



www.centic.es

Curso Android Edición 2013



CENTRO TECNOLÓGICO DE LAS TECNOLOGÍAS DE
LA INFORMACIÓN Y LAS COMUNICACIONES

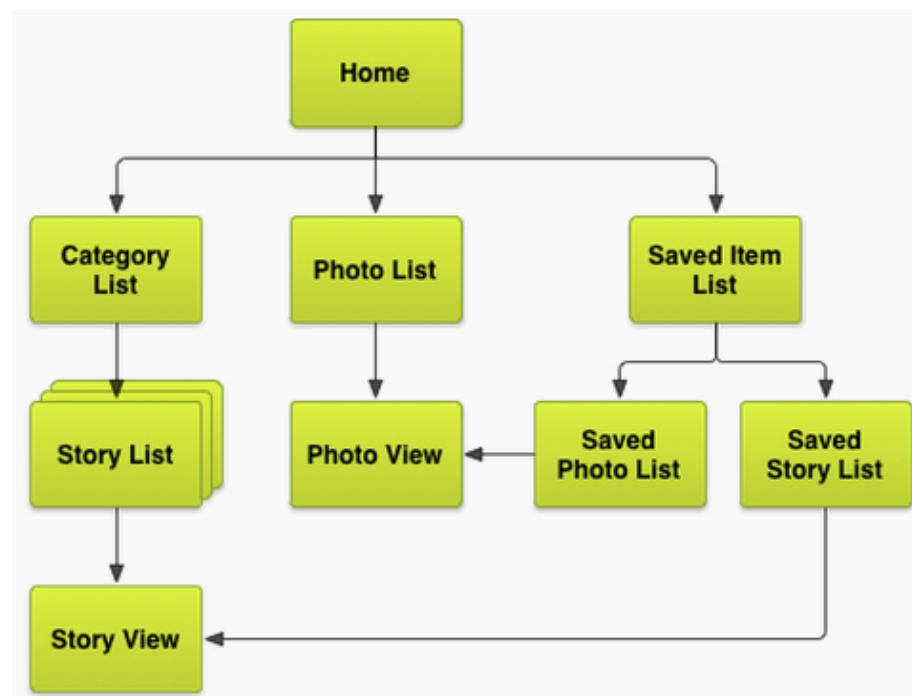


Interfaz de usuario

Navegación Coherente.

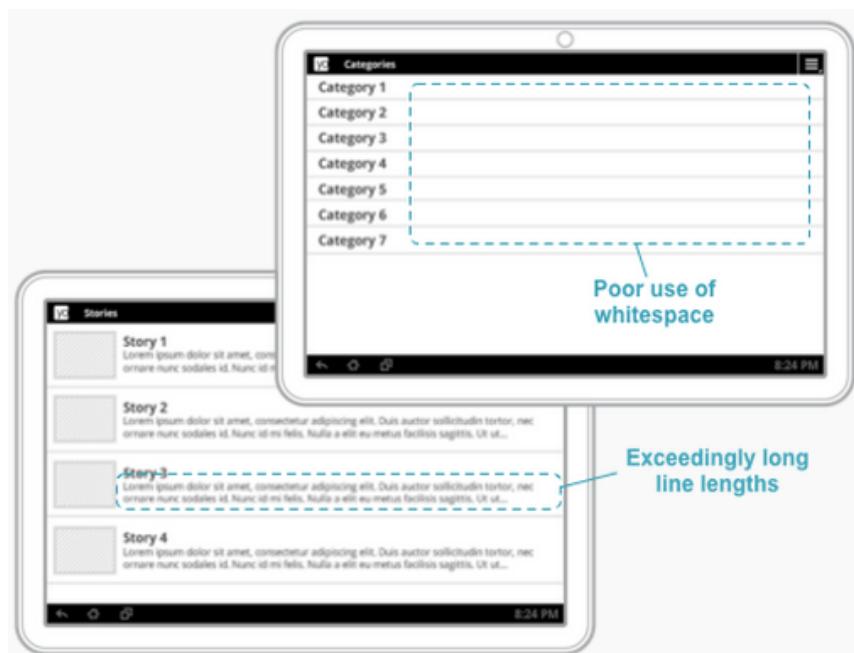
Una vez definido el modelo de información, debemos comenzar a definir el contexto necesario para permitir a los usuarios encontrar fácilmente los datos en la aplicación.

El conjunto de pantallas que necesitaremos variará en función del dispositivo en el que se ejecuta la aplicación.



Interfaz de usuario

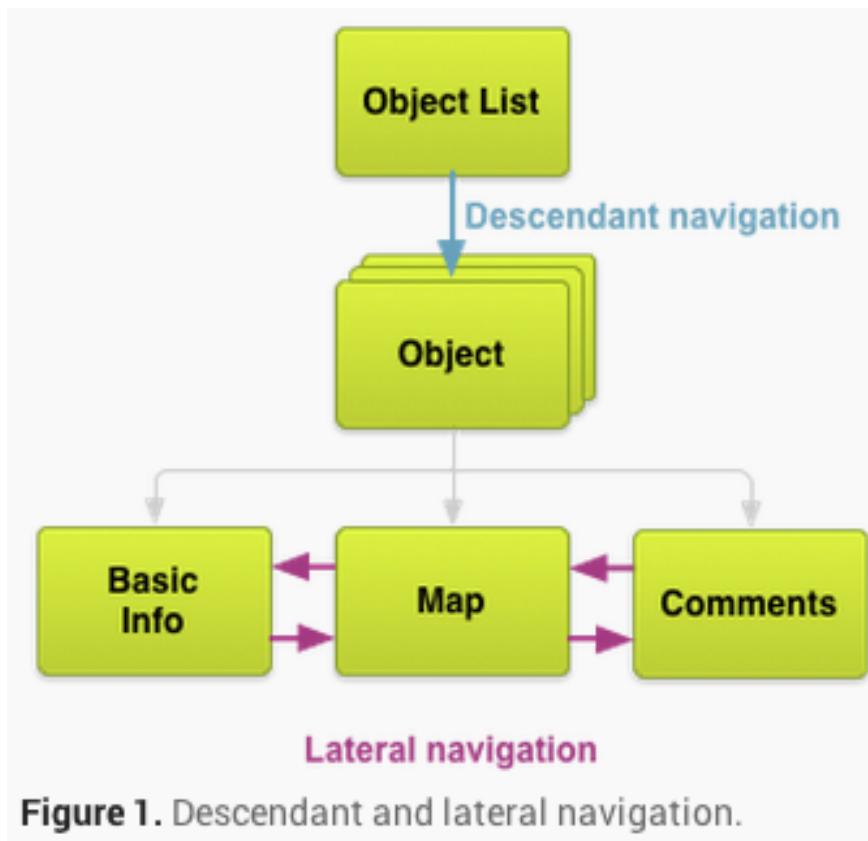
Navegación Coherente.



Interfaz de usuario

Navegación Coherente.

Navegación descendente dentro de una jerarquía.

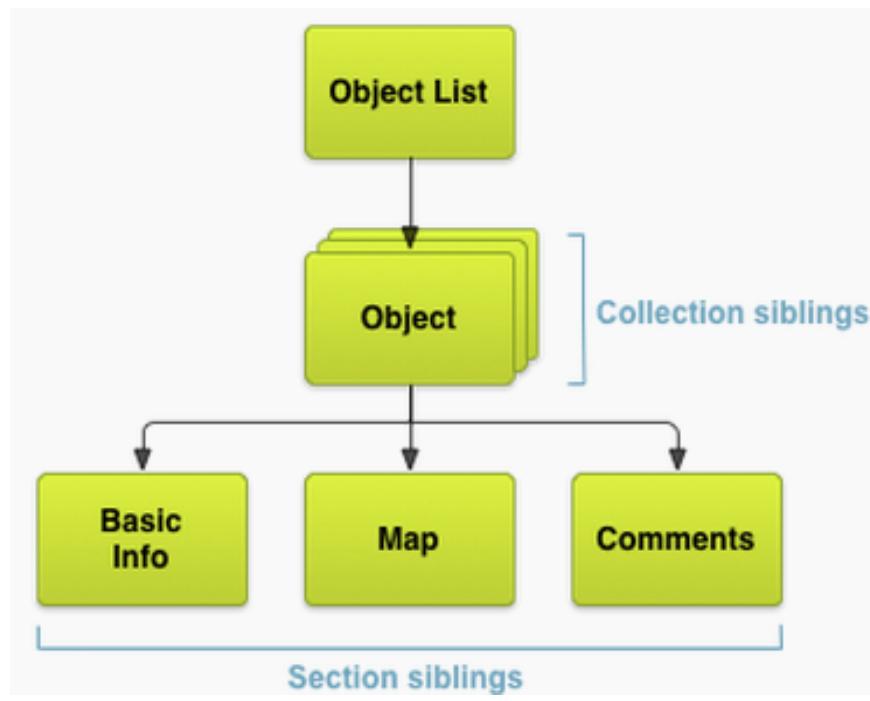


Interfaz de usuario

Navegación Coherente.

Hay dos tipos de relaciones de pantallas al mismo nivel.

- Pantallas relacionadas por colección.
- Pantallas relacionadas por sección.



Interfaz de usuario

Navegación Coherente.

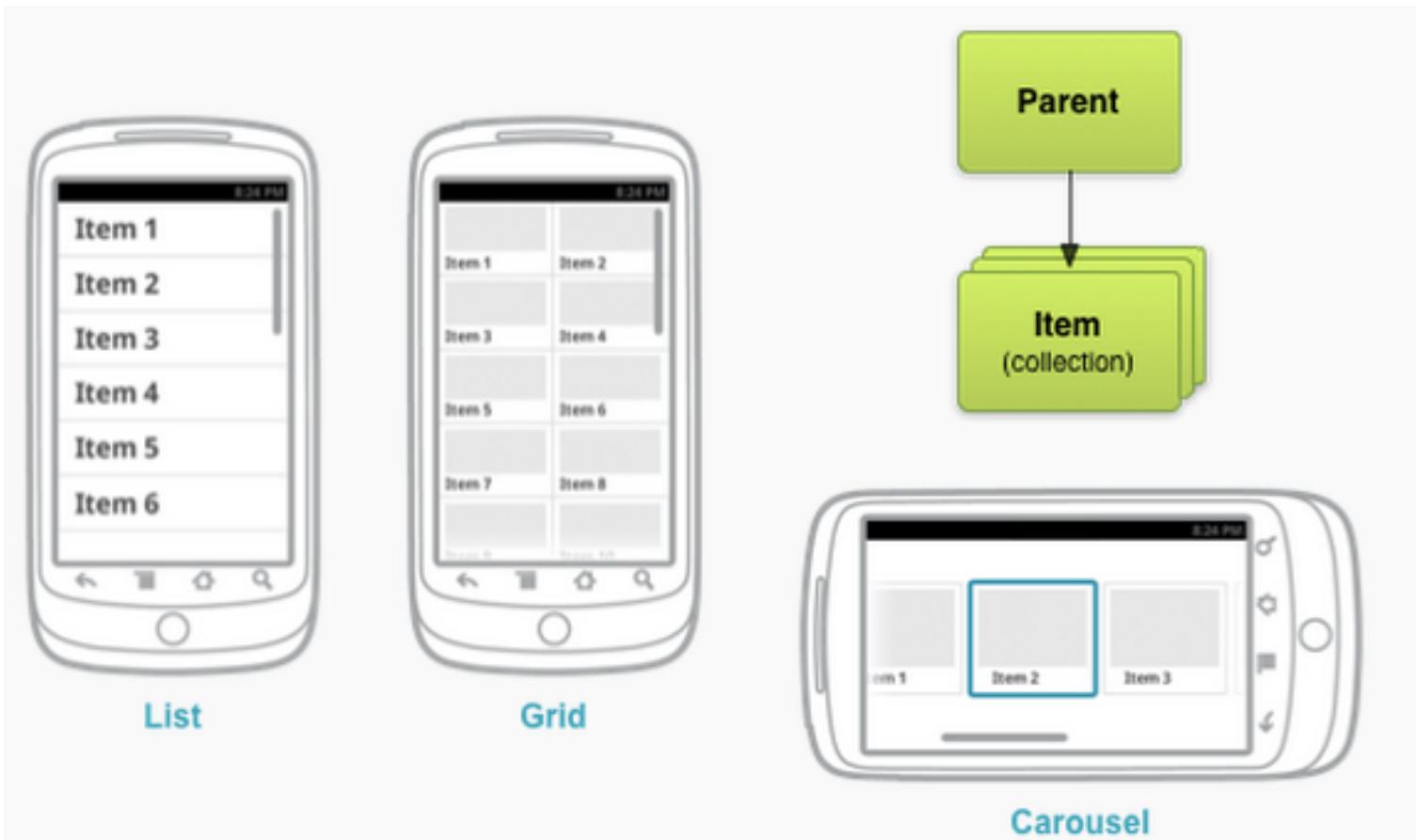
Patrón Dashboard



Interfaz de usuario

Navegación Coherente.

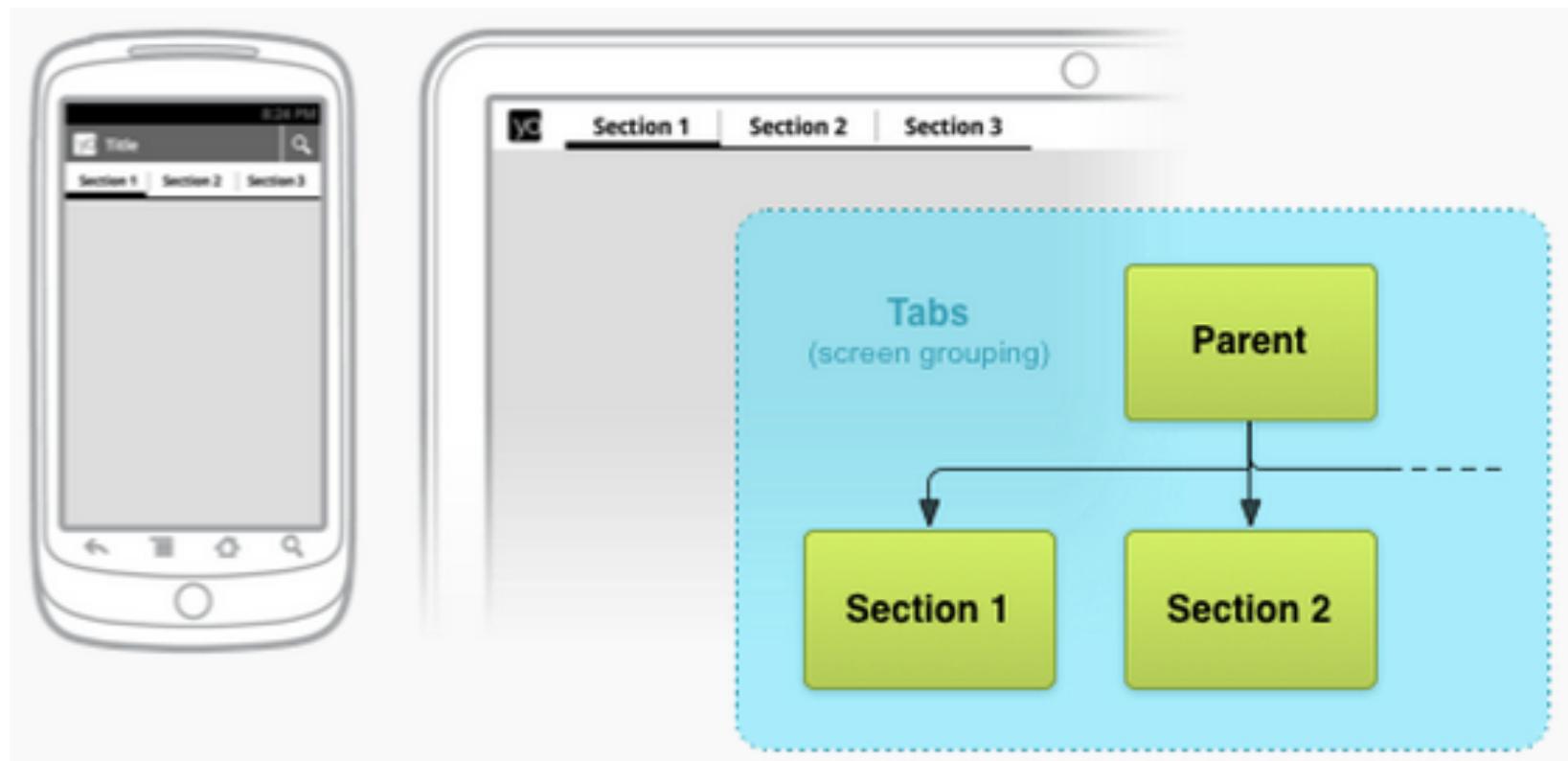
Listados.



Interfaz de usuario

Navegación Coherente.

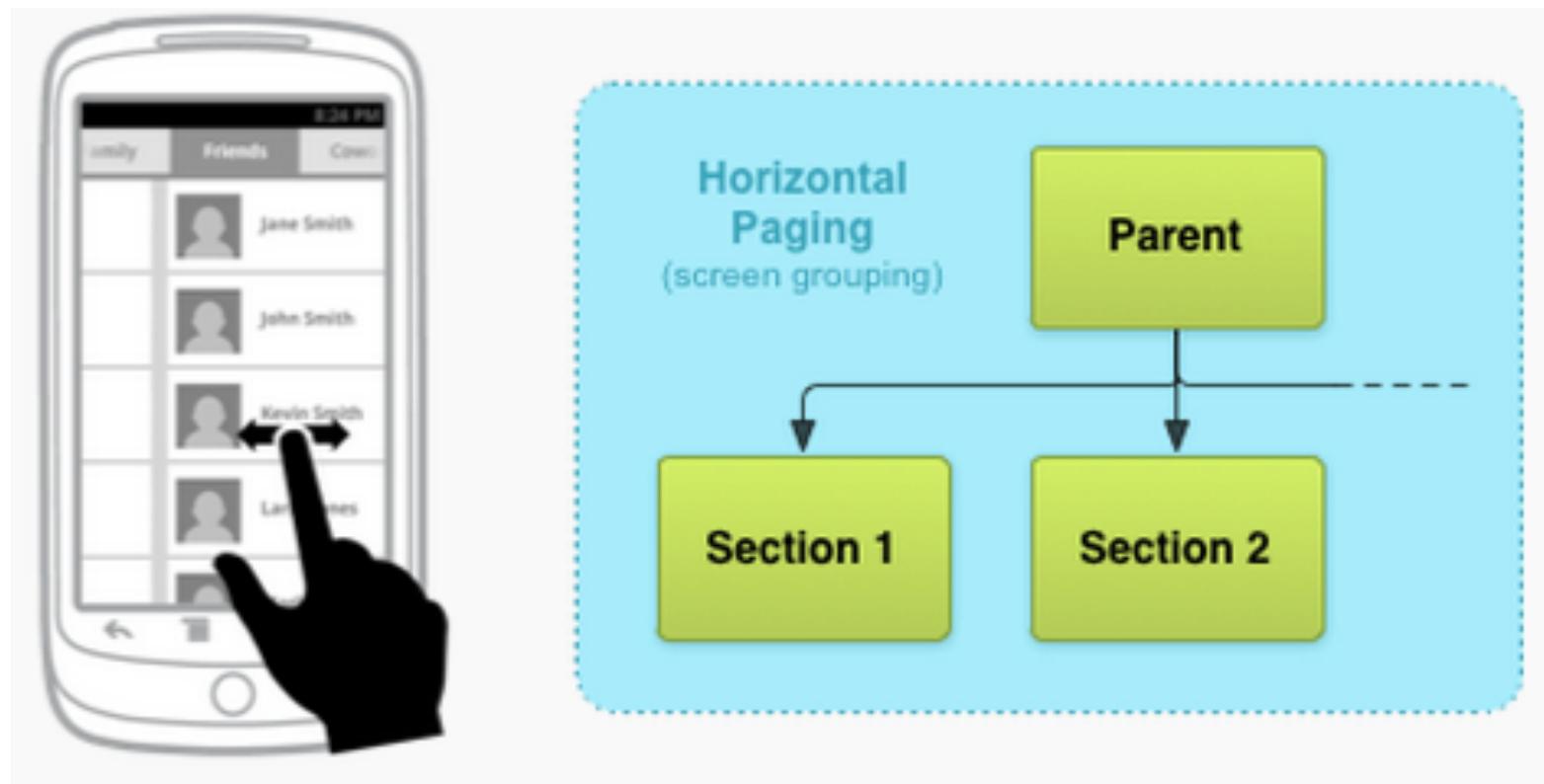
Pestañas.



Interfaz de usuario

Navegación Coherente.

Paginación Horizontal.



Interfaz de usuario

Navegación Coherente.

Navegación Temporal. Back (“Atrás”)

Los usuarios de Android esperan que el botón “Atrás” les lleve a la pantalla anterior.

El conjunto de pantallas siempre comienza en la pantalla "home" del teléfono.

Al presionar “Atrás” suficientes veces debemos llegar al HOME, después de lo cual el botón Atrás no hará nada.

Esta implementación es automática en los dispositivos Android.

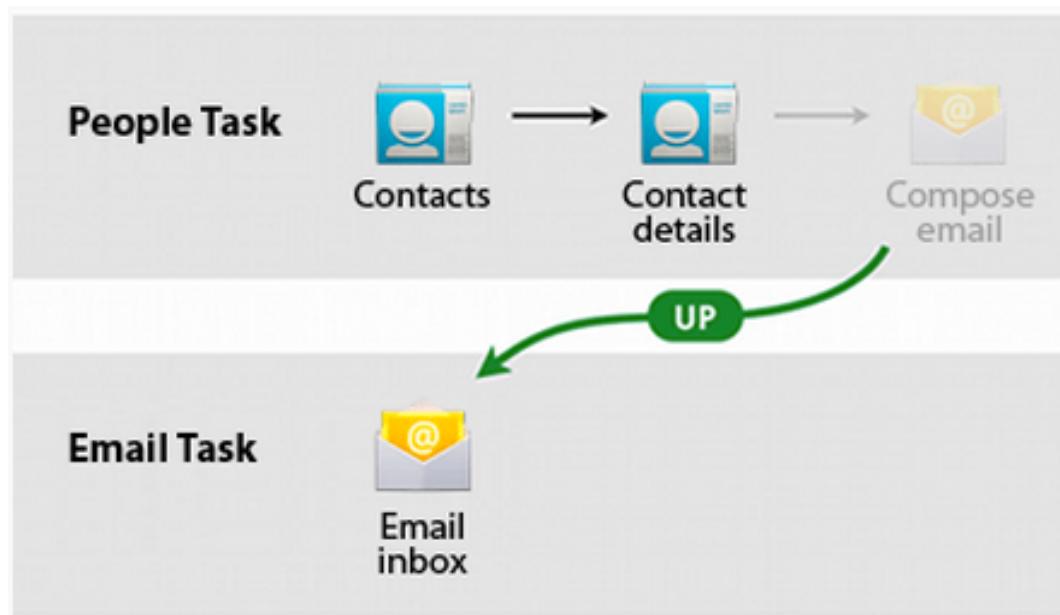


Interfaz de usuario

Navegación Coherente.

Navegación Jerárquica. HOME y UP

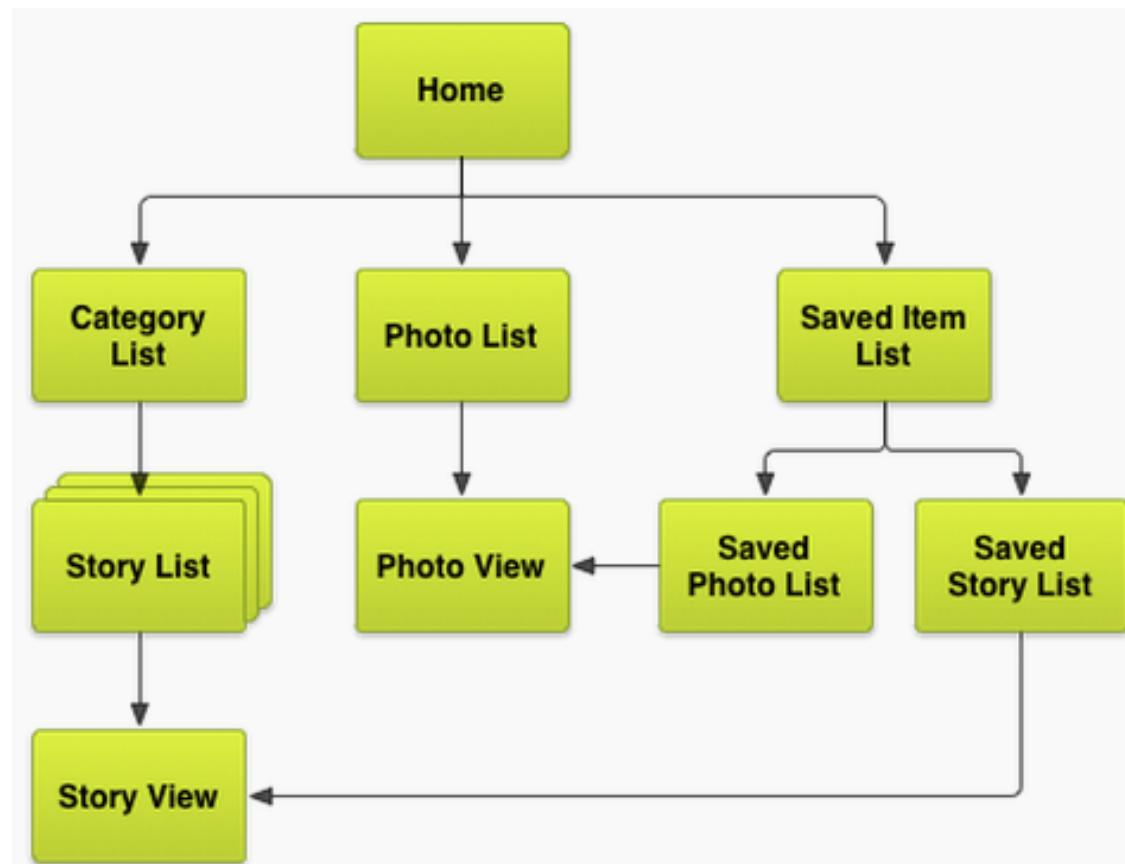
Los usuarios esperan que el botón Up le lleve al nivel anterior, acabando su funcionalidad en el HOME de nuestra aplicación.



Interfaz de usuario

Navegación Coherente.

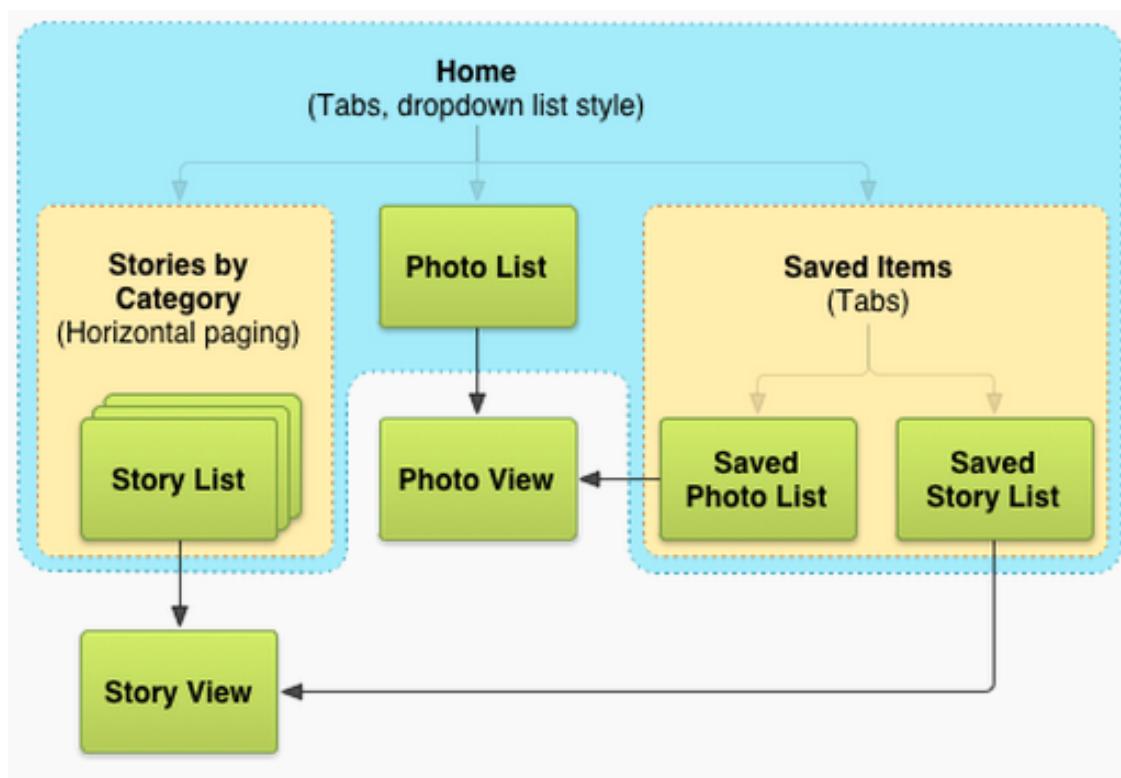
Ejemplo:



Interfaz de usuario

Navegación Coherente.

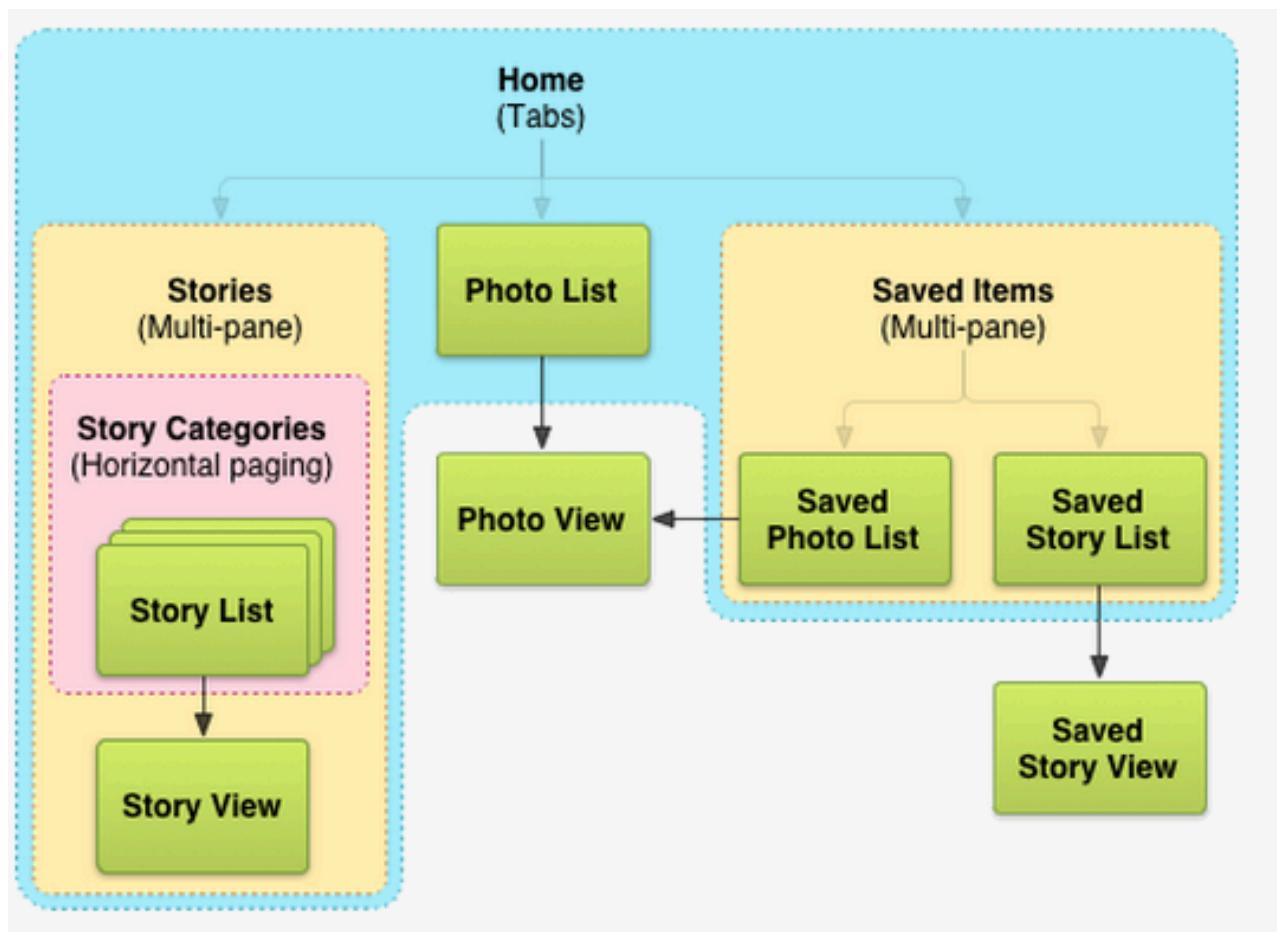
Teléfonos.



Interfaz de usuario

Navegación Coherente.

Tablets.



Navegación Coherente.

Resultado.



Interfaz de usuario

Navegación Coherente.

Resultado.



Interfaz de usuario

Navegación Coherente.

Swipe con pestañas.

Proporcionan una navegación lateral entre las pantallas del mismo nivel con pestañas.

Permite cambiar entre pestañas pulsando en estas o arrastrando el dedo.

Para implementar esta navegación usamos el componente ViewPager.

Para usar ViewPager agregamos un elemento <ViewPager> al diseño XML.

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.v4.view.ViewPager
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/pager"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />
```



Interfaz de usuario

Navegación Coherente.

Swipe con pestañas.

Para insertar vistas secundarias que representan a cada una de las páginas es necesario conectar el <ViewPager> con un adaptador del tipo PagerAdapter .

FragmentPagerAdapter

Este es el mejor cuando se navega entre las pantallas que representan un pequeño número fijo de páginas.

FragmentStatePagerAdapter

Este es el mejor para paginación a través de una colección de objetos para los que el número de páginas es indeterminado.

Destruye fragments cuando el usuario se desplaza a otras páginas, reduciendo al mínimo el uso de memoria.



Interfaz de usuario

Navegación Coherente.

Swipe con pestañas.

```
public class CollectionDemoActivity extends FragmentActivity {  
    // When requested, this adapter returns a DemoObjectFragment,  
    // representing an object in the collection.  
    DemoCollectionPagerAdapter mDemoCollectionPagerAdapter;  
    ViewPager mViewPager;  
  
    public void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_collection_demo);  
  
        // ViewPager and its adapters use support library  
        // fragments, so use getSupportFragmentManager.  
        mDemoCollectionPagerAdapter =  
            new DemoCollectionPagerAdapter(  
                getSupportFragmentManager());  
        mViewPager = (ViewPager) findViewById(R.id.pager);  
        mViewPager.setAdapter(mDemoCollectionPagerAdapter);  
    }  
}
```

Navegación C

Swipe con
pestañas.

```
// Since this is an object collection, use a FragmentStatePagerAdapter,  
// and NOT a FragmentPagerAdapter.  
  
public class DemoCollectionPagerAdapter extends FragmentStatePagerAdapter {  
    public DemoCollectionPagerAdapter(FragmentManager fm) {  
        super(fm);  
    }  
  
    @Override  
    public Fragment getItem(int i) {  
        Fragment fragment = new DemoObjectFragment();  
        Bundle args = new Bundle();  
        // Our object is just an integer :-P  
        args.putInt(DemoObjectFragment.ARG_OBJECT, i + 1);  
        fragment.setArguments(args);  
        return fragment;  
    }  
  
    @Override  
    public int getCount() {  
        return 100;  
    }  
  
    @Override  
    public CharSequence getPageTitle(int position) {  
        return "OBJECT " + (position + 1);  
    }  
}
```

Interfaz de usuario

Navegación Coherente.

Swipe con pestañas.

```
// Instances of this class are fragments representing a single
// object in our collection.
public static class DemoObjectFragment extends Fragment {
    public static final String ARG_OBJECT = "object";

    @Override
    public View onCreateView(LayoutInflater inflater,
                           ViewGroup container, Bundle savedInstanceState) {
        // The last two arguments ensure LayoutParams are inflated
        // properly.
        View rootView = inflater.inflate(
            R.layout.fragment_collection_object, container, false);
        Bundle args = getArguments();
        ((TextView) rootView.findViewById(android.R.id.text1)).setText(
            Integer.toString(args.getInt(ARG_OBJECT)));
        return rootView;
    }
}
```

Interfaz de usuario

Navegación Coherente.

Swipe con pestañas.

Para crear pestañas usando ActionBar, necesitamos habilitar NAVIGATION_MODE_TABS.

Proporcionar el callback ActionBar.TabListener

```
// Specify that tabs should be displayed in the action bar.  
actionBar.setNavigationMode(ActionBar.NAVIGATION_MODE_TABS);  
  
// Create a tab listener that is called when the user changes tabs.  
ActionBar.TabListener tabListener = new ActionBar.TabListener() {  
    public void onTabSelected(ActionBar.Tab tab, FragmentTransaction ft) {  
        // show the given tab  
    }  
  
    public void onTabUnselected(ActionBar.Tab tab, FragmentTransaction ft) {  
        // hide the given tab  
    }  
  
    public void onTabReselected(ActionBar.Tab tab, FragmentTransaction ft) {  
        // probably ignore this event  
    }  
};
```



Interfaz de usuario

Navegación Coherente.

Swipe con pestañas.

Crear tantas instancias de ActionBar.Tab como pestañas queramos.

```
// Add 3 tabs, specifying the tab's text and TabListener
for (int i = 0; i < 3; i++) {
    actionBar.addTab(
        actionBar.newTab()
            .setText("Tab " + (i + 1))
            .setTabListener(tabListener));
}
```



Interfaz de usuario

Navegación Coherente.

Swipe con pestañas.

Para pasar de página en un ViewPager cuando el usuario selecciona una pestaña, debemos implementar su ActionBar.TabListener para seleccionar la página apropiada llamando setCurrentItem() en su ViewPager.

```
// Create a tab listener that is called when the user changes tabs.  
ActionBar.TabListener tabListener = new ActionBar.TabListener() {  
    public void onTabSelected(ActionBar.Tab tab, FragmentTransaction ft) {  
        // When the tab is selected, switch to the  
        // corresponding page in the ViewPager.  
        mViewPager.setCurrentItem(tab.getPosition());  
    }  
    ...  
};
```



Interfaz de usuario

Navegación Coherente.

Swipe con pestañas.

Lo mismo pero al revés, al hacer un swipe debemos actualizar la pestaña seleccionada.

```
mViewPager = (ViewPager) findViewById(R.id.pager);
mViewPager.setOnPageChangeListener(
    new ViewPager.SimpleOnPageChangeListener() {
        @Override
        public void onPageSelected(int position) {
            // When swiping between pages, select the
            // corresponding tab.
            getActionBar().setSelectedNavigationItem(position);
        }
    });
});
```



Interfaz de usuario

Navegación Coherente.

Swipe con pestañas.

Si queremos incluir las pestañas en la barra de acciones y preferimos ofrecer pestañas desplazables dentro del <ViewPager> haremos uso de PagerTitleStrip.

```
<android.support.v4.view.ViewPager
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/pager"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <android.support.v4.view.PagerTitleStrip
        android:id="@+id/pager_title_strip"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_gravity="top"
        android:background="#33b5e5"
        android:textColor="#fff"
        android:paddingTop="4dp"
        android:paddingBottom="4dp" />

</android.support.v4.view.ViewPager>
```

```
@Override
public CharSequence getPageTitle(int position) {
    return "OBJECT " + (position + 1);
}
```

Interfaz de usuario

Navegación Coherente.

Navigation Drawer.

El “Navigation Drawer” es un panel que muestra las principales opciones de la aplicación a la izquierda de la pantalla.

Está oculto la mayor parte del tiempo, pero se abre cuando el usuario pasa el dedo desde el borde izquierdo de la pantalla o cuando el usuario toca el ícono de la aplicación en la barra de acción.

Para añadir un Navigation Drawer tenemos que declarar en la interfaz de usuario un objeto DrawerLayout como componente raíz.

Dentro del DrawerLayout añadimos un layout que contendrá la vista principal de la pantalla y otro para representar la pantalla del menú oculto.



Interfaz de usuario

Navegación Coherente.

Navigation Drawer.

```
<android.support.v4.widget.DrawerLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/drawer_layout"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <!-- The main content view -->
    <FrameLayout
        android:id="@+id/content_frame"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />
    <!-- The navigation drawer -->
    <ListView android:id="@+id/left_drawer"
        android:layout_width="240dp"
        android:layout_height="match_parent"
        android:layout_gravity="start"
        android:choiceMode="singleChoice"
        android:divider="@android:color/transparent"
        android:dividerHeight="0dp"
        android:background="#111" />
</android.support.v4.widget.DrawerLayout>
```

Interfaz de usuario

Navegación Coherente.

Navigation Drawer.

La vista principal de contenido debe ser el primer elemento en el DrawerLayout ya que el menú debe estar en la parte superior del contenido.

La vista de la página principal se ajusta para que coincida con la anchura y la altura de la vista del padre, ya que representa a toda la interfaz de usuario cuando se oculta el menú.

El menú especifica el ancho en DIPS y la altura coincide con la del elemento que la contiene.

La anchura del menú no debe superar los 320dp para que el usuario siempre puede ver una parte de la página principal.



Interfaz de usuario

Navegación Coherente.

Navigation Drawer.

Debemos inicializar el Menú del Navigation Drawer.

Depende del contenido de la aplicación, pero a menudo consiste en un ListView, por lo que la lista debe estar inicializada por un adaptador.

Debemos llamar al metodo setOnItemClickListener () para recibir eventos de la lista Navigation Drawer.



Interfaz de usuario

Navegación

Navegación

```
public class MainActivity extends Activity {  
    private String[] mPlanetTitles;  
    private ListView mDrawerList;  
    ...  
  
    @Override  
    public void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
  
        mPlanetTitles = getResources().getStringArray(R.array.planets_array);  
        mDrawerList = (ListView) findViewById(R.id.left_drawer);  
  
        // Set the adapter for the list view  
        mDrawerList.setAdapter(new ArrayAdapter<String>(this,  
            R.layout.drawer_list_item, mPlanetTitles));  
        // Set the list's click listener  
        mDrawerList.setOnItemClickListener(new DrawerItemClickListener());  
  
        ...  
    }  
}
```

Interfaz de usuario

Navegación Coherente.

Navigation Drawer.

Cuando el usuario selecciona un elemento en la lista, el sistema llama a `onItemClick()` en el `OnItemClickListener` dado a `setOnItemClickListener ()`.

```
private class DrawerItemClickListener implements ListView.OnItemClickListener {  
    @Override  
    public void onItemClick(AdapterView parent, View view, int position, long id) {  
        selectItem(position);  
    }  
}
```



Interfaz de usuario

Navegación Coherente.

Navigation Drawer.

```
/** Swaps fragments in the main content view */
private void selectItem(int position) {
    // Create a new fragment and specify the planet to show based on position
    Fragment fragment = new PlanetFragment();
    Bundle args = new Bundle();
    args.putInt(PlanetFragment.ARG_PLANET_NUMBER, position);
    fragment.setArguments(args);

    // Insert the fragment by replacing any existing fragment
    FragmentManager fragmentManager = getFragmentManager();
    fragmentManager.beginTransaction()
        .replace(R.id.content_frame, fragment)
        .commit();

    // Highlight the selected item, update the title, and close the drawer
    mDrawer.setItemChecked(position, true);
    setTitle(mPlanetTitles[position]);
    mDrawerLayout.closeDrawer(mDrawer);
}
```



Interfaz de usuario

Navegación Coherente.

Navigation Drawer.

Para escuchar los eventos de abrir y cerrar el menú, usamos el método `setDrawerListener()` del `DrawerLayout`.

Establecemos la implementación de la interfaz `DrawerLayout.DrawerListener`.

Esta interfaz proporciona las llamadas
`onDrawerOpened()`
`onDrawerClosed()`

En su lugar podemos extender la clase `ActionBarDrawerToggle`.

El `ActionBarDrawerToggle` implementa `DrawerLayout.DrawerListener` por lo que podemos gestionar la apertura y cierre del menú y además también facilita la integración entre el ícono de la barra de acciones y el menú de navegación.



Interfaz de usuario

Navegación Coherente.

Navigation Drawer.

Debemos modificar el contenido de la barra de acción cuando se muestra el menú.

Por ejemplo cambiar el título y quitar elementos de acción que dependen de la vista actual y no estarán en la nueva página.

```
/* Called whenever we call invalidateOptionsMenu() */
@Override
public boolean onPrepareOptionsMenu(Menu menu) {
    // If the nav drawer is open, hide action items related to the content view
    boolean drawerOpen = mDrawerLayout.isDrawerOpen(mDrawerList);
    menu.findItem(R.id.action_websearch).setVisible(!drawerOpen);
    return super.onPrepareOptionsMenu(menu);
}
```



Interfaz de usuario

```
mTitle = mDrawerTitle = getTitle();
mDrawerLayout = (DrawerLayout) findViewById(R.id.drawer_layout);
mDrawerToggle = new ActionBarDrawerToggle(this, mDrawerLayout,
    R.drawable.ic_drawer, R.string.drawer_open, R.string.drawer_close) {

    /** Called when a drawer has settled in a completely closed state. */
    public void onDrawerClosed(View view) {
        getActionBar().setTitle(mTitle);
        invalidateOptionsMenu(); // creates call to onPrepareOptionsMenu()
    }

    /** Called when a drawer has settled in a completely open state. */
    public void onDrawerOpened(View drawerView) {
        getActionBar().setTitle(mDrawerTitle);
        invalidateOptionsMenu(); // creates call to onPrepareOptionsMenu()
    }
};

// Set the drawer toggle as the DrawerListener
mDrawerLayout.setDrawerListener(mDrawerToggle);
```

Interfaz de usuario

Navegación Coherente.

Navigation Drawer.

Los usuarios pueden abrir y cerrar el menú con un gesto “swipe” LTR, pero también debemos permitir que los usuarios abran y se cierren al tocar el ícono de la aplicación.

Además el ícono de la aplicación también debe indicar la presencia del menú con un ícono especial.



Interfaz de usuario

Navegación Coherente.

Navigation Drawer.

Requiere los siguientes argumentos:

La actividad que contiene el menú.

El DrawerLayout.

Un ícono que hará de indicador del menú.

Un recurso de cadena para describir la acción de "menú abierto" (para la accesibilidad).

Un recurso de cadena para describir el "menú cerrado" acción (para la accesibilidad).



Interfaz de usuario

Navegación Coherente.

Navigation Drawer.

```
mDrawerLayout = (DrawerLayout) findViewById(R.id.drawer_layout);
mDrawerToggle = new ActionBarDrawerToggle(
    this,                      /* host Activity */
    mDrawerLayout,             /* DrawerLayout object */
    R.drawable.ic_drawer,      /* nav drawer icon to replace 'Up' caret */
    R.string.drawer_open,     /* "open drawer" description */
    R.string.drawer_close    /* "close drawer" description */
) {

    /** Called when a drawer has settled in a completely closed state. */
    public void onDrawerClosed(View view) {
        getActionBar().setTitle(mTitle);
    }

    /** Called when a drawer has settled in a completely open state. */
    public void onDrawerOpened(View drawerView) {
        getActionBar().setTitle(mDrawerTitle);
    }
};
```

Interfaz de usuario

Navegación Coherente.

Navigation Drawer.

```
// Set the drawer toggle as the DrawerListener  
mDrawerLayout.setDrawerListener(mDrawerToggle);  
  
getActionBar().setDisplayHomeAsUpEnabled(true);  
getActionBar().setHomeButtonEnabled(true);
```



Navegac

Navigation

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    // Sync the toggle state after onRestoreInstanceState has occurred.
    mDrawerToggle.syncState();
}

@Override
public void onConfigurationChanged(Configuration newConfig) {
    super.onConfigurationChanged(newConfig);
    mDrawerToggle.onConfigurationChanged(newConfig);
}

@Override
public boolean onOptionsItemSelected(MenuItem item) {
    // Pass the event to ActionBarDrawerToggle, if it returns
    // true, then it has handled the app icon touch event
    if (mDrawerToggle.onOptionsItemSelected(item)) {
        return true;
    }
    // Handle your other action bar items...

    return super.onOptionsItemSelected(item);
}
```

Interfaz de usuario

Navegación Coherente.

Botón UP.

Todas las pantallas en la aplicación que no sean la entrada principal de la aplicación (la pantalla "home") deben ofrecer al usuario una forma para ir a la pantalla principal pulsando el botón UP de la barra de acción.

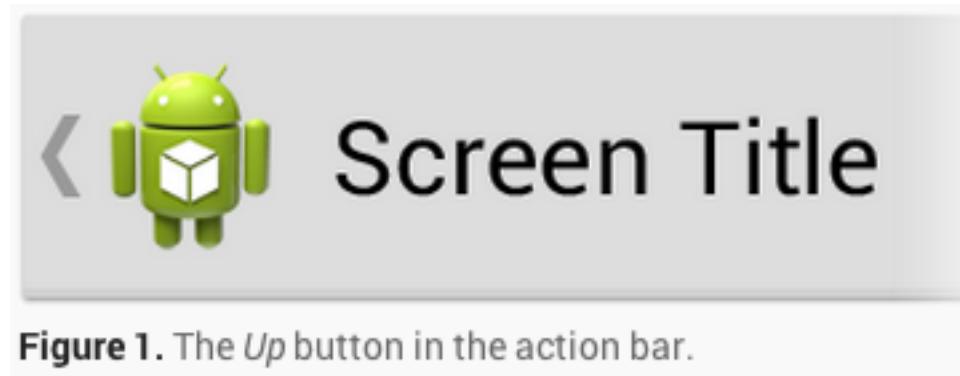


Figure 1. The Up button in the action bar.



Interfaz de usuario

Navegación Coherente.

Botón UP.

Para poner en práctica de navegación hacia arriba, el primer paso es declarar qué actividad es padre de cada actividad.

Esto permite que el sistema pueda determinar la actividad desde el archivo manifest.

```
<application ... >
    ...
    <!-- The main/home activity (it has no parent activity) -->
    <activity
        android:name="com.example.myfirstapp.MainActivity" ...>
        ...
    </activity>
    <!-- A child of the main activity -->
    <activity
        android:name="com.example.myfirstapp.DisplayMessageActivity"
        android:label="@string/title_activity_display_message"
        android:parentActivityName="com.example.myfirstapp.MainActivity" >
        <!-- Parent activity meta-data to support 4.0 and lower -->
        <meta-data
            android:name="android.support.PARENT_ACTIVITY"
            android:value="com.example.myfirstapp.MainActivity" />
    </activity>
</application>
```



Interfaz de usuario

Navegación Coherente.

Botón UP.

Para permitir la navegación usando el icono de la aplicación de la barra de acciones, establecemos la llamada `setDisplayHomeAsUpEnabled()`

Esto añade el símbolo < junto al icono de la aplicación y permite que actue como un botón de acción de tal manera que cuando el usuario lo presiona, su actividad recibe una llamada a `onOptionsItemSelected()`.

```
@Override  
public void onCreate(Bundle savedInstanceState) {  
    ...  
    getActionBar().setDisplayHomeAsUpEnabled(true);  
}
```



Interfaz de usuario

Navegación Coherente.

Botón UP.

El ID de la acción es android.R.id.home.

```
@Override  
public boolean onOptionsItemSelected(MenuItem item) {  
    switch (item.getItemId()) {  
        // Respond to the action bar's Up/Home button  
        case android.R.id.home:  
            NavUtils.navigateUpFromSameTask(this);  
            return true;  
    }  
    return super.onOptionsItemSelected(item);  
}
```



Interfaz de usuario

Navegación Coherente.

Botón UP.

Para desplazarnos hacia arriba cuando el usuario pulse el icono de la aplicación, utilizaremos el método estático de la clase NavUtils `navigateUpFromSameTask ()`.

Al llamar a este método, se termina la actividad actual e inicia (o reanuda) la actividad correspondiente.

```
@Override  
public boolean onOptionsItemSelected(MenuItem item) {  
    switch (item.getItemId()) {  
        // Respond to the action bar's Up/Home button  
        case android.R.id.home:  
            NavUtils.navigateUpFromSameTask(this);  
            return true;  
    }  
    return super.onOptionsItemSelected(item);  
}
```



Interfaz de usuario

Navegación Coherente.

Botón UP.

Si la actividad proporciona “Intent Filters” que permiten a otras aplicaciones o actividades iniciar la actividad, se deben implementar la función `onOptionsItemSelected ()` de forma que si el usuario pulsa el botón UP, la aplicación inicia una nueva tarea con el pila de retroceso apropiado antes de navegar hacia arriba.

Esta implementación requiere llamar a `shouldUpRecreateTask()` para comprobar si existe la instancia de actividad actual en la tarea de una aplicación diferente.

Si devuelve true, crearemos una nueva tarea con `TaskStackBuilder`.

Si devuelve false, podemos utilizar el `navigateUpFromSameTask()`.



Interfaz de usuario

```
@Override  
public boolean onOptionsItemSelected(MenuItem item) {  
    switch (item.getItemId()) {  
        // Respond to the action bar's Up/Home button  
        case android.R.id.home:  
            Intent upIntent = NavUtils.getParentActivityIntent(this);  
            if (NavUtils.shouldUpRecreateTask(this, upIntent)) {  
                // This activity is NOT part of this app's task, so create a new task  
                // when navigating up, with a synthesized back stack.  
                TaskStackBuilder.create(this)  
                    // Add all of this activity's parents to the back stack  
                    .addNextIntentWithParentStack(upIntent)  
                    // Navigate up to the closest parent  
                    .startActivities();  
            } else {  
                // This activity is part of this app's task, so simply  
                // navigate up to the logical parent activity.  
                NavUtils.navigateUpTo(this, upIntent);  
            }  
            return true;  
    }  
    return super.onOptionsItemSelected(item);  
}
```

Interfaz de usuario

Navegación Coherente.

Botón Back.

Es el mecanismo con el cual los usuarios se mueven hacia atrás en la secuencia de pantallas previamente visitadas.

Todos los dispositivos Android ofrecen un botón Atrás para este tipo de navegación, por lo que la aplicación no debe añadir un botón Volver a la interfaz de usuario. (Esto no es iOS!!!!)

En casi todas las situaciones, el sistema mantiene una pila de retroceso de las actividades, mientras el usuario se desplaza por la aplicación.

Esto permite que el sistema pueda navegar correctamente hacia atrás cuando el usuario pulsa el botón Atrás.

Hay algunos casos en que una aplicación debe especificar manualmente el comportamiento Back con el fin de proporcionar la mejor experiencia de usuario.



Interfaz de usuario

Navegación Coherente.

Botón Back.

Los Patrones de navegación que sugieren gestionar manualmente el comportamiento de Back son los que requieren:

1. Que el usuario entre en una actividad de nivel profundo directamente desde una notificación, un widget, o el Navigation Darwer.
2. Algunos casos en los que el usuario navega entre FRAGMENTS.
3. Cuando el usuario navega por las páginas web en un WebView.



Interfaz de usuario

Navegación Coherente.

Botón Back.

El sistema construye gradualmente la pila de retroceso cuando el usuario navega de una actividad a la siguiente.

Cuando el usuario entra en su aplicación desde un vínculo profundo que se inicia la actividad en su propia tarea.

Por ejemplo, cuando accedemos a una actividad anidada profundamente en la jerarquía de nuestra aplicación desde una notificación, debemos agregar las actividades necesarias para que al pulsar Back naveguemos por la jerarquía de aplicación en lugar de salir de la aplicación.



Interfaz de usuario

Navegación Coherente.

Botón Back.

```
// Intent for the activity to open when user selects the notification
Intent detailsIntent = new Intent(this, DetailsActivity.class);

// Use TaskStackBuilder to build the back stack and get the PendingIntent
PendingIntent pendingIntent =
    TaskStackBuilder.create(this)
        // add all of DetailsActivity's parents to the stack,
        // followed by DetailsActivity itself
        .addNextIntentWithParentStack(upIntent)
        .getPendingIntent(0, PendingIntent.FLAG_UPDATE_CURRENT);

NotificationCompat.Builder builder = new NotificationCompat.Builder(this);
builder.setContentIntent(pendingIntent);
```



Interfaz de usuario

Navegación Coherente.

Botón Back.

Al utilizar fragments en la aplicación, los objetos FragmentTransaction individuales pueden representar los cambios de contexto que se deben agregar a la pila de retroceso.

Si vamos a implementar un flujo de maestro / detalle de un teléfono mediante el cambio de fragments, debemos asegurarnos que al pulsar el botón Atrás en la pantalla de detalle, el usuario vuelve a la pantalla principal.

Para ello, usamos addToBackStack () antes de confirmar la transacción.



Interfaz de usuario

Navegación Coherente.

Botón Back.

```
// Works with either the framework FragmentManager or the
// support package FragmentManager (getSupportFragmentManager).
getSupportFragmentManager().beginTransaction()
    .add(detailFragment, "detail")
    // Add this transaction to the back stack
    .addToBackStack()
    .commit();
```

No debemos agregar las transacciones a la pila de retroceso cuando la transacción es para la navegación horizontal (por ejemplo, al cambiar de pestañas) o al modificar el aspecto de contenido (por ejemplo, cuando aplicamos filtros).



Interfaz de usuario

Navegación Coherente.

Botón Back.

Si la aplicación actualiza otros elementos de la interfaz de usuario para reflejar el estado actual de los Fragments, como la barra de acción, habrá que actualizar la interfaz de usuario al confirmar la transacción.

Las transacciones son notificadas en
FragmentManager.OnBackStackChangedListener:

```
getSupportFragmentManager().addOnBackStackChangedListener(  
    new FragmentManager.OnBackStackChangedListener() {  
        public void onBackStackChanged() {  
            // Update your UI here.  
        }  
    });
```



Interfaz de usuario

Navegación Coherente.

Botón Back.

Si implementamos un WebView en nuestra aplicación deberemos modificar el comportamiento de Back para permitir navegar por el historial del componente WebView.

```
@Override  
public void onBackPressed() {  
    if (mWebView.canGoBack()) {  
        mWebView.goBack();  
        return;  
    }  
  
    // Otherwise defer to system default behavior.  
    super.onBackPressed();  
}
```



Interfaz de usuario

Navegación Coherente.

Botón Back.

Al poner en marcha una actividad de otra aplicación para que el usuario pueda por ejemplo componer un correo electrónico o elegir un contacto, no queremos que el usuario vuelva a esta actividad si relanza la aplicación.

Sería confuso si al tocar el icono de la aplicación apareciese la pantalla de "componer correo electrónico" o "Seleccionar Contacto"

```
Intent externalActivityIntent = new Intent(Intent.ACTION_PICK);
externalActivityIntent.setType("image/*");
externalActivityIntent.addFlags(
    Intent.FLAG_ACTIVITY_CLEAR_WHEN_TASK_RESET);
startActivity(externalActivityIntent);
```



Interfaz de usuario

Notificaciones.

Notification Builder.

NotificationCompat.Builder se encuentra en la biblioteca de compatibilidad.

Debemos utilizar NotificationCompat.Builder para asegurar la compatibilidad entre plataformas.

Cuando se crea una notificación, hay que especificar el contenido de la interfaz de usuario y las acciones con un objeto NotificationCompat.Builder.

Un pequeño ícono, establecido por setSmallIcon()

Un título, establecido por setContentTitle()

El texto de detalle, fijado por setContentText()

```
NotificationCompat.Builder mBuilder =  
    new NotificationCompat.Builder(this)  
        .setSmallIcon(R.drawable.notification_icon)  
        .setContentTitle("My notification")  
        .setContentText("Hello World!");
```



Interfaz de usuario

Notificaciones.

Notification Builder.

Aunque las acciones son opcionales, debemos establecer al menos una acción a la notificación.

La acción lleva a los usuarios directamente desde la notificación a una actividad de la aplicación.

La acción se define con un PendingIntent que contiene un Intent que lanzará una actividad de la aplicación.

```
Intent resultIntent = new Intent(this, ResultActivity.class);
...
// Because clicking the notification opens a new ("special") activity, there's
// no need to create an artificial back stack.
PendingIntent resultPendingIntent =
    PendingIntent.getActivity(
        this,
        0,
        resultIntent,
        PendingIntent.FLAG_UPDATE_CURRENT
);
```



Interfaz de usuario

Notificaciones.

Notification Builder.

Para asociar el PendingIntent llamamos al método setContentIntent() de NotificationCompat.Builder.

Por ejemplo, para iniciar una actividad cuando el usuario hace clic en el texto de la notificación, añadimos el PendingIntent llamando a setContentIntent().

```
PendingIntent resultPendingIntent;  
...  
mBuilder.setContentIntent(resultPendingIntent);
```



Interfaz de usuario

Notificaciones.

Notification Builder.

Para lanzar la notificación tenemos que:

Obtener una instancia de NotificationManager.

Usar el método notify() para enviar la notificación.

Cuando se llama a notify(), especificaremos un ID de notificación.

Utilizaremos este ID para actualizar la notificación.

El método build() devuelve la notificación que contiene las especificaciones usadas.



Interfaz de usuario

Notificaciones.

Notification Builder.

```
NotificationCompat.Builder mBuilder;  
...  
// Sets an ID for the notification  
int mNotificationId = 001;  
// Gets an instance of the NotificationManager service  
NotificationManager mNotifyMgr =  
    (NotificationManager) getSystemService(NOTIFICATION_SERVICE);  
// Builds the notification and issues it.  
mNotifyMgr.notify(mNotificationId, mBuilder.build());
```



Interfaz de usuario

Notificaciones.

Notification Builder.

El diseño de una notificación está pensado en mantener la experiencia de navegación que espera del usuario.

Hay dos situaciones generales:

La actividad regular

Estás empezando una actividad que es parte del flujo normal de trabajo de la aplicación.

La actividad especial

El usuario sólo ve esta actividad si se inició a partir de la notificación.

En cierto sentido, la actividad extiende a la notificación ya que sería difícil de mostrar toda la información en una notificación.



Interfaz de usuario

Notificaciones.

Notification Builder.

Iniciar una actividad de forma regular.

```
<activity
    android:name=".MainActivity"
    android:label="@string/app_name" >
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
<activity
    android:name=".ResultActivity"
    android:parentActivityName=".MainActivity">
    <meta-data
        android:name="android.support.PARENT_ACTIVITY"
        android:value=".MainActivity"/>
</activity>
```



Interfaz de usuario

Notificaciones.

Notification Builder.

Iniciar una actividad de forma regular.

```
Intent resultIntent = new Intent(this, ResultActivity.class);
TaskStackBuilder stackBuilder = TaskStackBuilder.create(this);
// Adds the back stack
stackBuilder.addParentStack(ResultActivity.class);
// Adds the Intent to the top of the stack
stackBuilder.addNextIntent(resultIntent);
// Gets a PendingIntent containing the entire back stack
PendingIntent resultPendingIntent =
    stackBuilder.getPendingIntent(0, PendingIntent.FLAG_UPDATE_CURRENT);
...
NotificationCompat.Builder builder = new NotificationCompat.Builder(this);
builder.setContentIntent(resultPendingIntent);
NotificationManager mNotificationManager =
    (NotificationManager) getSystemService(Context.NOTIFICATION_SERVICE);
mNotificationManager.notify(id, builder.build());
```

Interfaz de usuario

Notificaciones.

Notification Builder.

Iniciar una actividad de forma especial.

Una actividad especial no necesita una pila de retroceso, por lo que no tendrá que definir su jerarquía de Actividad en el manifest.

No necesita tener que llamar a addParentStack () para construir una pila de retroceso.

Utilizamos el manifest para configurar las opciones de la tarea de la actividad, y crear un PendingIntent llamando getActivity()

```
<activity
    android:name=".ResultActivity"
    ...
    android:launchMode="singleTask"
    android:taskAffinity=""
    android:excludeFromRecents="true">
</activity>
```



Interfaz de usuario

Notificaciones.

Notification Builder.

Iniciar una actividad de forma especial.

Construir y lanzar la notificación:

Crear un Intent que inicia la actividad.

Ajustar la actividad para iniciar una nueva tarea, estableciendo setFlags () con FLAG_ACTIVITY_NEW_TASK y FLAG_ACTIVITY_CLEAR_TASK.

Crear un PendingIntent del Intent llamando getActivity () .

Usar el PendingIntent como argumento para setContentIntent () .



Interfaz de usuario

Notificaciones.

Notification Builder.

Iniciar una actividad de forma especial.

```
// Instantiate a Builder object.  
NotificationCompat.Builder builder = new NotificationCompat.Builder(this);  
// Creates an Intent for the Activity  
Intent notifyIntent =  
    new Intent(new ComponentName(this, ResultActivity.class));  
// Sets the Activity to start in a new, empty task  
notifyIntent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK |  
    Intent.FLAG_ACTIVITY_CLEAR_TASK);  
// Creates the PendingIntent  
PendingIntent notifyIntent =  
    PendingIntent.getActivity(  
        this,  
        0,  
        notifyIntent,  
        PendingIntent.FLAG_UPDATE_CURRENT  
    );
```



Interfaz de usuario

Notificaciones.

Notification Builder.

Iniciar una actividad de forma especial.

```
// Puts the PendingIntent into the notification builder
builder.setContentIntent(notifyIntent);
// Notifications are issued by sending them to the
// NotificationManager system service.
NotificationManager mNotificationManager =
    (NotificationManager) getSystemService(Context.NOTIFICATION_SERVICE);
// Builds an anonymous Notification object from the builder, and
// passes it to the NotificationManager
mNotificationManager.notify(id, builder.build());
```



Interfaz de usuario

Notificaciones.

Notification Builder.

Actualizar una notificación.

Usar el mismo ID que la vez que lanzamos la notificación.

```
mNotificationManager =
        (NotificationManager) getSystemService(Context.NOTIFICATION_SERVICE);
// Sets an ID for the notification, so it can be updated
int notifyID = 1;
mNotifyBuilder = new NotificationCompat.Builder(this)
    .setContentTitle("New Message")
    .setContentText("You've received new messages.")
    .setSmallIcon(R.drawable.ic_notify_status)
numMessages = 0;
// Start of a loop that processes data and then notifies the user
...
    mNotifyBuilder.setContentText(currentText)
        .setNumber(++numMessages);
// Because the ID remains unchanged, the existing notification is
// updated.
mNotificationManager.notify(
    notifyID,
    mNotifyBuilder.build());
...
```



Interfaz de usuario

Notificaciones.

Notification Builder.

Eliminar una notificación.

Las Notificaciones permanecen visibles hasta que ocurre uno de los casos siguientes:

El usuario cierra la notificación de forma individual o mediante el uso de "Clear All" (si la notificación se puede borrar).

El usuario toca la notificación y se estableció setAutoCancel() al crear la notificación.

Llamando a cancel() para un ID específico de una notificación.

Este método también elimina las notificaciones en curso.

Llamando a cancelAll() que elimina todos los avisos que se emitieron con anterioridad.



Interfaz de usuario

Notificaciones.

Notification Builder.

Las notificaciones tienen dos estilos, vista normal, y vista extendida.

La vista extendida de una notificación sólo aparece cuando la notificación se expande.

Esto sucede cuando la notificación es la primera de la lista de notificaciones o si el usuario hace clic en la notificación.

Las vistas expandidas fueron introducidas en Android 4.1 y no están soportadas en los dispositivos más antiguos.

En dispositivos antiguos se mantiene el formato viejo si usamos el compatibility pack.



Interfaz de usuario

Notificaciones.

Notification Builder.

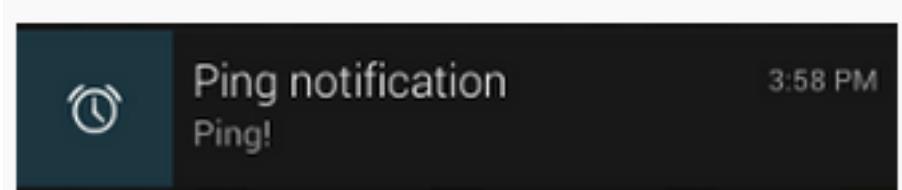


Figure 1. Normal view notification.

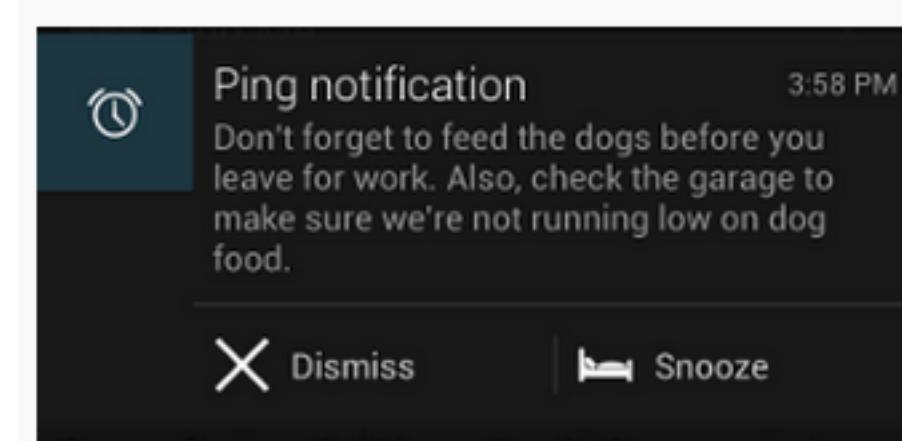


Figure 2. Big view notification.



Interfaz de usuario

Notificaciones.

Notification Builder.

```
// Sets up the Snooze and Dismiss action buttons that will appear in the  
// big view of the notification.  
  
Intent dismissIntent = new Intent(this, PingService.class);  
dismissIntent.setAction(CommonConstants.ACTION_DISMISS);  
PendingIntent piDismiss = PendingIntent.getService(this, 0, dismissIntent, 0);  
  
Intent snoozeIntent = new Intent(this, PingService.class);  
snoozeIntent.setAction(CommonConstants.ACTION_SNOOZE);  
PendingIntent piSnooze = PendingIntent.getService(this, 0, snoozeIntent, 0);
```



Interfaz de usuario

Notificaciones.

Notification Builder.

```
// Constructs the Builder object.
NotificationCompat.Builder builder =
    new NotificationCompat.Builder(this)
    .setSmallIcon(R.drawable.ic_stat_notification)
    .setContentTitle(getString(R.string.notification))
    .setContentText(getString(R.string.ping))
    .setDefaults(Notification.DEFAULT_ALL) // requires VIBRATE permission
/*
 * Sets the big view "big text" style and supplies the
 * text (the user's reminder message) that will be displayed
 * in the detail area of the expanded notification.
 * These calls are ignored by the support library for
 * pre-4.1 devices.
 */
.setStyle(new NotificationCompat.BigTextStyle()
    .bigText(msg))
.addAction (R.drawable.ic_stat_dismiss,
    getString(R.string.dismiss), piDismiss)
.addAction (R.drawable.ic_stat_snooze,
    getString(R.string.snooze), piSnooze);
```



Interfaz de usuario

Notificaciones.

Notification Builder.

Las notificaciones pueden incluir un indicador de progreso animado que muestra a los usuarios el estado de una operación en curso.

Si se puede estimar el tiempo que dura la operación y cuánto está completado en cualquier momento utilizaremos la forma "determinada" del indicador (una barra de progreso).

Si no se puede estimar la duración de la operación, utilizaremos el la forma "indeterminada" del indicador (un indicador de actividad).

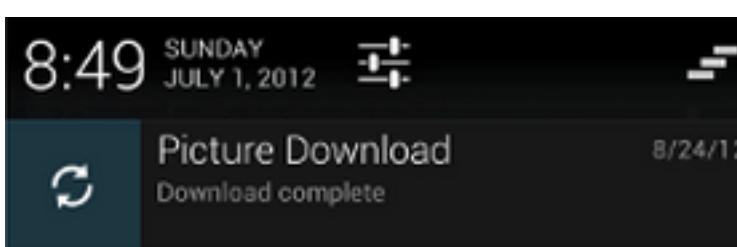
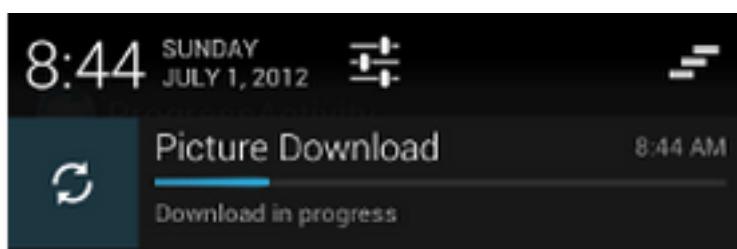
Para mostrar una barra de progreso determinada, añadiremos la barra a la notificación llamando a `setProgress (max, progreso, ¿indeterminada?)` y luego lanzaremos la notificación.

El tercer argumento es un booleano que indica si la barra de progreso es indeterminada (true) o determinada (false).



Notificaciones.

Notification Builder.



```
mNotifyManager =
    (NotificationManager) getSystemService(Context.NOTIFICATION_SERVICE);
mBuilder = new NotificationCompat.Builder(this);
mBuilder.setContentTitle("Picture Download")
    .setContentText("Download in progress")
    .setSmallIcon(R.drawable.ic_notification);
// Start a lengthy operation in a background thread
new Thread(
    new Runnable() {
        @Override
        public void run() {
            int incr;
            // Do the "lengthy" operation 20 times
            for (incr = 0; incr <= 100; incr+=5) {
                // Sets the progress indicator to a max value, the
                // current completion percentage, and "determinate"
                // state
                mBuilder.setProgress(100, incr, false);
                // Displays the progress bar for the first time.
                mNotifyManager.notify(0, mBuilder.build());
                // Sleeps the thread, simulating an operation
                // that takes time
                try {
                    // Sleep for 5 seconds
                    Thread.sleep(5*1000);
                } catch (InterruptedException e) {
                    Log.d(TAG, "sleep failure");
                }
            }
            // When the loop is finished, updates the notification
            mBuilder.setContentText("Download complete")
            // Removes the progress bar
            .setProgress(0,0,false);
            mNotifyManager.notify(ID, mBuilder.build());
        }
    }
)
// Starts the thread by calling the run() method in its Runnable
.start();
```

Interfaz de usuario

Notificaciones.

Notification Builder.

Para mostrar un indicador de actividad continuada (indeterminado), añadiremos a la notificación setProgress(0, 0, true) y lanzaremos la notificación.

Se omiten los dos primeros argumentos, y el tercer argumento declara que el indicador es indeterminado.

El resultado es un indicador que tiene el mismo estilo que la barra de progreso, con la excepción de que la animación está en curso.

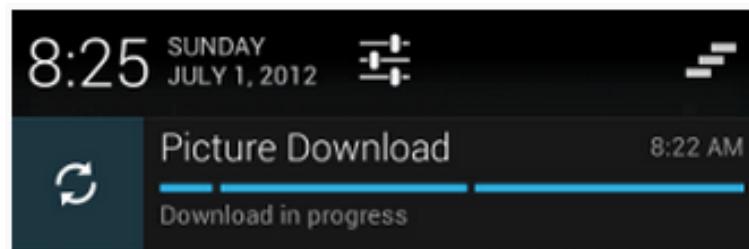


Figure 2. An ongoing activity indicator.



Interfaz de usuario

Notificaciones.

Notification Builder.

```
// Sets an activity indicator for an operation of indeterminate length  
mBuilder.setProgress(0, 0, true);  
// Issues the notification  
mNotifyManager.notify(0, mBuilder.build());
```



Interfaz de usuario

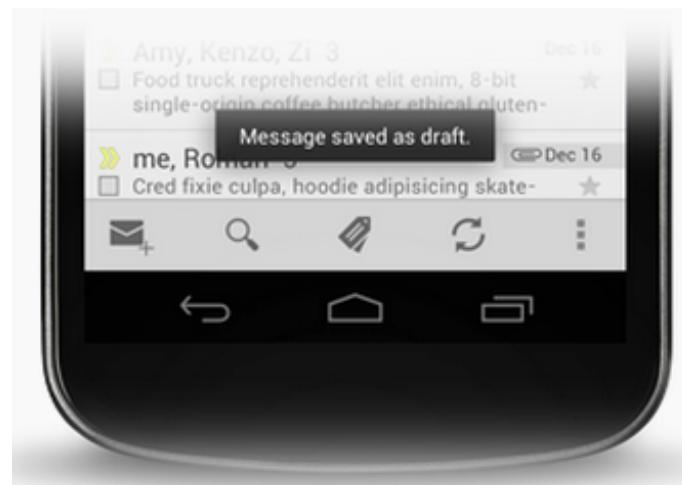
Notificaciones.

Toast.

Un TOAST proporciona información sencilla acerca de una operación en una pequeña ventana emergente.

Utiliza sólo la cantidad de espacio requerido para el mensaje y la actividad actual permanece visible.

Desaparecen automáticamente después de un tiempo de espera.
(Long/Short)



Interfaz de usuario

Notificaciones.

Toast.

Creamos instancias de un objeto TOAST maketext().

Este método tiene tres parámetros: el contexto de aplicación, el mensaje de texto, y su duración.

Devuelve un objeto TOAST inicializado correctamente, prearado para mostrarlo con show().

```
Context context = getApplicationContext();
CharSequence text = "Hello toast!";
int duration = Toast.LENGTH_SHORT;

Toast toast = Toast.makeText(context, text, duration);
toast.show();
```



Interfaz de usuario

Notificaciones.

Toast.

Si un simple mensaje de texto no es suficiente, podemos crear un diseño personalizado.

Para crear un diseño personalizado, definimos un diseño para la notificación y lo establecemos con su método `setView(View view)`.

```
LayoutInflater inflater = getLayoutInflater();
View layout = inflater.inflate(R.layout.custom_toast,
                               (ViewGroup) findViewById(R.id.toast_layout_root));

TextView text = (TextView) layout.findViewById(R.id.text);
text.setText("This is a custom toast");

Toast toast = new Toast(getApplicationContext());
toast.setGravity(Gravity.CENTER_VERTICAL, 0, 0);
toast.setDuration(Toast.LENGTH_LONG);
toast.setView(layout);
toast.show();
```

Interfaz de usuario

Otras Interfaces. GTV

Android nos permite llevar nuestras aplicaciones favoritas a la mejor pantalla de la casa.

Hay miles de aplicaciones disponibles en la tienda Play Google que ya están optimizados para los televisores.

Hay dos vertientes muy diferenciadas a la hora de desarrollar para GTV, las aplicaciones nativas y las Webapps

Las webapps consisten en aplicaciones web optimizadas para poder ser usadas en televisiones.

Muchas webapps tienen una parte nativa que hace como lanzador de la aplicación web además de permitirnos injectar cierta información a esta.



Interfaz de usuario

Otras Interfaces. GTV

Cuando la aplicación se ejecuta en una TV, se debe asumir que el usuario está sentado a unos tres metros de distancia de la pantalla.

Debemos proporcionar recursos de diseño adecuados para el modo Landscape.

Asegurarnos de que el texto y los controles son lo suficientemente grandes como para ser visibles desde una distancia grande.

Proporcionar mapas de bits e iconos para pantallas de televisión de alta definición y de alta resolución.



Interfaz de usuario

Otras Interfaces. GTV

Debemos poner los controles de navegación en la pantalla al lado izquierdo o derecho de la pantalla y guardar el espacio vertical para el contenido.

Crear interfaces de usuario que se dividan en secciones, mediante el uso de fragments y utilizar vistas como GridView evitando ListView para hacer un mejor uso del espacio horizontal.

Debemos usar RelativeLayout o LinearLayout para organizar las visitas.

Añadir un margen suficiente entre los controles para evitar una interfaz desordenada.



Interfaz de usuario

Otras Interfaces. GTV

El texto y los controles de la interfaz de usuario de una aplicación de TV deben ser fácilmente visibles y navegables desde larga distancia.

Dividiremos el texto en trozos pequeños que los usuarios pueden distinguir rápidamente.

Usaremos texto claro sobre un fondo oscuro. Este estilo es más fácil de leer en un televisor.

Es preferible usar las fuentes sans-serif simples y usar anti-aliasing para aumentar la legibilidad.

Utilizaremos esquemas de tamaño relativo y mediremos las dimensiones en “dips” y los tamaños de texto en “sp”



Interfaz de usuario

Otras Interfaces. GTV

Las resoluciones comunes de TV son 720p, 1080i y 1080p.

Diseñaremos la interfaz de usuario en 1080p y luego permitiremos que el sistema escale a 720p si es necesario.

En general, la reducción de escala (eliminación de píxeles) no degrada la interfaz de .

Para obtener los mejores resultados de escala para las imágenes, usaremos 9-patch.

Cargaremos imágenes sólo cuando van a ser mostradas en la pantalla.

Debemos reciclar los bitmaps para optimizar la memoria.

Utilizaremos WeakReference para almacenar BMPs en una colección en memoria.



Interfaz de usuario

Otras Interfaces. GTV

Un aspecto importante del GTV es el mando a distancia.

Debemos prestar especial atención a la forma en que el usuario realmente navega alrededor de la aplicación al utilizar un control remoto en lugar de una pantalla táctil.

Debemos asegurarnos de que todos los controles de diseño son navegables mediante el D-pad.

Debes proporcionar la información de navegación por la interfaz de usuario por muy obvia que parezca, el mando a distancia es un control oculto en la pantalla.

Buscar una colocación de los controles para facilitar el acceso a ellos.



Interfaz de usuario

Otras Interfaces. GTV

```
<EditText android:id="@+id/LastNameField" android:nextFocusDown="@+id/FirstNameField"\>
```

Attribute	Function
nextFocusDown	Defines the next view to receive focus when the user navigates down.
nextFocusLeft	Defines the next view to receive focus when the user navigates left.
nextFocusRight	Defines the next view to receive focus when the user navigates right.
nextFocusUp	Defines the next view to receive focus when the user navigates up.



Interfaz de usuario

Otras Interfaces. GTV

Usaremos detalles de color adecuados para los elementos seleccionables en la interfaz de usuario.

res/drawable/button.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<selector xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:state_pressed="true"
          android:drawable="@drawable/button_pressed" /> <!-- pressed -->
    <item android:state_focused="true"
          android:drawable="@drawable/button_focused" /> <!-- focused -->
    <item android:state_hovered="true"
          android:drawable="@drawable/button_focused" /> <!-- hovered -->
    <item android:drawable="@drawable/button_normal" /> <!-- default -->
</selector>        <Button
                      android:layout_height="wrap_content"
                      android:layout_width="wrap_content"
                      android:background="@drawable/button" />
```



Interfaz de usuario

Otras Interfaces. GTV

Los televisores son muy diferentes de otros dispositivos Android:

No son móviles.

Los usuarios los usan para consumir contenidos.

Los usuarios interactúan con ellos desde la distancia.

Carecen de algunas características:

Hardware	Android feature descriptor
Camera	android.hardware.camera
GPS	android.hardware.location.gps
Microphone	android.hardware.microphone
Near Field Communications (NFC)	android.hardware.nfc
Telephony	android.hardware.telephony
Touchscreen	android.hardware.touchscreen

Interfaz de usuario

Otras Interfaces. GTV

Pero no todas carecen de los mismo...

Debemos comprobar el HW del que disponemos...

```
// Check if android.hardware.telephony feature is available.  
if (getPackageManager().hasSystemFeature("android.hardware.telephony")) {  
    Log.d("Mobile Test", "Running on phone");  
// Check if android.hardware.touchscreen feature is available.  
} else if (getPackageManager().hasSystemFeature("android.hardware.touchscreen")) {  
    Log.d("Tablet Test", "Running on devices that don't support telephony but have a touchscreen");  
} else {  
    Log.d("TV Test", "Running on a TV!");  
}
```



Interfaz de usuario

Otras Interfaces. GTV

Disponemos de ejemplos de aplicaciones optimizadas para GTV y algunos códigos para la integración entre dispositivos móviles y GTV en:

<https://developers.google.com/tv/>

<https://code.google.com/p/googletv-android-samples/>

- ▶ [AnymoteLibrary](#)
- ▶ [BlackJackGTV](#)
- ▶ [BlackJackTVRemote](#)
- ▶ [ChannelChangingSample](#)
- ▶ [LeftNavBarDemo](#)
- ▶ [LeftNavBarLibrary](#)
- ▶ [MapsOnTV](#)
- ▶ [Notifier](#)
- ▶ [Panoramio](#)
- ▶ [USBHostProber](#)
- ▶ [WebAppNativePlayback](#)
- ▶ [aspect-ratio-video-library](#)



Interfaz de usuario

Otras Interfaces. GTV

Hibridas (Nativa/WEB)

- Permiso de Internet

```
<manifest ... >
    <uses-permission android:name="android.permission.INTERNET" />
    ...
</manifest>
```

- Habilitar JavaScript

```
WebView myWebView = (WebView) findViewById(R.id.webview);
WebSettings webSettings = myWebView.getSettings();
webSettings.setJavaScriptEnabled(true);
```



Otras Interfaces. GTV

Hibridas (Nativa/WEB)

- Enlazar JavaScript y Java
 - Llamar a código «Nativo» desde JavaScript
 - JavaScriptInterface

```
public class JavaScriptInterface {  
    Context mContext;  
  
    /** Instantiate the interface and set the context */  
    JavaScriptInterface(Context c) {  
        mContext = c;  
    }  
  
    /** Show a toast from the web page */  
    public void showToast(String toast) {  
        Toast.makeText(mContext, toast, Toast.LENGTH_SHORT).show();  
    }  
}
```



Interfaz de usuario

Otras Interfaces. GTV

Hibridas (Nativa/WEB)

- Enlazar JavaScript y Java
 - WebView.addJavaScriptInterface()
 - JavaScriptInterface
 - Nombre de la interfaz
 - Se transforma en un objeto JS

```
WebView webView = (WebView) findViewById(R.id.webview);
webView.addJavascriptInterface(new JavaScriptInterface(this), "Android");
```



Interfaz de usuario

Otras Interfaces. GTV

Hibridas (Nativa/WEB)

- Enlazar JavaScript y Java
 - Llamadas desde la interfaz

```
<input type="button" value="Say hello" onClick="showAndroidToast('Hello Android!')"/>

<script type="text/javascript">
    function showAndroidToast(toast) {
        Android.showToast(toast);
    }
</script>
```

- El código JS se ejecuta en otro hilo
- Habilita el uso de JS de forma automática
- Ejecutar código de terceros es potencialmente peligroso



Interfaz de usuario

Otras Interfaces. GTV

Hibridas (Nativa/WEB)

- Manejar la navegación
 - Implementar WebClient
 - Reaccionar a ciertas URL
 - Añadir parámetros a URLs
 - Redireccionar según la URL

```
WebView myWebView = (WebView) findViewById(R.id.webview);
myWebView.setWebViewClient(new WebViewClient());
```



Otras Interfaces. GTV

Hibridas (Nativa/WEB)

- Manejar la navegación
- shouldOverrideUrlLoading()
 - True El webview sigue gestionando la url
 - False Nosotros nos hacemos cargo de la url

```
private class MyWebViewClient extends WebViewClient {  
    @Override  
    public boolean shouldOverrideUrlLoading(WebView view, String url) {  
        if (Uri.parse(url).getHost().equals("www.example.com")) {  
            // This is my web site, so do not override; let my WebView  
            return false;  
        }  
        // Otherwise, the link is not for a page on my site, so launch  
        Intent intent = new Intent(Intent.ACTION_VIEW, Uri.parse(url));  
        startActivity(intent);  
        return true;  
    }  
}
```



Otras Interfaces. GTV

Hibridas (Nativa/WEB)

- Manejar el historial
 - Webview.goBack()
 - Webview.canGoBack()

```
@Override
public boolean onKeyDown(int keyCode, KeyEvent event) {
    // Check if the key event was the BACK key and if there's history
    if ((keyCode == KeyEvent.KEYCODE_BACK) && myWebView.canGoBack()) {
        myWebView.goBack();
        return true;
    }
    // If it wasn't the BACK key or there's no web page history, bubble
    // system behavior (probably exit the activity)
    return super.onKeyDown(keyCode, event);
}
```



Interfaz de usuario

Otras Interfaces. GTV

Hibridas (Nativa/WEB)

- Depuración
 - WebChromeClient
 - onConsoleMessage()

```
WebView myWebView = (WebView) findViewById(R.id.webview);
myWebView.setWebChromeClient(new WebChromeClient() {
    public void onConsoleMessage(String message, int lineNumber, String sourceID) {
        Log.d("MyApplication", message + " -- From line "
              + lineNumber + " of "
              + sourceID);
    }
});
```



Interfaz de usuario

Otras Interfaces. GTV

Hibridas (Nativa/WEB)

- Recomendaciones
 - Redireccionar los dispositivos móviles a una web adaptada
 - Usar un DOCTYPE adecuado -xHTML-

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML Basic 1.1//EN"
 "http://www.w3.org/TR/xhtml-basic/xhtml-basic11.dtd">
```

- Usar Viewport para adaptar el contenido a los distintos tipos de pantalla

```
<meta name="viewport" content="width=device-width, initial-scale=1.0, user-scalable=no">
```



Interfaz de usuario

Otras Interfaces. GTV

Hibridas (Nativa/WEB)

- Recomendaciones
 - Evitar cargar múltiples ficheros para reducir el consumo de ancho de banda
 - Cargar CSS y JS en <HEAD> o <BODY>
 - Usar compresores de JS y CSS como Minify
 - <http://code.google.com/p/minify/>
 - Evitar que aparezca el scroll horizontal



Interfaz de usuario

Otras Interfaces. Google Glass

Es un conjunto de servicios RESTful que transmiten información y reciben notificaciones de los dispositivos GGlass.

Google Glass se basa en una línea de tiempo.

La línea de tiempo contiene elementos o "tarjetas" que muestran información al usuario.

Los usuarios navegan a través de su línea de tiempo, mostrando las "tarjetas" de esta.

Cada tarjeta de la línea de tiempo contiene información de interés para el usuario. ¿Do You Know Google Now?



Interfaz de usuario

Otras Interfaces. Google Glass

Hay tarjetas predeterminadas en la línea de tiempo que están "asociadas" a una posición en las gafas, por lo que siempre aparecen en el mismo lugar.

La tarjeta que muestra la hora actual y la tarjeta que muestra todas las tareas que pueden ejecutarse son ejemplos de tarjetas predeterminadas.

Podemos definir opciones para permitir a los usuarios ejecutar acciones como eliminar o compartir una tarjeta.



Interfaz de usuario

Otras Interfaces. Google Glass

Para comunicarnos con la línea de tiempo de un usuario, se llama al servicio RESTful adecuado para llevar a cabo la acción que queremos hacer.

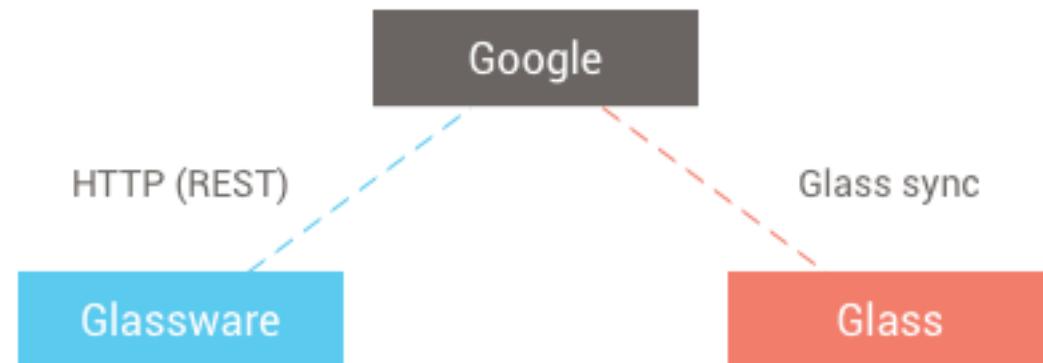
Google se encarga de todos los detalles necesarios de la sincronización con las gafas del usuario.

Algunas de las acciones más comunes son:

Creación y gestión de tarjetas.

Subscripción a notificaciones.

Obtener de la ubicación de un usuario.



Interfaz de usuario

Otras Interfaces. Google Glass

Google Mirror API.

Permite crear servicios basados en la web que interactúan con Google Glass.

Se proporciona esta funcionalidad en un API basada en la nube y no requiere la ejecución de código nativo.

Las tarjetas muestran el contenido que ven los usuarios y son la base de la experiencia del usuario.

Las tarjetas pueden ser de texto, HTML, imágenes o video.

También podemos agrupar las tarjetas en una tarjeta que se expande en una sub-línea de tiempo para facilitar su organización.



Interfaz de usuario

Otras Interfaces. Google Glass

Google Mirror API.

Podemos especificar los elementos del menú que aparecerán en las tarjetas.

Cada elemento puede incluir una colección de acciones como leer en voz alta, responder con la voz o navegar.

Podemos especificar nuestras propias acciones.

Podemos suscribirnos a notificaciones en nuestra aplicación:

Cuando un usuario selecciona los elementos del menú.

Cuando un usuario comparte contenido con un contacto.

La ubicación del usuario, cada diez minutos mínimo



Interfaz de usuario

Otras Interfaces. Google Glass

Google Mirror API.

Para recibir notificaciones, creamos una suscripción para ellas, especificando una URL de devolución donde se recibe la notificación cuando se produzca.

Al recibir la notificación, se puede llevar a cabo la acción deseada en base a la notificación.

Un contacto es una entidad con la que los usuarios pueden compartir sus tarjetas.

Esta entidad puede representar a una persona o incluso a otras aplicaciones.

También podemos especificar los tipos MIME que cada contacto puede manejar, por lo que los usuarios sólo pueden compartir las tarjetas compatibles con la entidad destino.



Interfaz de usuario

Otras Interfaces. Google Glass

Google Mirror API.

Podemos hacer aplicaciones que sean conscientes de localizaciones de los usuarios siempre que los usuarios nos concedan acceso.

Solicitaremos información de la ubicación a través de una petición especial, cuya respuesta será la ubicación de las gafas.

También podemos suscribirnos a actualizaciones de ubicación que se producen a intervalos regulares.

Además, podemos adjuntar una ubicación para una tarjeta y ofrecer una opción en el menú para que el usuario pueda navegar a dicha ubicación.



Interfaz de usuario

Otras Interfaces. Google Glass

Plataforma de desarrollo.

Go



Java



.NET



PHP



Python



Interfaz de usuario

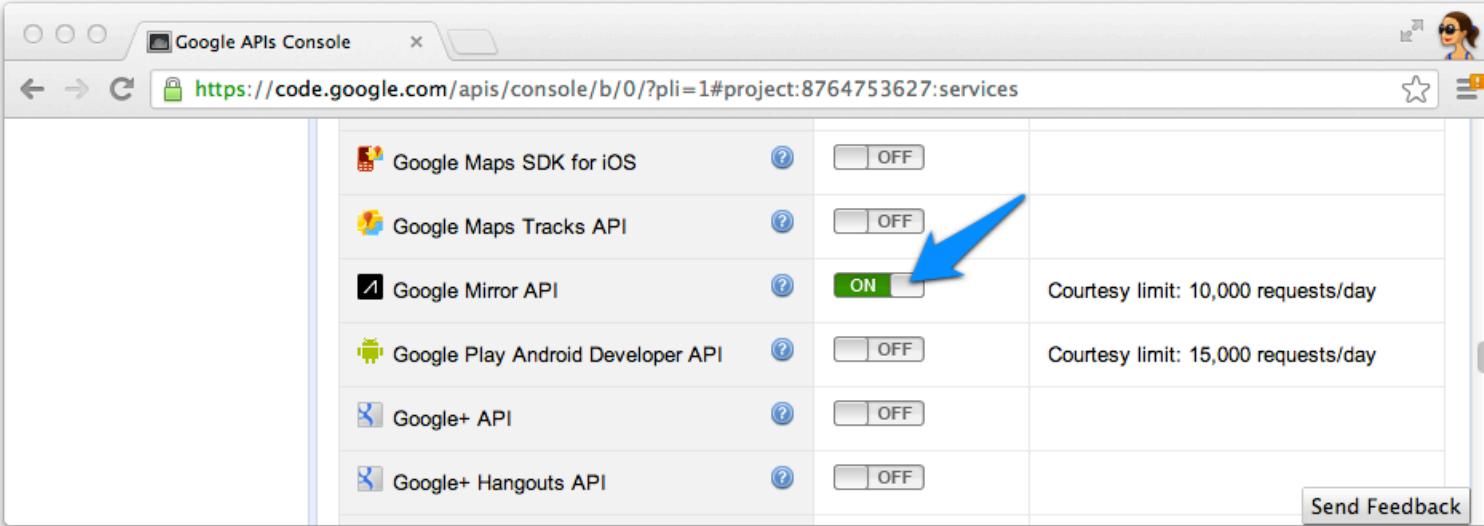
Otras Interfaces. Google Glass

Requisitos.

Dependientes de la plataforma utilizada. (Es un Restful).

Recomendado usar AppEngine. Java + Phyton + PHP

Crear un proyecto en “Googel Api Console”. Requiere ser uno de los afortunados tester seleccionados por Google.



The screenshot shows the Google APIs Console interface. It lists several APIs with their current status (ON or OFF) and courtesy limits. A blue arrow points to the 'ON' button for the Google Mirror API.

API	Status	Courtesy limit
Google Maps SDK for iOS	OFF	
Google Maps Tracks API	OFF	
Google Mirror API	ON	Courtesy limit: 10,000 requests/day
Google Play Android Developer API	OFF	Courtesy limit: 15,000 requests/day
Google+ API	OFF	
Google+ Hangouts API	OFF	



Interfaz de usuario

Otras Interfaces. Google Glass

Playground.

Playground [AUTHORIZE](#)

Authorize Mirror API Playground to access your timeline cards.

This item auto-resizes according to the text length

just now

{
 "text": "This item auto-resizes according to the text length",
 "notification": {
 "level": "DEFAULT"
 }
}

[Download CSS](#) [JSON](#) [Text](#)

[Templates](#) [Timeline](#)

This item auto-resizes according to the text length

TEXT

This **paragraph** auto-resizes according to the **HTML** content length.

AUTO RESIZE



Spring Fling Fundraiser at Filoli

HYBRID

8:00 PM
Dinner with f
tonight
Their place



Interfaz de usuario

Otras Interfaces. Google Glass

Time Line.

```
HTTP/1.1 201 Created
```

```
Date: Tue, 25 Sep 2012 23:30:11 GMT
```

```
Content-Type: application/json
```

```
Content-Length: 303
```

```
{
```

```
  "kind": "glass#timelineItem",
  "id": "1234567890",
  "selfLink": "https://www.googleapis.com/mirror/v1/timeline/1234567890",
  "created": "2012-09-25T23:28:43.192Z",
  "updated": "2012-09-25T23:28:43.192Z",
  "etag": "\"G5BI0RWvj-0jWdBrdWrPZV7xPKw/t25selcGS3uDEVT6FB09hAG-QQ\"",
  "text": "Hello world"
}
```

```
POST /mirror/v1/timeline HTTP/1.1
Host: www.googleapis.com
Authorization: Bearer {auth token}
Content-Type: application/json
Content-Length: 26

{ "text": "Hello world" }
```

Hello World

0 seconds ago



Interfaz de usuario

Otras Interfaces. Google Glass

Menu Items.

```
HTTP/1.1 201 Created
Date: Tue, 25 Sep 2012 23:30:11 GMT
Content-Type: application/json
Content-Length: 303

{
  "text": "Hello world",
  "menuItems": [
    {
      "action": "CUSTOM",
      "id": "complete"
      "values": [
        {
          "displayName": "Complete",
          "iconUrl": "http://example.com/icons/complete.png"
        }
      ]
    }
  ]
}
```

```
HTTP/1.1 201 Created
Date: Tue, 25 Sep 2012 23:30:11 GMT
Content-Type: application/json
Content-Length: 303
```

```
{
  "text": "Hello world",
  "menuItems": [
    {
      "action": "REPLY"
    }
  ]
}
```



Interfaz de usuario

Otras Interfaces. Google Glass

Locations.

```
GET /mirror/v1/locations/ HTTP/1.1  
Authorization: Bearer {auth token}
```

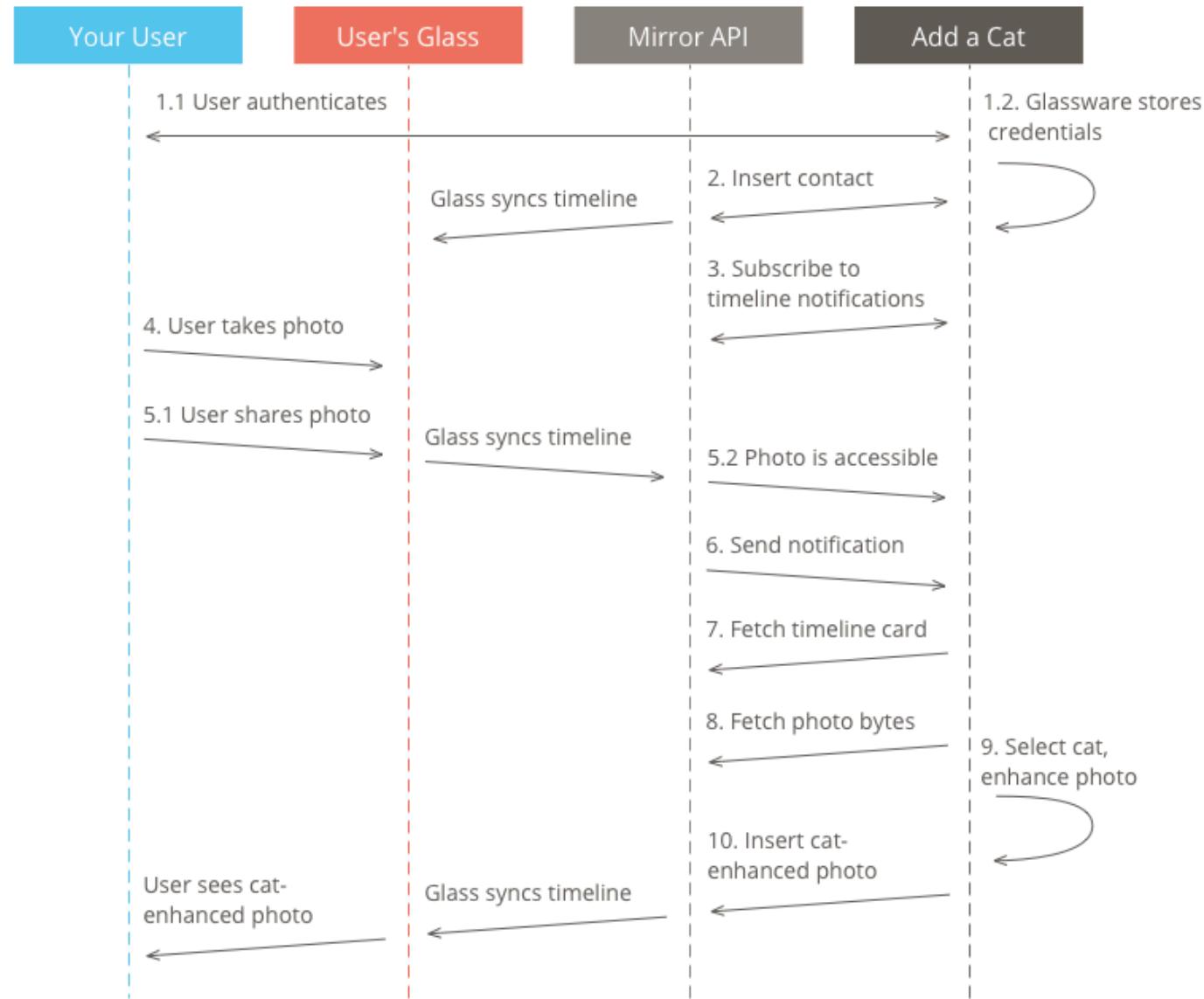
```
POST /mirror/v1/subscriptions HTTP/1.1  
Authorization: Bearer {auth token}  
Content-Type: application/json  
Content-Length: {length}  
  
{  
  "collection": "locations",  
  "userToken": "harold_penguin",  
  "verifyToken": "random_hash_to_verify_referer",  
  "callbackUrl": "https://example.com/notify/callback"  
}
```



Interfaz de usuario

Otras Interfaces. Google Glass

Ejemplos.



Interfaz de usuario

Otras Interfaces. Google Glass

Ejemplos.

