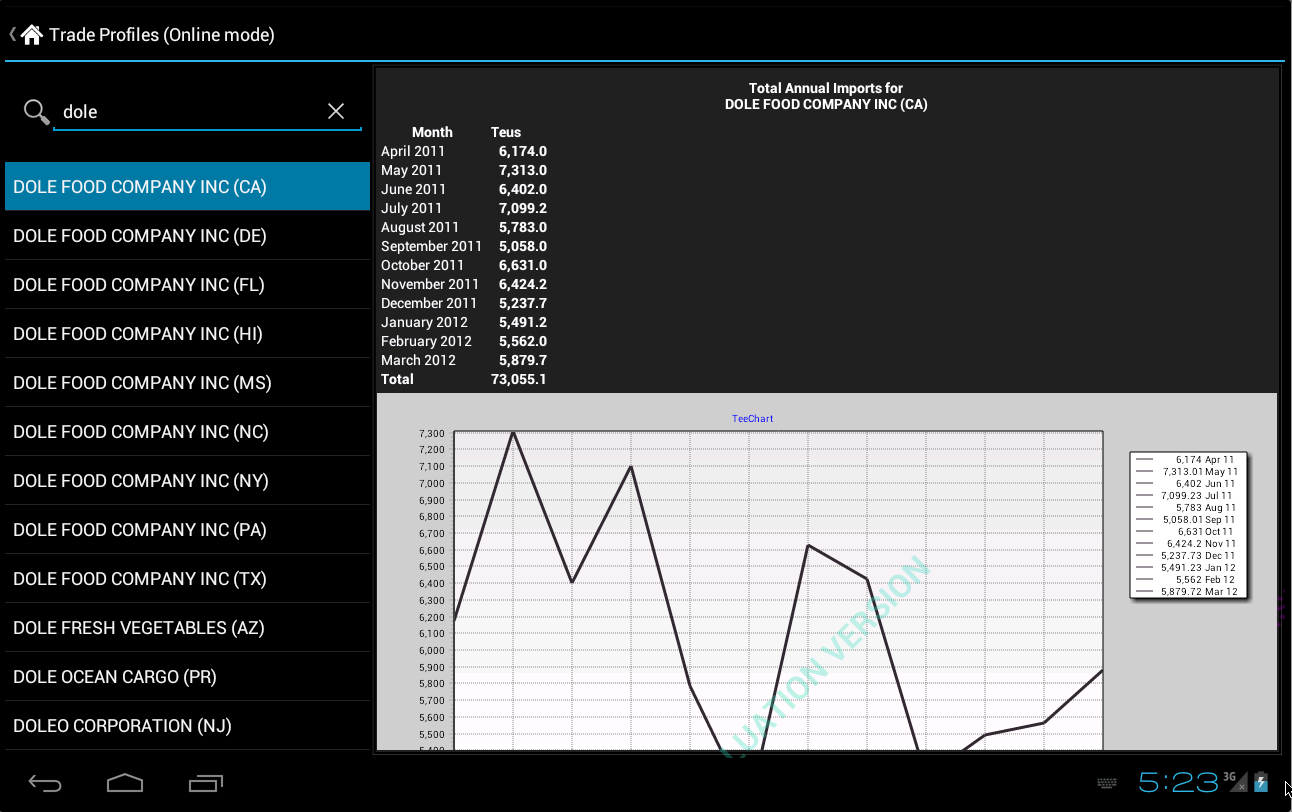
Si elegimos el icono de *Trade Profiles* accedemos a la ventana de búsqueda. En este ejemplo se busco el texto “dole” (una empresa conocida en el mundo de los Consignees) y se eligió el primer resultado de la lista de la izquierda. Eso provoco que se cargue el *Profile* y se muestre la primera página a la derecha, los datos de contacto de la empresa:



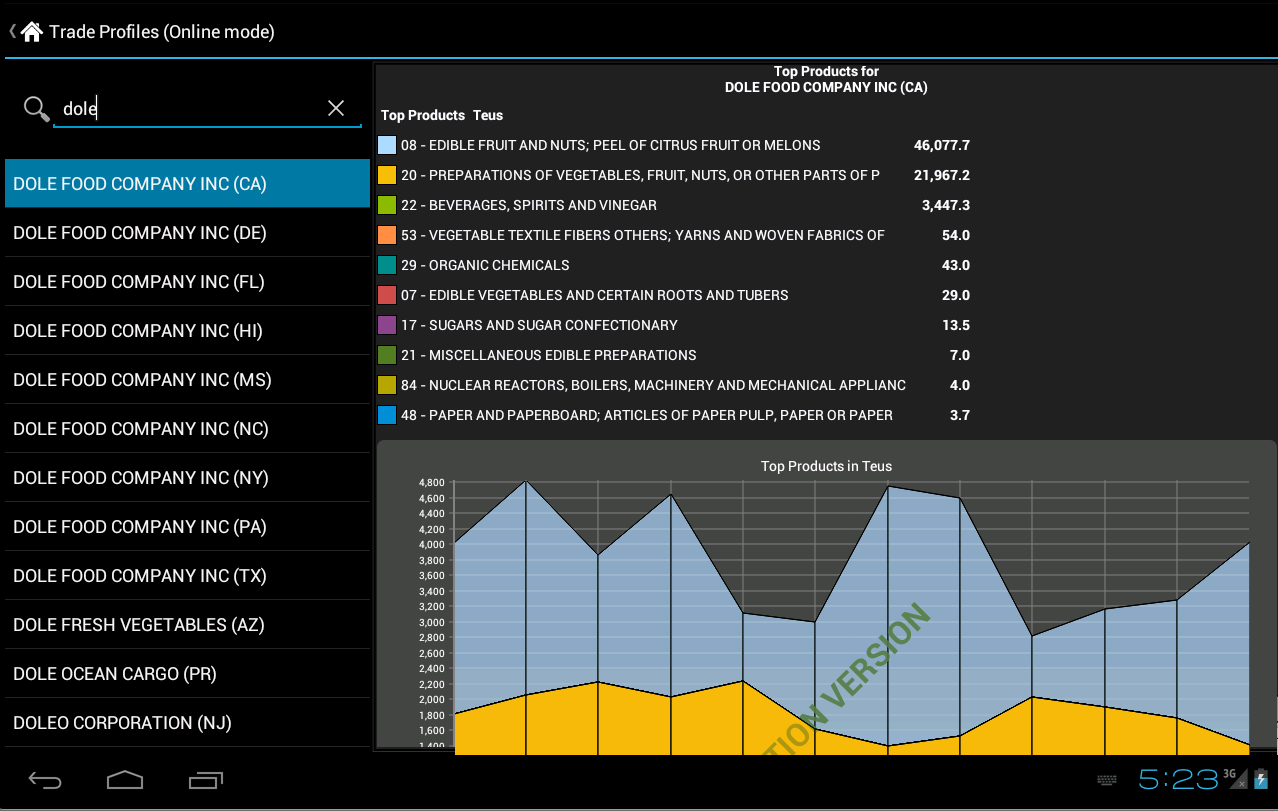
La misma ventana de búsqueda pero con el Layout vertical (notar cómo cambian los componentes):



Volviendo al layout horizontal, hacemos *swipe* para mover la página a la derecha, podemos ver los totales mensuales de importaciones de la empresa seleccionada:



Haciendo *swipe* a la derecha varias veces llegamos a la página de los top productos que importo la empresa seleccionada:



Todas las *Activities* tienen el botón *Home* arriba a la izquierda para volver al dashboard.

En general la aplicación se parece bastante a los mockups que diseñamos al principio del semestre, un gran faltante son los tabs en las paginas donde se muestra la información.

# Conclusiones

Todas las veces que en este documento mencionamos falta de tiempo no estamos poniendo excusas, sino marcando un error de nuestra parte. Sin duda subestimamos la complejidad de la tecnología y lo poco maduro de las herramientas. Todo nos llevaba mucho más tiempo de lo que llevaría en otra herramienta y plataforma conocida.

Fue una buena experiencia (al menos para nosotros que no conocíamos nada de Android), pero nos hubiera gustado agarrarle la mano con más tiempo para hacer una aplicación de mejor calidad. A priori parecía una elección que se caía de madura dado que se desarrolla en Java, un lenguaje dominado por nosotros, pero aun así se hizo muy complicado.

En cuanto al alcance no estuvimos tan errados, solo dejamos afuera el Login del Usuario y el uso de GPS y la API de mapas, no son elementos menores, pero vistas las cartas ahora no planificamos tan mal.