UNIVERSIDAD POLITÉCNICA DE MADRID
ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE SISTEMAS INFORMÁTICOS
MÁSTER EN INGENIERÍA WEB

# PROYECTO FIN DE MÁSTER: TUTORIAL UBUNTU PHONE

**Autor:** Francisco Javier Álvarez Sánchez
**Tutor:** Luis Fernando de Mingo López
Julio 2015

## Agradecimientos

En primer lugar, agradezco a mi tutor del trabajo fin de máster la oportunidad de haber desarrollado este trabajo, en el que he aprendido nuevas tecnologías… nuevas formas de hacer las cosas.

En segundo lugar, no podrían faltar mis padres, que siempre tienen que aguantarme cuando la tensión se apodera de mí en esos momentos en los que el tiempo se me echa encima.

En tercer lugar, a mis colegas, que aunque no ayuden de forma directa al desarrollo del proyecto, siempre nos consolamos al saber que alguien que va aún peor que tú.

En último lugar, y no por ello menos importante, doy las gracias a BQ, que me ha prestado un terminal Aquaris 4.5 Ubuntu Edition en el que he podido testear las aplicaciones desarrolladas.

*"¿Oye eso, Señor Anderson? Eso, es el sonido de lo inevitable."*

## Resumen

El presente trabajo se enmarca dentro del contexto del desarrollo de aplicaciones móviles para el sistema operativo Ubuntu. Android e iOS son las dos tecnologías a la cabeza de esta carrera, pero Ubuntu lucha por hacerse un hueco en el mercado.

Consecuentemente, el propósito principal de este trabajo es introducir a los usuarios a este nuevo mundo, de forma que, lejos de haber desarrollado una aplicación concreta para cubrir unas necesidades específicas, se han implementado cinco aplicaciones sencillas para cada uno de los cinco tipos de programación que Ubuntu proporciona.

Por tanto, el trabajo consiste en un tutorial que permita, a un usuario con conocimientos medios en informática y nulos en desarrollo de aplicaciones móviles, crear su primera aplicación móvil, asentando una fina base sobre la que crear aplicaciones más grandes, completas y estables.

La solución adoptada para la resolución de esta problemática en el presente trabajo fin de máster es un documento que detalla paso a paso cómo instalar el sistema operativo Ubuntu en el ordenador personal del usuario final, configurar el entorno, instalar el entorno de desarrollo de aplicaciones móviles para Ubuntu, el Ubuntu SDK, configurarlo, crear proyectos nuevos para cada tipo de aplicación, desarrollarlas, empaquetarlas y publicarlas en el *market* para su libre distribución.

Al tratarse de un tutorial, se incluye el código necesario en el Anexo I de forma que el usuario final desarrolle sus aplicaciones sin tener que abandonar el presente documento.

## Abstract

This work is framed within the context of the development of mobile applications for the Ubuntu operating system. Android and iOS are the two technologies at the forefront of this race, but Ubuntu struggle to gain a foothold in the market.

Consequently, the main purpose of this paper is to introduce users to the new world, so that, far from having developed a specific application to meet specific needs, are implemented five simple applications for each of the five types of programming Ubuntu provides.

Therefore, the work consists of a tutorial that allows to a user with average knowledge in computers and void in mobile application development, creating its first mobile application, laying a thin base on which to build larger, comprehensive and stable applications.

The solution adopted to solve this problem in this Final Project is a document which details step by step how to install the Ubuntu operating system in the personal computer of the end user configure the environment, install the development environment for mobile applications Ubuntu, the Ubuntu SDK, configure it, create new projects for each type of application, develop them, package them and publish them on the market for free distribution.

As this is a tutorial, the necessary code is included in Annex I so that the end user to develop their applications without leaving herein.

# Índice de contenidos

## Índice de ilustraciones

# Índice de tablas

## Glosario de términos y acrónimos

Catálogo de términos y acrónimos específicos del contexto del trabajo.

- **Ubuntu**: sistema operativo basado en GNU/Linux y que se distribuye como software libre, el cual incluye su propio entorno de escritorio denominado Unity.
- **OS:** sistema operativo, programa o conjunto de programas de un sistema informático que gestiona los recursos de hardware y provee servicios a los programas de aplicación.
- **Windows:** familia de distribuciones de software para PC, smartphone, servidores y sistemas empotrados, desarrollados y vendidos por Microsoft, y disponibles para múltiples arquitecturas, tales como x86 y ARM.
- **Mac:** sistema operativo creado por Apple para su línea de computadoras Macintosh, también aplicado retroactivamente a las versiones anteriores a System 7.6, y que apareció por primera vez en System 7.5.1.
- **LTS:** *Long Term Support*. Una version LTS de Ubuntu que ha tenido 5 años de soporte tanto en la versión *Desktop* (escritorio) como en la *Server*.
- **Login:** proceso mediante el cual se controla el acceso individual a un sistema informático mediante la identificación del usuario utilizando credenciales provistas por el usuario.
- **SDK:** *Software Development Kit*. Conjunto de herramientas de desarrollo de software que le permite al programador crear aplicaciones para un sistema concreto, por ejemplo ciertos paquetes de software, frameworks, plataformas de hardware, computadoras, videoconsolas, sistemas operativos, etc.
- **Release:** versión de un programa o aplicación informáticos que se distribuye a los clientes, incluyendo no sólo el código ejecutable del sistema, sino también archivos de configuración, los archivos de datos, el programa de instalación, la documentación electrónica y el embalaje y la publicidad asociados.
- **PPA:** Personal Package Archives. Repositorio de software especial para subir paquetes fuente para ser construidos y publicados como un repositorio APT para Launchpad. Actualmente es un término que se usa exclusivamente para Ubuntu.
- **QML:** *Qt Meta Language.* Lenguaje basado en JavaScript creado para diseñar aplicaciones enfocadas a la interfaz de usuario.
- **UI:** *User Interface*. Medio con que el usuario puede comunicarse con una máquina, un equipo o una computadora, y comprende todos los puntos de contacto entre el usuario y el equipo.
- **JSON:** *JavaScript Object Notation*. Formato ligero para el intercambio de datos. Es un subconjunto de la notación literal de objetos de JavaScript que no requiere el uso de XML.
- **CSS:** *Cascading Style Sheets*. Lenguaje usado para definir y crear la presentación de un documento estructurado escrito en HTML o XML.
- **XML:** *eXtensible Markup Language.* Lenguaje de marcas desarrollado por el *World Wide Web Consortium* (W3C) utilizado para almacenar datos en forma legible.
- **JavaScript:** lenguaje de programación interpretado, orientado a objetos, basado en prototipos, imperativo, débilmente tipado y dinámico.

- **Back-end:** parte de un sistema software que se encarga de la administración del sitio. La otra parte es el front-end, encargada de la visualización del usuario navegante de dicho sitio.
- **Layout:** esquema de distribución de los elementos dentro del diseño de una página web.
- **Mockup:** en la manufactura y diseño, un *mockup*, o maqueta, es un modelo a escala o tamaño real de un diseño o un dispositivo, utilizado para la demostración, evaluación del diseño, promoción, y para otros fines. Un *mockup* es un prototipo si proporciona al menos una parte de la funcionalidad de un sistema y permite pruebas del diseño.
- **HTML/HTML5:** *HyperText Markup Language*. Lenguaje de marcado para la elaboración de páginas web. HTML5 es la quinta revisión importante y la más empleada actualmente.
- **Apache:** servidor web HTTP de código abierto, para plataformas Unix (BSD, GNU/Linux, etc.), Microsoft Windows, Macintosh y otras, que implementa el protocolo HTTP/1.1 y la noción de sitio virtual.
- **Apache Cordova:** *framework* que permite a los desarrolladores web enfocarse en el desarrollo para los teléfonos inteligentes teniendo como base un código genérico con herramientas tales como JavaScript, HTML, CSS, y creando una interfaz de funciones foráneas para embeber una vista Web en el dispositivo móvil.
- **API:** *Application Programming Interface*. Conjunto de subrutinas, funciones y procedimientos (o métodos, en la programación orientada a objetos) que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción. Son usadas generalmente en las bibliotecas.
- **Framework:** estructura conceptual y tecnológica de soporte definido, normalmente con artefactos o módulos de software concretos, que puede servir de base para la organización y desarrollo de software.
- **Plugin:** aplicación que se relaciona con otra para aportarle una función nueva y generalmente muy específica. Esta aplicación adicional es ejecutada por la aplicación principal e interactúan por medio de la API.
- **Runtime**: en informática se emplea para referirse al software diseñado para dar soporte a la ejecución de programas.
- **USB**: *Universal Serial Bus*. Bus estándar industrial que define los cables, conectores y protocolos usados en un bus para conectar, comunicar y proveer de alimentación eléctrica entre computadoras, periféricos y dispositivos electrónicos.
- **Widget**: pequeña aplicación o programa, usualmente presentado en archivos o ficheros pequeños que son ejecutados por un motor de *widgets* o *Widget Engine*. Entre sus objetivos están dar fácil acceso a funciones frecuentemente usadas y proveer de información visual.

# 1 Introducción

El presente trabajo se enmarca dentro del contexto del desarrollo de aplicaciones móviles para el sistema operativo Ubuntu. Android e iOS son las dos tecnologías a la cabeza de esta carrera, pero Ubuntu lucha por hacerse un hueco en el mercado.

## 1.1 Definición del problema

El problema consiste en que usuarios con conocimientos medios en informática y nulos en desarrollo de aplicaciones móviles no tienen forma de crear su primera aplicación móvil, asentando una fina base sobre la que crear aplicaciones más grandes, completas y estables.

## 1.2 Objetivos

El propósito principal de este trabajo es introducir a los usuarios a este nuevo mundo, de forma que, lejos de haber desarrollado una aplicación concreta para cubrir unas necesidades específicas, se han implementado cinco aplicaciones sencillas para cada uno de los cinco tipos de programación que Ubuntu proporciona.

## 1.3 Medios empleados

Los recursos empleados para elaborar el presente trabajo han sido la bibliografía que aporte el conocimiento necesario de las tecnologías a emplear y de los formatos de visualización, y las herramientas de desarrollo necesarias para cubrir los requisitos.

Las fuentes de bibliografía empleadas han sido las siguientes:

- *Google Scholar.*
- Foros y blogs de internet.

Las herramientas de desarrollo empleadas han sido:

- Ordenador de sobremesa con el sistema Ubuntu instalado.
- Ubuntu SDK.
- BQ Aquaris 4.5 Ubuntu Edition.

## 1.4 Estructura de la memoria

La memoria se inicia con una **primera sección** cuyo objetivo es presentar el trabajo realizado y el contenido del documento.

La **segunda sección**, el estado de la cuestión, establece el contexto en el que se enmarca el problema a abordar en el presente trabajo fin de máster, revisando las tecnologías con las que se va a desarrollar la solución.

La **tercera sección**, solución, detalla la solución llevada a cabo, describiendo el tipo adoptado y el proceso de desarrollo.

La **cuarta sección**, conclusión, recoge un resumen del trabajo realizado y opinión personal del mismo.

La **quinta sección**, bibliografía, reúne las referencias utilizadas para la realización del trabajo.

La **sexta y última sección** contiene un anexo con los fragmentos de código utilizados para desarrollar la solución.

# 2 El estado de la cuestión

Revisión de los trabajos/tecnologías/soluciones relacionadas con la solución a elaborar.

## 2.1 Evolución histórica del desarrollo de aplicaciones para móviles

Están con nosotros en nuestro día a día, los llevamos a todas partes y son capaces de entretenernos, informarnos o hasta trabajar con nosotros. Las aplicaciones se han vuelto parte de nuestra vida y son ya tan comunes por la variedad de plataformas que podemos llegar a encontrar que cualquiera puede acceder a una.

Si nos ponemos a pensar en el primer celular con el que tuvimos contacto (que por el simple hecho de poder hacer llamadas era increíble) comparémoslo con los teléfonos actuales - los famosos *smartphones* – y dimensionemos las diferencias; llevar una cámara, reproductor de música, consola de videojuegos, *e-reader*, *gps*, computadora portátil y teléfono en el bolsillo y en un solo dispositivo, son cosas que veíamos en películas de ciencia ficción y parecían la gran cosa.

La función principal de un Smartphone, o más bien su concepto, es de servir como plataforma para aplicaciones que hagan provecho de las características del teléfono mismo.

En sí, los desarrolladores se ponen a pensar ***¿qué problema puedo solucionar?*** Y tomando en cuenta que los teléfonos cuentan con cosas desde cámaras hasta giroscopios en su hardware y una conexión regular a internet; logran hacer programitas denominadas aplicaciones que pueden ser desde un organizador personal hasta un juego absurdo. Las posibilidades son, prácticamente, infinitas.

Las aplicaciones cumplen una pequeña función dentro de nuestro teléfono, ya sea para comunicarnos como *Whatsapp* y Line que nos ahorran mucho dinero con la mensajería de texto, u otras como *Suit Office* que hacen nos permiten visualizar y editar documentos en nuestro teléfono rápidamente. Y no podemos olvidarnos de los juegos como el ya clásico *Angry Birds* que ha sido descargado mil millones de veces.

Los primeros teléfonos catalogados como *smartphones* aparecieron a finales de los 90 y traían pre cargadas aplicaciones muy básicas como agenda, contactos, *ringtones*, juegos y en algunos casos email. La evolución llega con la tecnología EDGE y su conexión a internet, permitiendo un mayor desarrollo de las aplicaciones ya existentes, pero las restricciones de los fabricantes que hacían sus propios sistemas operativos y que no permitían desarrolladores externos no hacían más que estancar a la industria. Era una época en la que se prestaba más atención al hardware y a los *features*: la evolución de la industria móvil era desordenada y no tenía un rumbo fijo.

Todo cambia con la aparición en 2007 del *Iphone* de Apple que plantea una nueva estrategia, cambiando las reglas de juego, ofreciendo su teléfono como una plataforma para correr aplicaciones que dejaban a desarrolladores y compañías externas ofrecerlas en su app store.

Como se suele decir, dos cabezas piensan mejor que una. Y miles de cabezas pensaban mejor que unas cuantas. Así empezó el boom; para finales del 2008 había prácticamente una aplicación para todo. Cuando la *App Store* abrió contaba con 500 aplicaciones y *Android Market* (*Google Play*) con 50; ahora en 2013 la App Store tiene 775.000 y Google Play 800.000 cada una con una función o funciones que aprovechan las características del teléfono.

Android con su *market* al ser una plataforma *open source* permitió una mayor libertad, y con esto llegaron *smartphones* de bajo costo. Desde el mes anterior la venta de *smartphones* superó a la de teléfonos normales. Ahora es normal de ver a personas de cualquier edad y estatus utilizando aplicaciones, y es por que poco a poco estos aparatos se están volviendo

imprescindibles, y no serían nada sin el abundante y variado ecosistema de aplicaciones que existe para todas las plataformas.

Como dato curioso, desde el año pasado *facebook* registra un mayor tráfico originado desde dispositivos móviles que desde computadoras. Ahora todas las páginas web se preocupan de tener una buena versión adaptada para móviles o en mejor caso una app. El internet tiene la vista puesta en los teléfonos inteligentes y es de esperarse, los contratos de internet móvil doblan a los de conexión fija. El futuro es móvil.

## 2.2 Comparativa entre las principales tecnologías de desarrollo de aplicaciones para móviles

A continuación se incluye una tabla comparativa entre los principales sistemas operativos para terminales móviles. No se va a entrar al detalle en términos de ventajas y desventajas, ya que este documento no pretende desbancar a otras tecnologías, sino presentar la seria de pasos con las que introducirse en este mundo.

| SISTEMA | Black Berry OS | FireFox OS | IOS | TIZEN OS | BREW MP | Ubuntu Phone | Windows Phone | Symbian | Android |
|---|---|---|---|---|---|---|---|---|---|
| IMAGEN REPRESENTATIVA |  |  |  |  |  |  |  |  |  |
| PLATAFORMA (HARDWARE/SOFTWARE) | -(MIDP JAVA Applications, custom B.B. JAVA Applications, default B.B JAVA Applications) -(B.B. API, CLDC, B.B. Plataform) -(B.B. boot rom) -Device (hardware) | -Gaia (GUI). -Gecko (motor de firefox os). -Gonk (kernel y lo necesario para que funcione el dispositivo). -Device (hardware). | -Cocoa Touch. -Media. -Core Services. Core Os. -Device Hardware. | -Tizen Application / Runtime. -Application core Framework. -Recently used application (RUA). -Application utility library (AUL). -OS. -Device (hardware). | -Brew. -Device UI app(s). -Device User Interface. -Device Driver. | -Application native. -Application web. | -Application Runtime. -(Application Mode, UI Model, Cloud Integration) -Kernel -Hardware Device. | -Java ME (UI Framework, aplicación Services). -OS Services -Base Services -Kernel Architecture -Hardware Device. | -Aplicaciones -Entorno de aplicaciones -Librerías nativas. -Runtime de android. -Kernel. -Hardware Device. |
| EMPRESA A CARGO | RIM | Mozilla Corporation | APPLE INC. | Fundación Linux, LIMO, Samsung, Intel. | Qualcomm | Canonicak Ltd. | Microsoft | Symbian Ltc. | Google |
| AÑO DE LANZAMIENTO | 2000 | Inicia en el 2011, se lanza al mercado en 2013. | 2007 | 2012 | B.R.E.W (2001) BREW MP (2010) | 2013 | Windows Movile 2000. Windows Phone 2010 | 1998 | 2007 |

**Tabla 1. Comparativa entre tecnologías - Parte I**

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| TIPO DE CODIGO DE DESAROLLO | Cerrado, rumores de su apertura. | Abierto, multiplataforma. | Cerrado. | Abierto. | Abierto. | Abierto. | Cerrado. | Abierto. | Abierto. |
| COSTO DE LICENCIAS DE DESARROLLO | Gratuita | Gratuita | -80 Euros al año. -PC Mac. | Hasta el momento Gratuita. Bajos costos al publicar en la store. | Gratuita. | Gratuita. | 80 Euros al año. | Gratuita. | 19,22 Euros de por vida, para distribución de Apps en la Play S. |
| PROCESO DE VALIDACION DE APLICACIONES | Estricto y lento, de 1 a 3 semanas. | Lento, sin restricción por ahora. | -Muy estricto 1 semana en promedio. | -Flexible. | Depende de la región. | Sin restricción, publicación en segundos, retrospectiva. | Estricto y lento de 1 a 2 semanas. | Descontinuada. | Bastante flexible de 5 a 30 minutos. |
| VERCIONES DE SO Y AÑO DE LANZAMIENTO | -OS 2.0 (2000) -OS 3.X (2002) -OS 4.X (2004) -OS 5 (2008) -OS 6 (2010) -OS 7 (2011) -B.B. 10 (2013) | -1.0 PRUEBA B2G (2012). -1.0 TEF (Feb. 2013). -1.0.1 Shira (Sep. 2013). -1.1.0 Leo (Oct. 2013). -1.2.0 Koi (Dic. 2013). -1.3.0 TBD (mar. 2014). -1.4.0 TBD (jun. 2014). | -OS 1 (2007) -OS 2 (2009). -OS 3 (2009). -IOS 4 (2010) -IOS 5 (2011) -IOS 6 (2012) -IOS 7 (2013) | Tizen 2.2 (2013) Tizen 2.2.1 (2013) Tizen 3.0 (Próximo al mercado 2014) | B.R.E.W (2001) BREW MP (2010) | Ubuntu OS 2013 | *W. Mobile* -PocketPc (2002). -W. Mobile (2003). -W. Mobile II (2003). -W. Mobile 5 (2005). -W. Mobile 6 (2007). -W. Mobile 6.1 (2008). -W. Mobile 6.5 (2009). *W. Phone* -W. Phone 7 | -EPOC 32 1.0 (1998). -Symbian OS 6.0 (2001). -Symbian OS 6.1 (2002). -Symbian OS 7.0 (2003). -Symbian OS 7.0 II (2004). -Symbian OS 8.1 (2005). -Symbian OS 9.1 (2006). -Symbian OS 9.2 (2007). -Symbian OS 9.3 (2008). -Symbian OS 9.5 (2010). | -Beta no commercial (2007). - Apple Pie v1(2008). -Banana Bread v1.1(2009). -Cupcake v1.5(2009). -Donut v1.6(2009). -Eclair v2.0/v2.1.(2009). -Froyo v2.2(2010). -Gingebread |
| | | | | | | | Photon (2010). - W. Phone 7.1 Nodo (2010). - W. Phone 7.5 Mango (2011). - W. Phone 7.5.1 Refresh (2012). -W. Phone 7.8 Windows Phone 8 (2013). | -Symbian OS 9.5 Anna (2011). -Symbian OS 10.1 (2012). | v2.30(2010). - Honeycomb v3.0/3.1/3.2 (2011). -Ice Cream Sandwich v4.0(2011). -Jeally bean v4.1/4.2/4.3 (2012). -KitKat v4.4(2013). |
| USOS/ APLICACIONES | Profesional/empresarial/usuario normal. | Desarrollador/usuarios de bajos recursos /aficionados, usuario normal. | Profesional/ Usuario común. | Desarrollador / usuario común. | Usuario normal principalmente. | Desarrollador/ usuario común. | Desarrollador/ usuario común. | Desarrollador/ Usuario común. | Desarrollador/ empresarial / usuario común. |
| CARACTERISTICAS | -El s.o está orientado a su uso profesional como gestor de correos. -Usuarios pequeños cuentan con BB. Internet | Firefox OS posee diversas caracteristicas de uso, que varian en cada actualización. Entre las principales denotamos las siguientes: | -Interfaz de usuario multitouch. -Control Center. -AirDrop. -Siri. -Tienda de aplicaciones App store. | -Orientado para aplicaciones HTML5. -Bibliotecas desarrollo derivadas de Enlightement -Fácil de programar. -Tizen Store. | SDK, modelo basado en API's independientes que cubren toda la funcionalidad básica de una terminal. Multiplatafor | -Sistema diseñado para plataformas móviles. -Pantalla de inicio sin sistema de bloqueo/desbloqueo. -Aplicaciones en segundo | -interfaz de usuario. -Entrada de texto. -Navegador web. -Búsqueda. -Hubs. -Contactos. -Fotos. -Música y | -Uso eficiente de todos los recursos de la maquina. - Multitareas. -Manejo fiable de datos. -Solo trabaja | Almacenamiento. - Conectividad. -Mensajeria. -Navegador web. -Soporte de Java. -Soporte |

**Tabla 2. Comparativa entre tecnologías - Parte II**

**Tabla 3. Comparativa entre tecnologías - Parte III**

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | service. -Para crear una aplicación para este sistema ocupan una firma digital. - RIM desarrolla su propio software para sus dispositivos. -Tienda de aplicaciones AIR World. | -Almacenamiento. -Aplicaciones web. -Búsqueda adaptativa. -Diseño de dispositivo. -contactos. -Correo electrónico. -Calendario. -Navegador web. -GPS. -Mensajería. -Multimedia. Notificaciones -Radio. -Tienda de aplicaciones FireFox MarcketPlace. | | | ma con soporte a gran cantidad de fabricantes. Da mayor libertad a los desarrolladores (puesto que tienen API's en C, C++, LUA, Flash, Java y HTML). Geoposicionamiento. MultiThread (Multitarea). Reduce costos de implementación y los plazos de comercialización. | plano. -Integración con Ubuntu one. | videos. -Office. -Tienda Windows Phone Store. | sobre procesadores ARM. -Adaptativo a hardware. -Tienda de aplicaciones Ovi Store. | multimedia. -Soporte para streaming. -Soporte para hardware adicional. -Entorno de desarrollo. -Multitactil. -Bluetooth. - Videollamada. -Multitarea. - Características basadas en voz. -Tethering. -Tienda Google Play. |
| IDE (ENTORNO DE DESAROLLO) | -Java Plug in para eclipse. + IBB Java Development Environment (JDE) | -Firefox Aurora. Complemento de desarrollo. -Simulador. | -Xcode + ios SDK. -Instruments. -Dash code. -Simulador. -Interface | -Tizen SDK. | -Visual studio. Eclipse. | -SDK. -Ubuntu OS. | -Visual Studio + SDK Windows Microsoft -Silverlight. -Microsoft | -Caribe C++. -Plug ins. | -ADT. -Android Studio. -App Inventor. |

**Tabla 4. Comparativa entre tecnologías - Parte IV**

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | | Builder. -Corona SDK y Lua. -PhoneGap. -Mac Os. | | | | XNA Framework. Phone. | | |
| LENGUAJE(S) DE PROGRAMACION | C,C++, C#, JAVA. | HTML. CSS, JS, C++ | Objetive C, Java, C, C++. | HTML5, C++, JS. | C,C++. | Html5, QML, JS, CSS. | C# .NET | Java, c++, Visual Basic, Phyton, Perl, Flahs lite, etc. | C, C++, Java, XML. |
| URL OFICIAL DEL SITIO | http://mx.blackberry.com/ | http://www.mozilla.org/ | http://www.apple.com/ | https://www.tizen.org/ | https://www.brewmp.com/ | http://www.ubuntu.com/ | http://www.windowsphone.com/ | http://licensing.symbian.org/ | http://www.android.com/ |
| USO EN EL MERCADO (Resultados basados en estudios "Gantner " 2013) | Se estiman más de 50 millones de usuarios de este sistema. Representan el 1,9% del mercado. | No se tiene una estimación exista debido a su recién aparición en el mercado. | Aproximadamente ocupa el 15,6% del mercado internacional | | Se estima que ocupa un 15% en el mercado actual. | No se tiene una estimación exista debido a su recién aparición en el mercado. | Se estima que ocupa un 3,2% en el mercado actual. | A pesar de estar descontinuada sigue ocupando un porcentaje considerable en el mercado. | Se estima que Android a finales del 2013 ocupo el 78,4% del mercado. |

**Tabla 5. Comparativa entre tecnologías - Parte V**

## 2.3 Las diferentes formas de programar en Ubuntu

En este apartado se incluyen unas breves descripciones sobre las diferentes formas de programar en Ubuntu, para saber elegir en función de las necesidades.

### WebApp

Las webapps de Ubuntu son sitios alojados en la web que se muestran dentro de una app. En esencia son apps que se instalan, se ven, se ejecutan y se usan como cualquier otra, pero cuyo contenido es proporcionado a través de URLs.

### Scopes

Los scopes de Ubuntu son visualizaciones, más allá que simples aplicaciones, que muestran contenido y servicios directamente a la pantalla. La experiencia visual es la misma que al tratar con aplicaciones, con la ventaja de ahorrarse el trabajo necesario para empaquetar y mantener las aplicaciones tradicionales, y con un *toolkit* que aporta nuevos componentes visuales.

### QML

Es un lenguaje declarativo basado en JavaScript para diseñar interfaces de usuario muy intuitivas. Permite integración con JavaScript.

### QML con C++

En el caso de aplicaciones más complejas en la parte *backend*, más allá de la visualización de la aplicación, se puede integrar C++ para cubrir esta carencia.

### Apache Cordova con HTML5

Una aplicación Ubuntu HTML5 está escrita en HTML5, CSS y JavaScript y que se ejecuta en un contenedor web. El aporte de Ubuntu es la integración con la plataforma de APIs, como por ejemplo la API de Apache Cordova con la que tener acceso a la cámara y al acelerómetro.

# 3 Solución

El objetivo de este apartado consiste en presentar la solución propuesta al problema expuesto en el apartado Definición del problema. El apartado está dividido en dos sub-apartados: el primero está orientado a la descripción de la solución, exponiendo la forma en la que se abordan los objetivos principales del trabajo presentados en el apartado Objetivos; el segundo apartado se focaliza en el proceso de desarrollo, detallando las fases de desarrollo realizadas para la creación del tutorial: en este caso solo procede la fase de implementación.

## 3.1 Descripción de la solución

La solución adoptada para la resolución de esta problemática en el presente trabajo fin de máster es un documento que detalla paso a paso cómo instalar el sistema operativo Ubuntu en el ordenador personal del usuario final, configurar el entorno, instalar el entorno de desarrollo de aplicaciones móviles para Ubuntu, el Ubuntu SDK, configurarlo, crear proyectos nuevos para cada tipo de aplicación, desarrollarlas, empaquetarlas y publicarlas en el *market* para su libre distribución.

## 3.2 El proceso de desarrollo

En este apartado se detalla el proceso de desarrollo seguido, detallando el modelo de proceso aplicado para elaborar la solución y las fases de las que consta la solución: como se ha dicho, solo procede la **fase de implementación**, donde se detallan todos los pasos para elaborar las aplicaciones móviles.

### Implementación

La fase de implementación está dirigida a conocer las características del proyecto desarrollado, identificando la arquitectura del proyecto y la jerarquía de clases de la que se compone. En el caso de este tutorial, las sub-fases de la implementación son: instalación y configuración del sistema operativo Ubuntu y del SDK, o entorno de desarrollo; implementación de las apps, explicando también cómo crear proyectos y ejecutar aplicaciones genéricas, y cómo empaquetarlas y publicarlas.

#### *Instalación y configuración*

##### 3.2…1 Ubuntu OS

En este apartado se detallan los pasos para instalar el sistema operativo Ubuntu en nuestros ordenadores. Lo más común es que dispongamos de un equipo con Windows instalado, o Mac, y habilitemos una nueva partición en la que instalar Ubuntu para no perder nuestro sistema operativo previo (en este documento no se explica cómo realizar una partición). Todos los problemas que puedan surgir durante la instalación se pueden consultar bien en la página de Ubuntu (www.ubuntu.com) o en cualquier foro en general.

A continuación se detallan los pasos a seguir para la instalación:

- Grabar un DVD con la última versión de Ubuntu (la última es la 15.04 aunque la 14.04.2 LTS es la más estable; en el caso de este tutorial, la 14 es más que suficiente).
- Insertar el DVD en el ordenador y reiniciarlo. Debería aparecer una pantalla de bienvenida que de la opción de instalar Ubuntu (Ilustración 1).


**Ilustración 1. Pantalla de bienvenida de instalación de Ubuntu**

- *Otra opción es realizar la instalación desde un USB, cosa que la mayoría de ordenadores actuales permite, siguiendo para ello los pasos anteriores pero sustituyendo el DVD por el USB. En el caso de que al reiniciar, el ordenador no reconozca el USB, se debe acceder al menú de arranque para configurarlo (se accede normalmente presionando la tecla F12).
- Prepararse para la instalación: enchufar el ordenador a la corriente (si procede), asegurarse de que hay espacio suficiente para la instalación y estar conectado a internet para que se vayan descargando las últimas actualizaciones (Ilustración 2).


**Ilustración 2. Preparación para la instalación de Ubuntu**

- En el caso de no estar conectado a internet, la siguiente pantalla permite seleccionar una red de internet (Ilustración 3) (como ya se ha mencionado en el apartado anterior, esto es recomendable para que se vayan descargando e instalando las últimas actualizaciones).

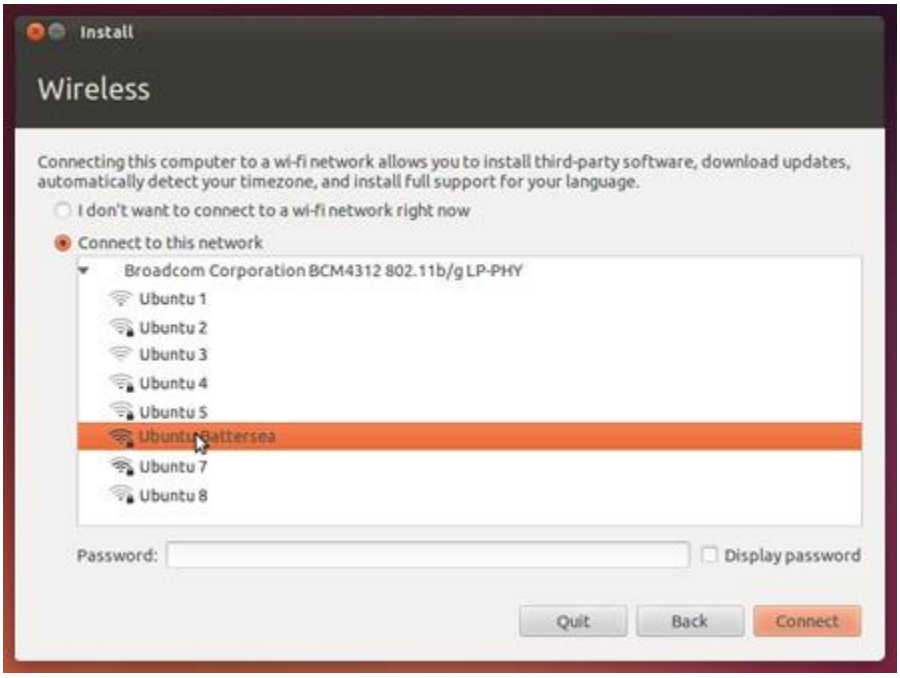



**Ilustración 3. Conectarse a una red durante la instalación de Ubuntu**

- Elegir dónde instalar Ubuntu, si en una partición independiente de Windows (o Mac), eliminando el sistema operativo actual, o en el caso de usuarios expertos, elegir la última opción con configuraciones más específicas (Ilustración 4).


**Ilustración 4. Seleccionar el tipo de instalación de Ubuntu**

- Empezar la instalación comprobando que los pasos anteriores son los deseados (Ilustración 5), y seleccionando en las pantallas siguientes la localización del usuario (Ilustración 6, para configurar la zona horaria), el lenguaje del teclado (Ilustración 7) e introduciendo los detalles de login (Ilustración 8, usuario y contraseña).


**Ilustración 5. Empezar la instalación de Ubuntu**

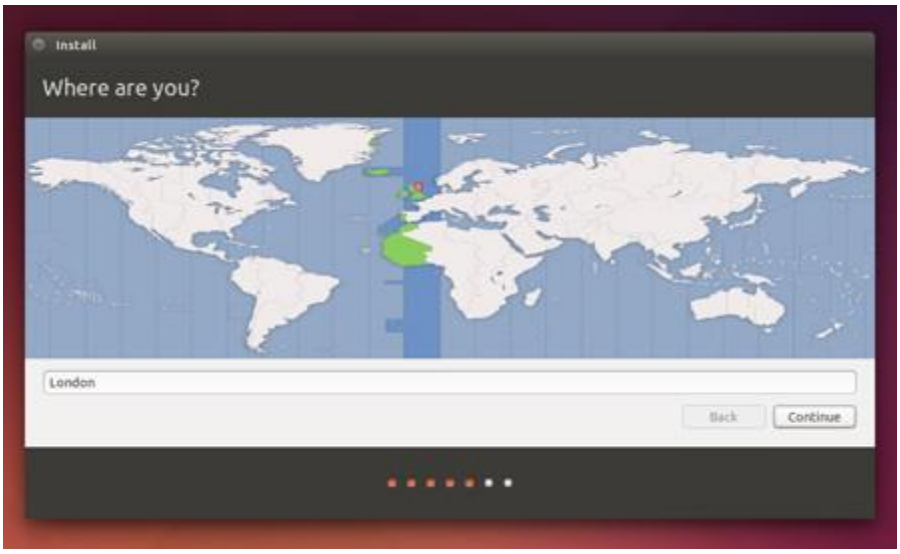



**Ilustración 6. Seleccionar localización para instalación de Ubuntu**

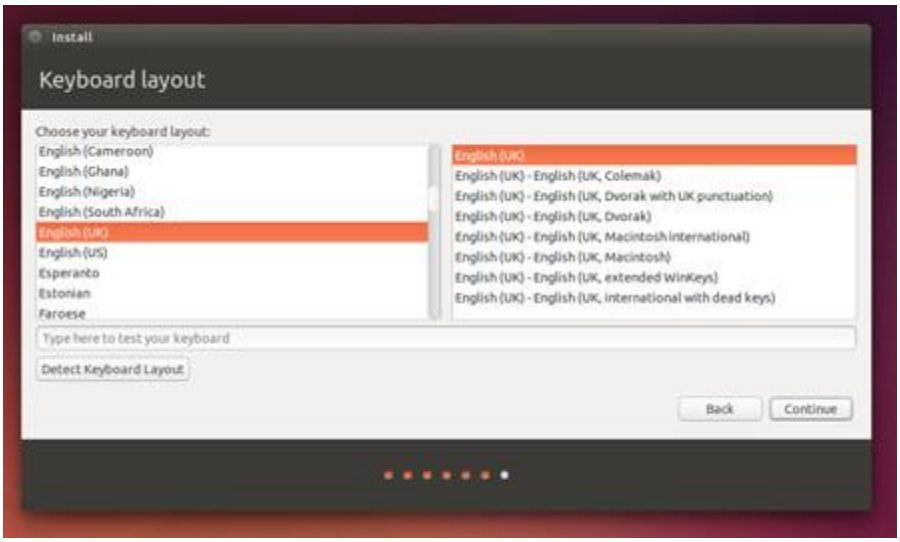



**Ilustración 7. Seleccionar lenguaje del teclado para instalación de Ubuntu**

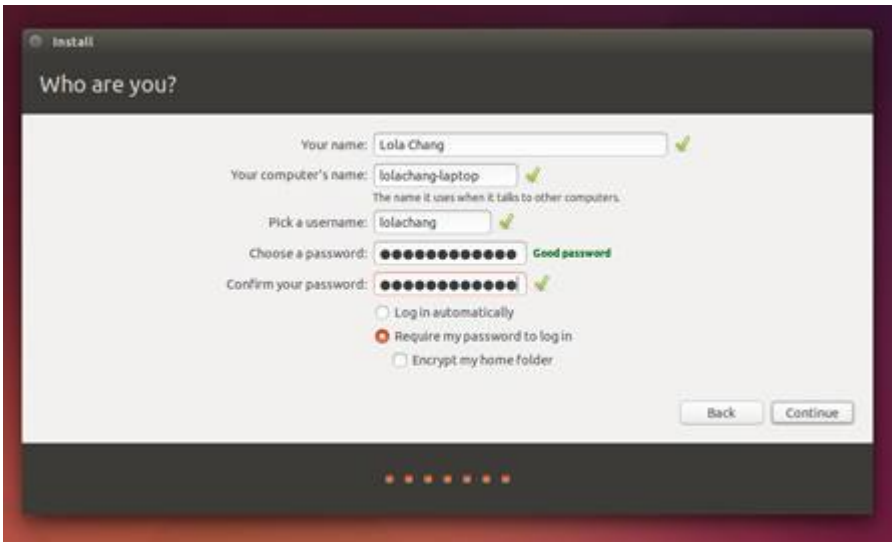



**Ilustración 8. Introducir datos de login para instalación de Ubuntu**

- Ahora sólo toca esperar a que se instale todo (Ilustración 9), hasta que se nos pida reiniciar el ordenador (Ilustración 10), momento en el que la instalación está completa.


**Ilustración 9. Progreso en la instalación de Ubuntu**

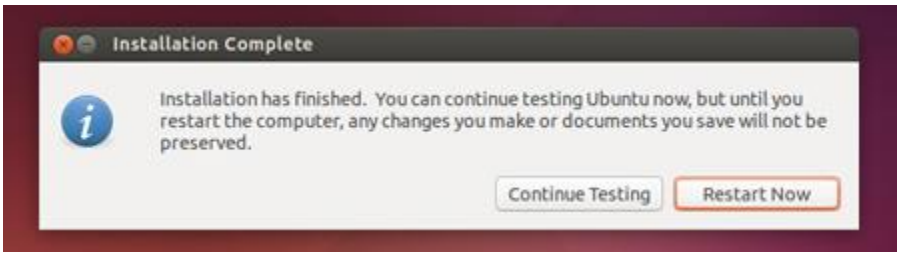



**Ilustración 10. Instalación de Ubuntu completada**

### 3.2…2 Ubuntu SDK

Una vez instalado el sistema operativo Ubuntu, se procede con la instalación del entorno de trabajo, el SDK de Ubuntu. Para ello, basta con ejecutar las siguientes líneas en un terminal de comandos:

- Añadir la SDK Release PPA:
  *sudo add-apt-repository ppa:ubuntu-sdk-team/ppa*
- Instalar el SDK de Ubuntu:
  *sudo apt-get update && sudo apt-get install ubuntu-sdk*
  Siendo también recommendable, aunque no obligatorio, actualizar los paquetes:
  *sudo apt-get update && sudo apt-get dist-upgrade*
- Una vez instalado, podemos ejecutar el entorno buscando el icono de "Ubuntu SDK" (Ilustración 11)


**Ilustración 11. Icono del Ubuntu SDK IDE**

O desde consola mediante la siguiente línea:
*ubuntu-sdk*

## *Implementación de las apps*

Este apartado tiene como objetivo el servir como tutorial para desarrollar aplicaciones Ubuntu mediante su SDK, previamente instalado y configurado en el apartado anterior Instalación y configuración. Los tutoriales se organizan en la serie de pasos que el usuario debe seguir para desarrollar las apps, ilustrados convenientemente para facilitar las tareas.

Este apartado se divide en ocho sub-apartados:

- Los dos primeros detallan cómo crear un proyecto genérico de Ubuntu y cómo lanzar la aplicación resultante.
- Los cinco siguientes explican cómo desarrollar una app concreta para cada uno de los tipos especificados en el apartado Las diferentes formas de programar en Ubuntu.
- El último expone cómo empaquetar una aplicación y cómo publicarla.

### 3.2…1 Creación de un nuevo proyecto en el Ubuntu SDK

El primer apartado está destinado a explicar los pasos a seguir para crear un nuevo proyecto con el SDK de Ubuntu y lanzar la app resultante en un emulador (o en un dispositivo si se tiene). Estos pasos son genéricos para cualquier proyecto Ubuntu, habiendo tomado como ejemplo uno de HTML5. Los proyectos específicos de Ubuntu están todos listados bajo el tipo de proyecto Ubuntu en la ventana de nuevo proyecto.

Para crear un proyecto, lanzamos el SDK de Ubuntu y seguimos estos siete pasos (las indicaciones se especifican en inglés, idioma más común del entorno):

1. Lanzar la ventana de creación de proyecto con: **File > New File or Project**.
2. Seleccionar el tipo de proyecto **Ubuntu**, después **HTML5 App**, y pulsar el botón **Choose**, como se muestra a continuación:

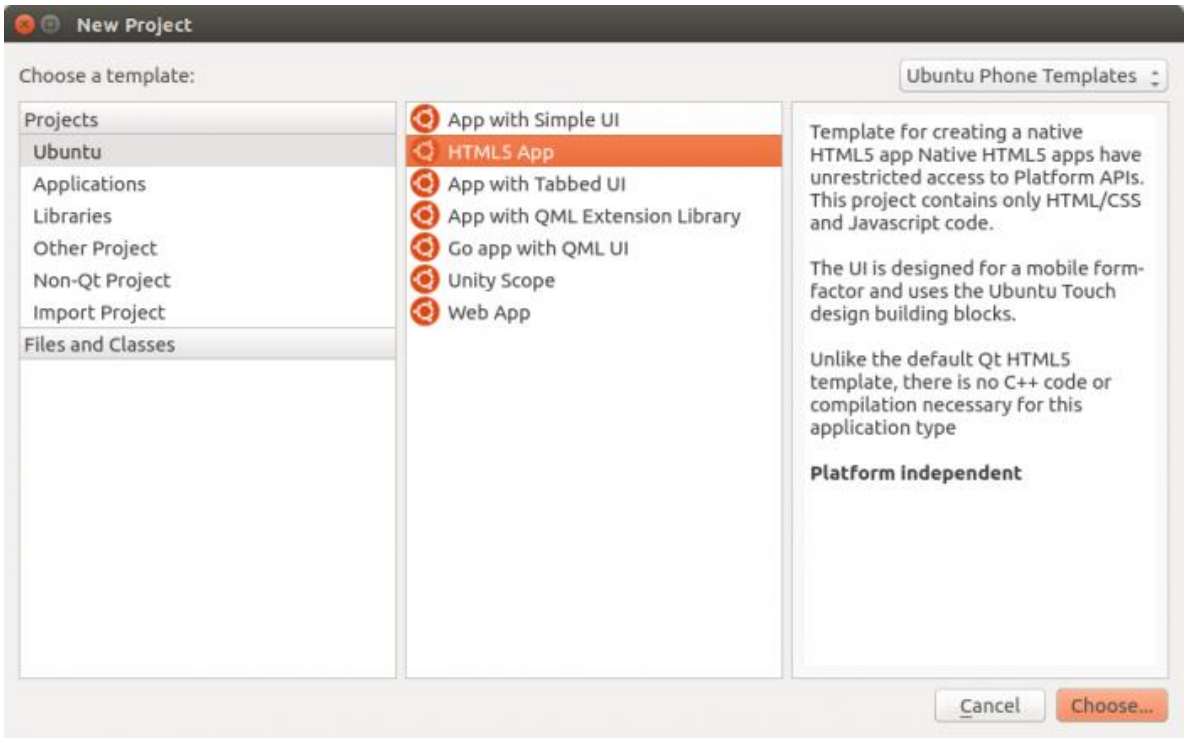



**Ilustración 12. Crear un proyecto Ubuntu I**

3. Especificar el nombre del proyecto (**Name**), el directorio donde crearlo, y clicar el botón **Next**, como se muestra a continuación:

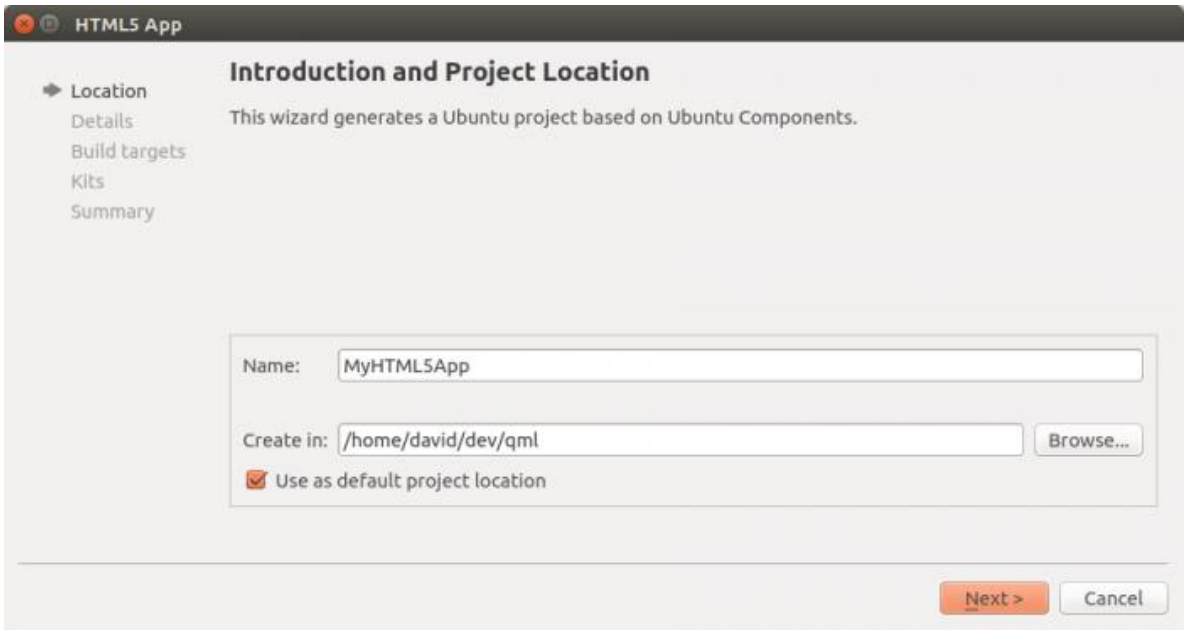



**Ilustración 13. Crear un proyecto Ubuntu II**

4. En la página **Click package parameters**, introducir:
    a. **Nickname**, el nombre de usuario empleado para publicar aplicaciones.
    b. **Manteiner**, información personal tal que nombre y email.
    c. **App name**, el nombre de la aplicación, independiente del nombre del proyecto.

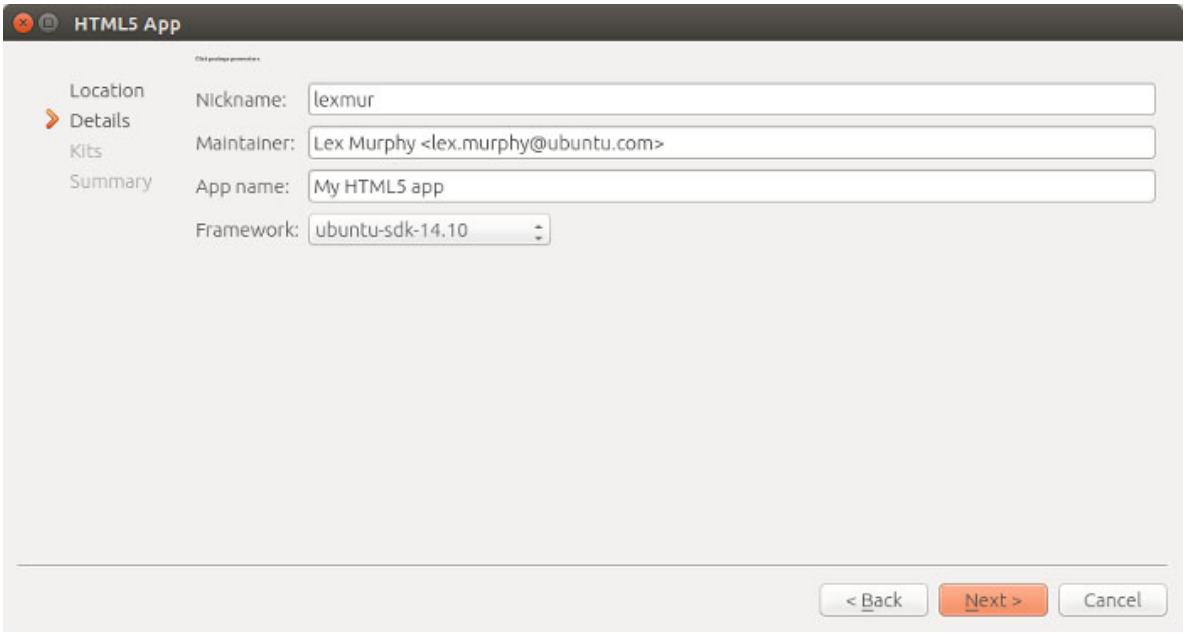



**Ilustración 14. Crear un proyecto Ubuntu III**

5. En la página **Build Targets** se pide crear al menos un kit de dispositivo para el proyecto (en el caso de que no exista ninguno). Los kit son necesarios para construir y ejecutar las aplicaciones. El botón **Create new kit** permite crear un kit de dispositivo de una manera

muy sencilla (en el siguiente apartado, Ejecutar aplicaciones en el Ubuntu SDK, se especifican los diferentes kit y sus características):

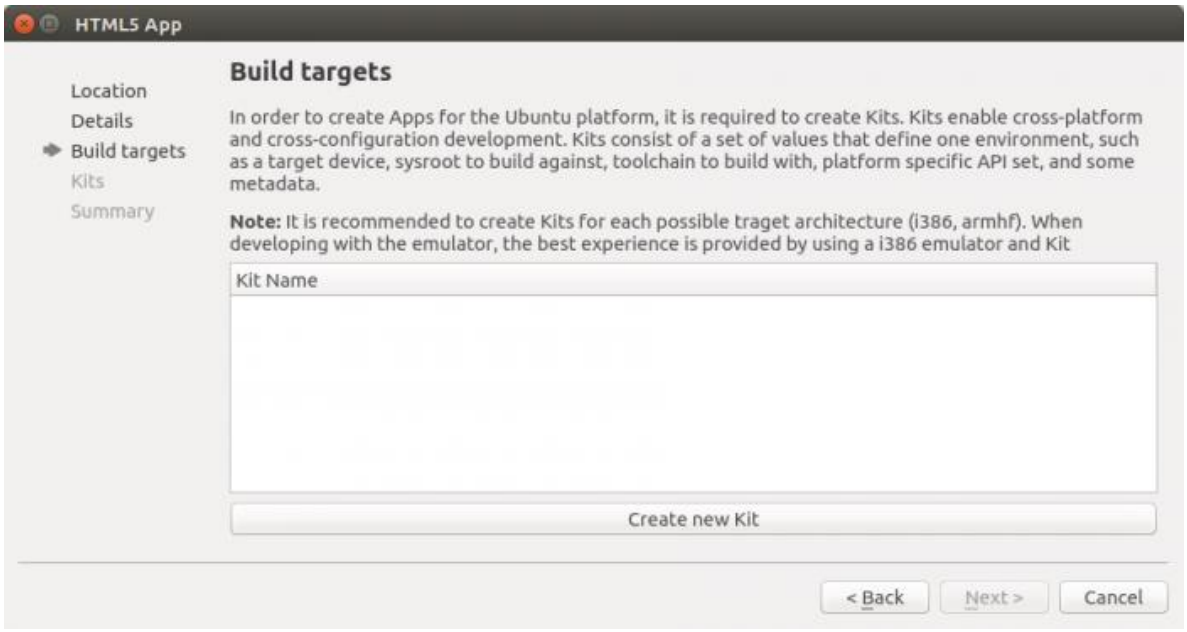



**Ilustración 15. Crear un proyecto Ubuntu IV**

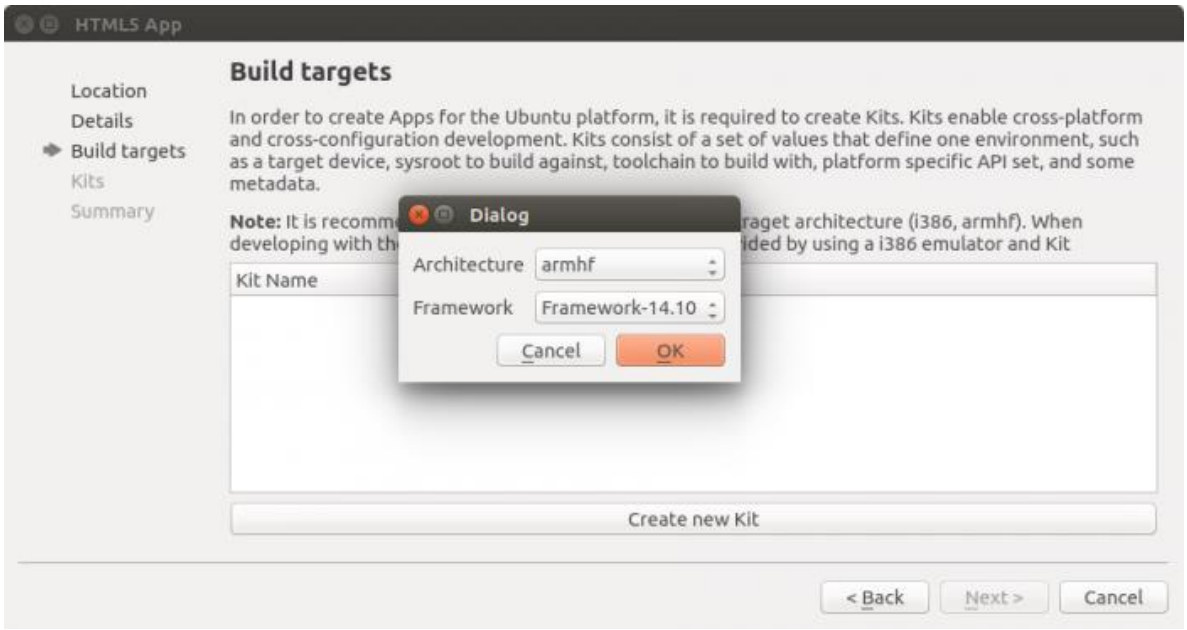



**Ilustración 16. Crear un proyecto Ubuntu V**

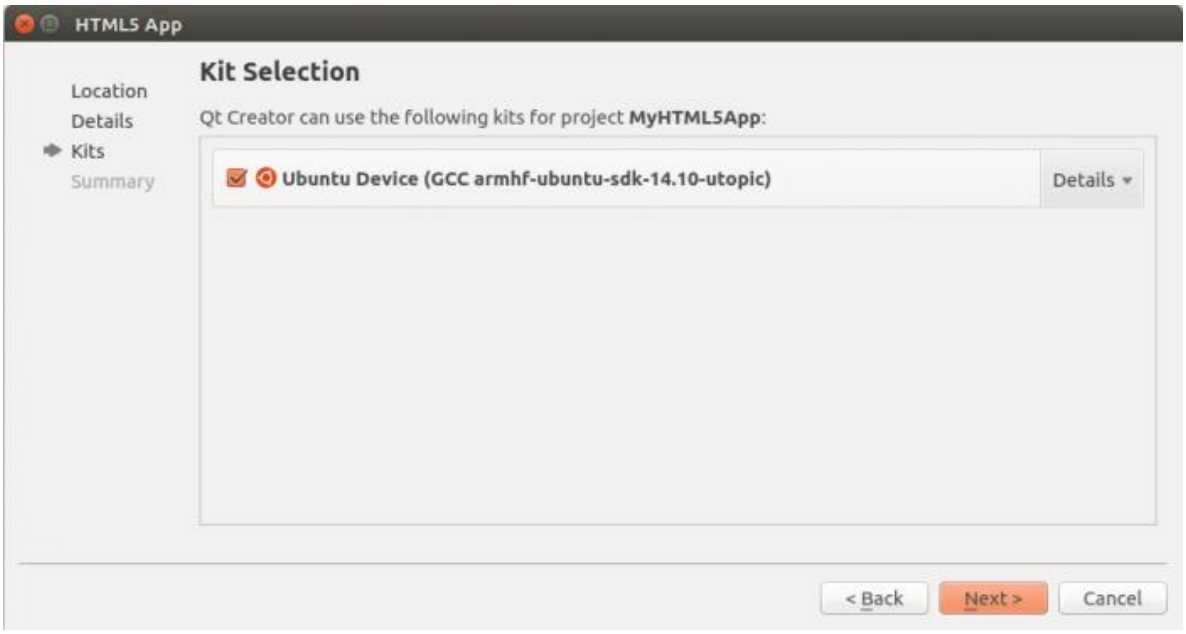



**Ilustración 17. Crear un proyecto Ubuntu VI**

6. Después de crearlo, se selecciona para continuar al siguiente paso. En la página Project Management, se pulsa el botón Finish para finalizar y tener el proyecto creado:

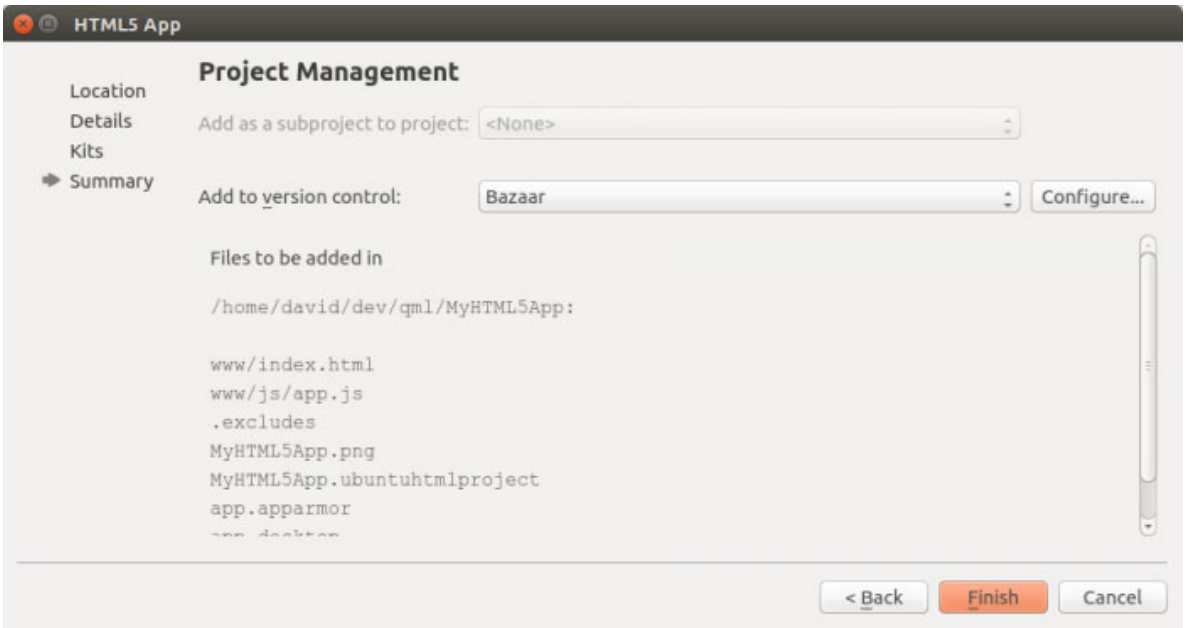



**Ilustración 18. Crear un proyecto Ubuntu VII**

7. Una vez creado el proyecto, el sistema vuelve a la interfaz del SDK, mostrando la página principal (archivo 'index.html' en el caso de este tipo de proyecto) del proyecto:

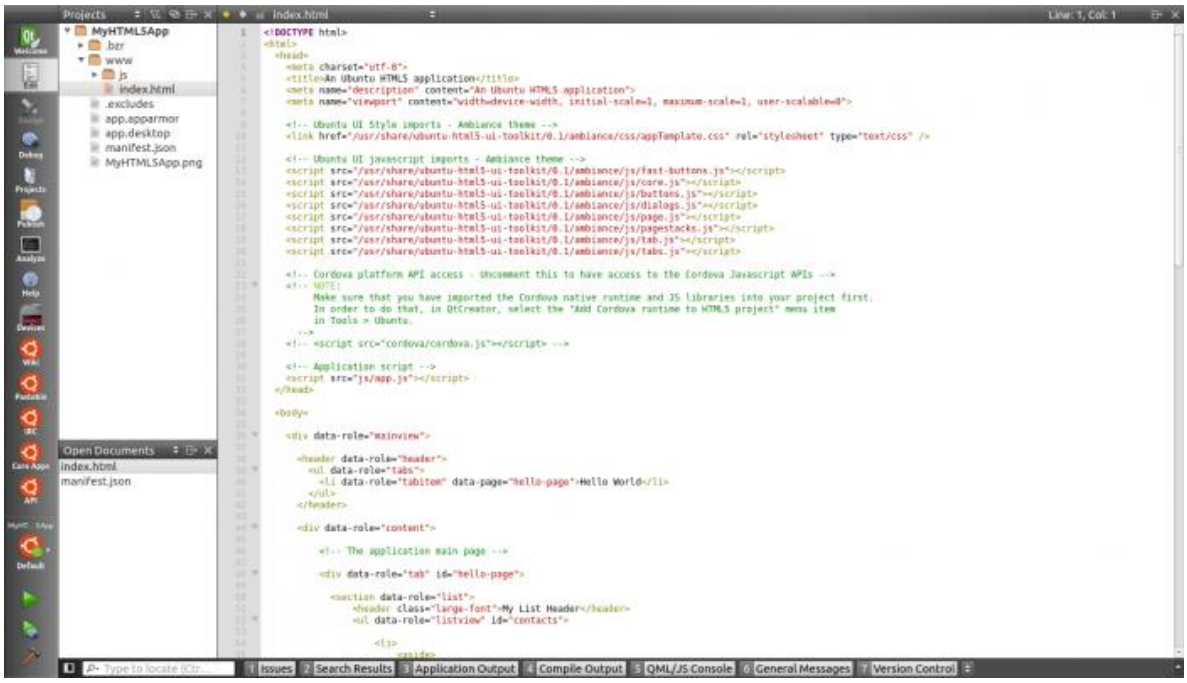



**Ilustración 19. Crear un proyecto Ubuntu VIII**

### 3.2…2 Ejecutar aplicaciones en el Ubuntu SDK

Este apartado explica las distintas formas, y los pasos a seguir, para ejecutar una aplicación Ubuntu en el SDK. Existen tres formas de hacerlo: en el escritorio, en un dispositivo Ubuntu conectado al equipo y en un emulador. Los tres tipos se consideran complementarios, si bien cada uno tiene sus ventajas e inconvenientes. En este punto se aprende a manejar los diferentes tipos de dispositivos y cual usar según lo que se quiera testear de las aplicaciones desarrolladas.

En el punto anterior, Creación de un nuevo proyecto en el Ubuntu SDK, se detallan los pasos a seguir para crear un proyecto y en el paso 5 se selecciona el kit de dispositivo, o se crea sino se tiene ninguno. Este kit consta básicamente de dos aspectos:

- El **framework** que la aplicación usará. Normalmente se selecciona una versión estable que garantice una correcta ejecución, como la 14.04 por ejemplo.
- La arquitectura (**architecture**) del dispositivo (*armfh* para la mayoría de *smartphones* y *tablets* e *i386* para el escritorio).

Añadir múltiples kits permite testear las aplicaciones en múltiples contextos, arquitecturas y tamaños de pantalla.

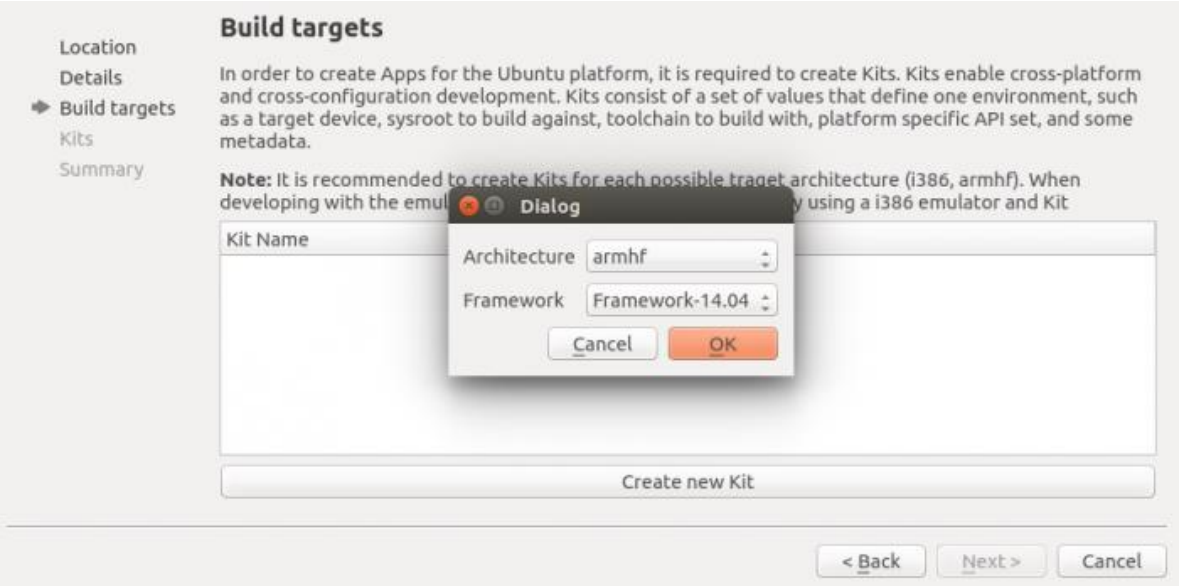



**Ilustración 20. Ejecutar aplicación Ubuntu I**

En el Ubuntu SDK, la pestaña "*Build & Run*" dentro de la página de "*Projects*" permite crear, eliminar y modificar los kits existentes:

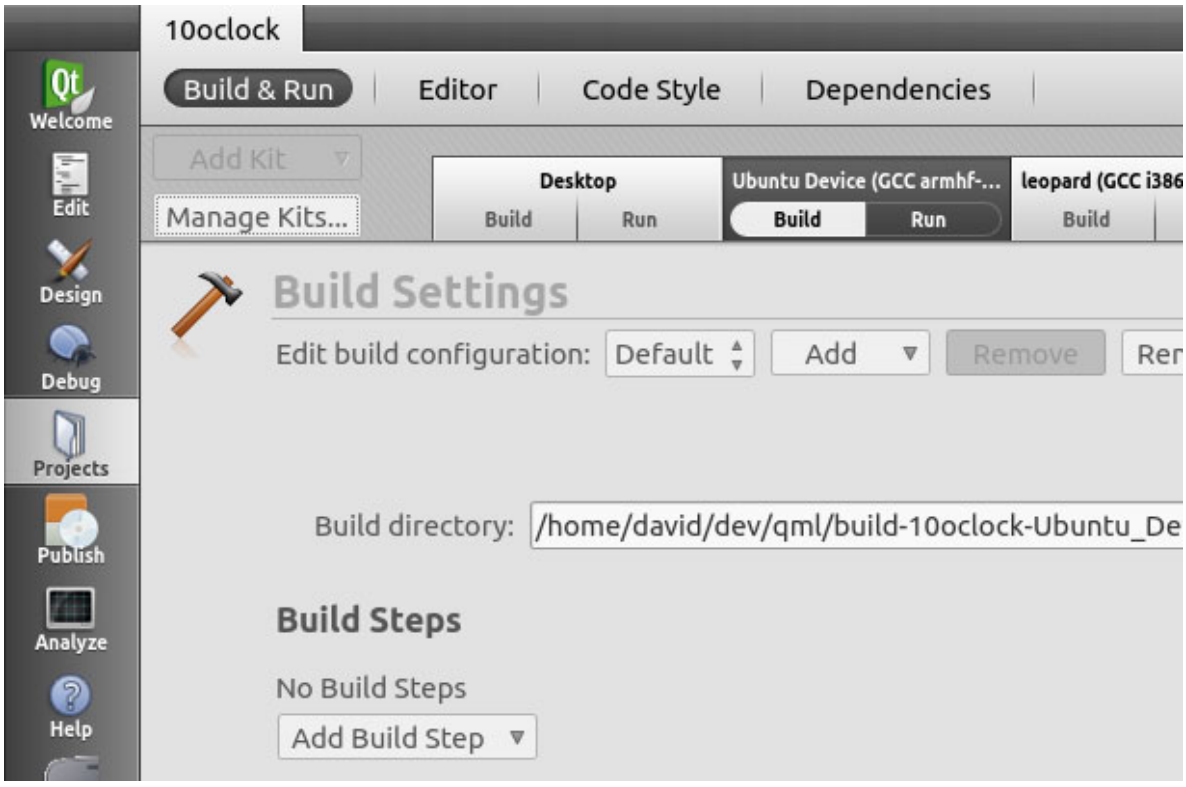



**Ilustración 21. Ejecutar aplicación Ubuntu II**

La página "*Devices*" permite ver los dispositivos Ubuntu conectados, gestionar sus configuraciones y crear nuevos dispositivos (emuladores).

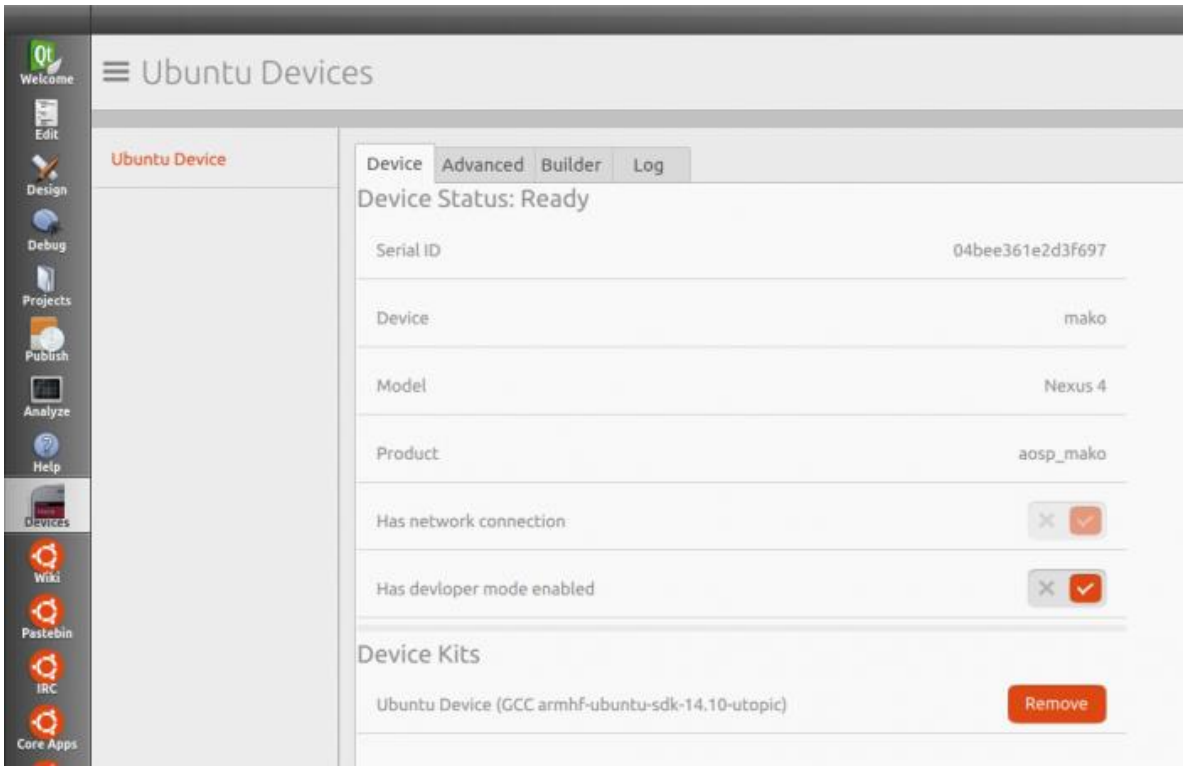



**Ilustración 22. Ejecutar aplicación Ubuntu III**

Pulsando el botón "+" en la parte de debajo de la página anterior abre un diálogo "*Create emulator*". Se debe introducir un nombre para el emulador, una arquitectura (las opciones son las mismas que con los kits de dispositivos) y un canal para la imagen del sistema: *stable* es la última *release* oficial de Ubuntu, *devel* es la última construcción validada diariamente y *devel-proposed* contiene nuevos cambios aún pendientes de testear.

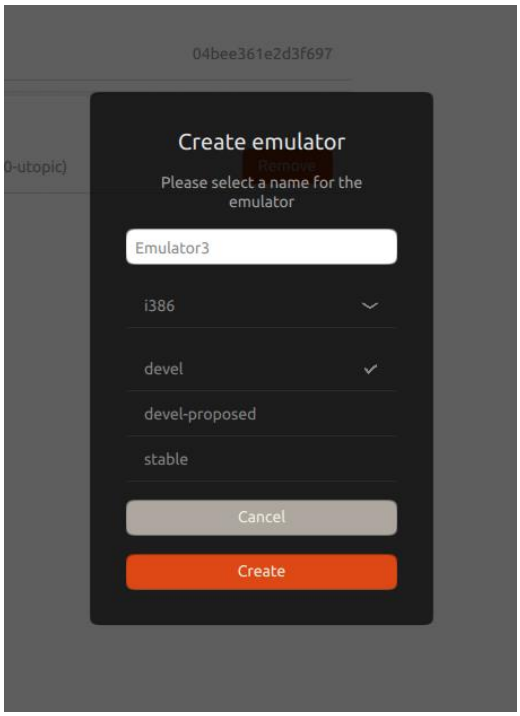



**Ilustración 23. Ejecutar aplicación Ubuntu IV**

Una vez que se ha creado el emulador, se pueden configurar algunos parámetros:

- **Scale**: el tamaño de la ventana del emulador.
- **Memory**: la memoria RAM asociada al dispositivo (512 – 1024 MB)

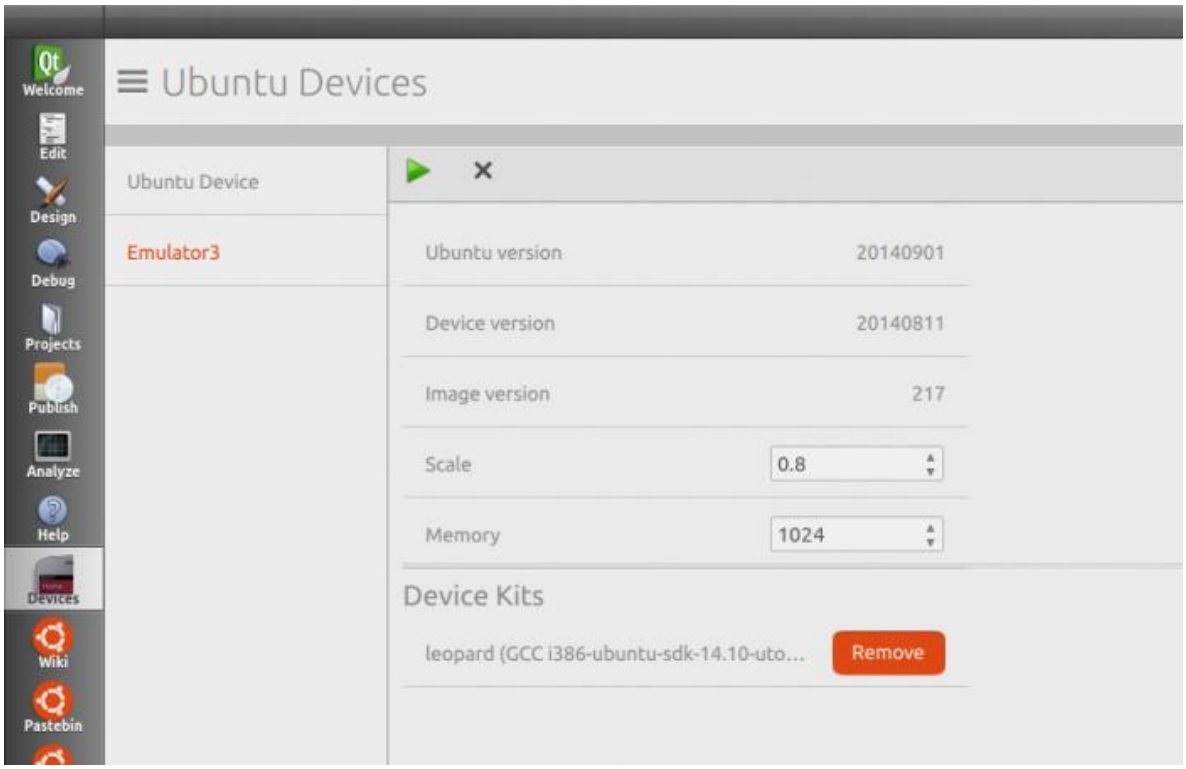



**Ilustración 24. Ejecutar aplicación Ubuntu V**

Cuando se lanza el emulador, este puede ser manejado como cualquier dispositivo físico, siendo capaz de ejecutar aplicaciones usando sus kits compatibles (al lanzar el emulador, se pide una contraseña, que es "0000" por defecto).

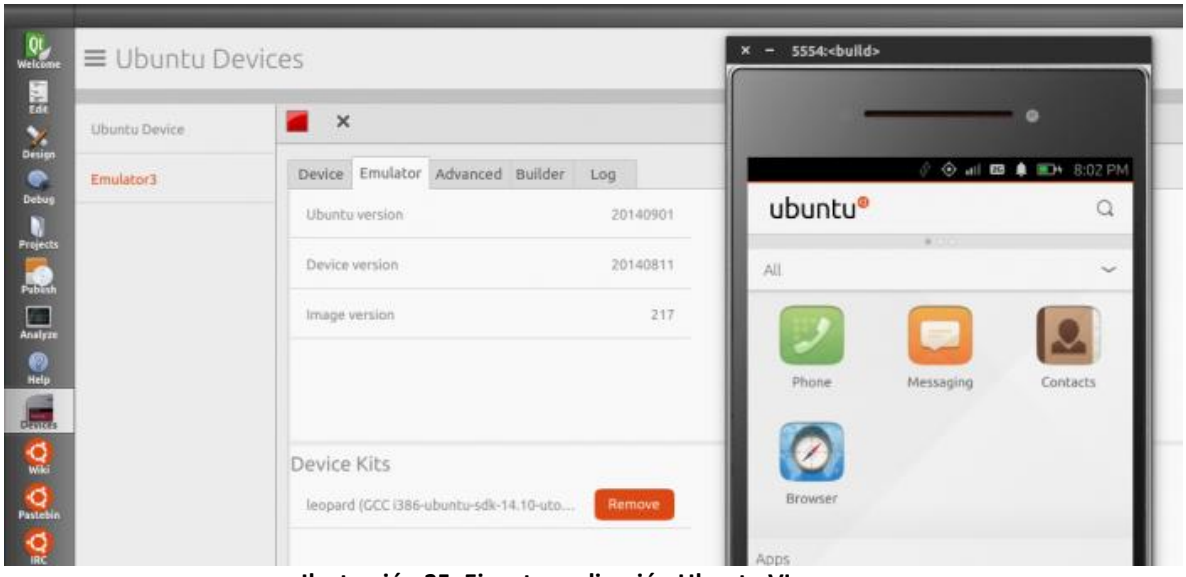



**Ilustración 25. Ejecutar aplicación Ubuntu VI**

Ahora que ya se ha creado el emulador, o bien tenemos un dispositivo conectado, se puede ejecutar la aplicación pulsando en un botón situado en la parte izquierda del SDK. Dicho botón

tiene asociado un punto de color (rojo, naranja o verde) que indica el estado del dispositivo o emulador. En la siguiente ilustración se muestra el botón:

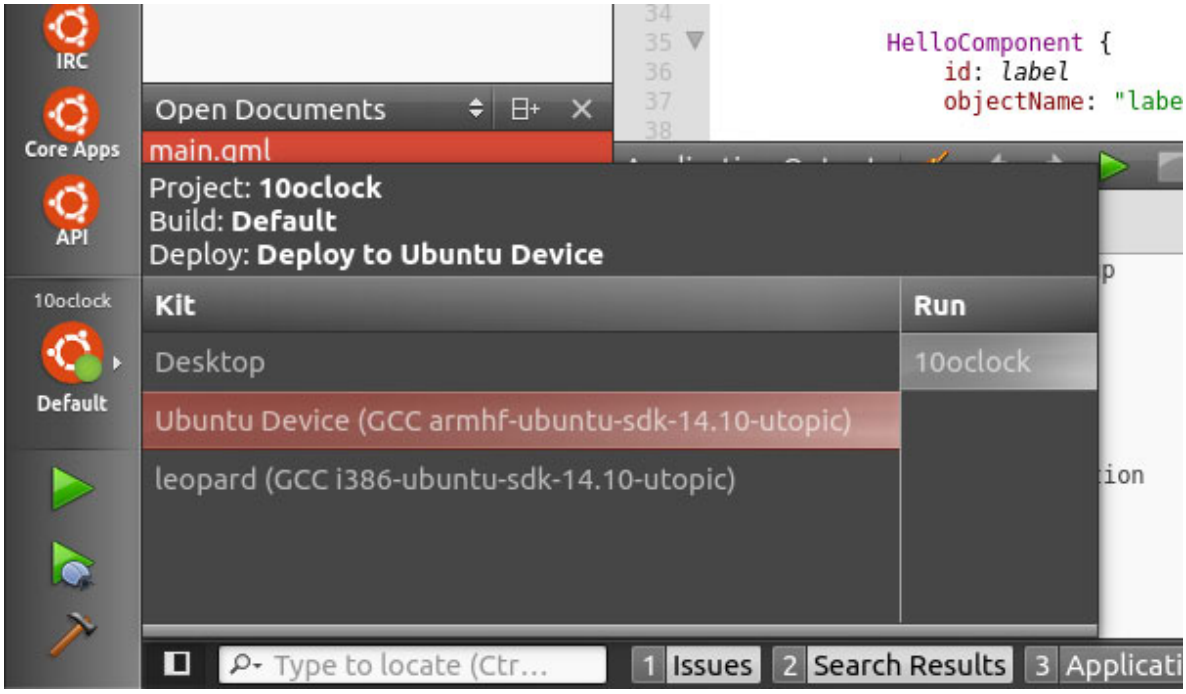



**Ilustración 26. Ejecutar aplicación Ubuntu VII**

### *3.2…2.1 Diferencias entre tipos de dispositivos*

#### 3.2…2.1.1 Escritorio (Desktop)

El SDK monta y ejecuta la aplicación como cualquier otra aplicación de escritorio. Esto supone que la aplicación no será capaz de usar toda la funcionalidad de una plataforma móvil (la pantalla táctil sin ir más lejos). No obstante, esta es la forma más rápida de probar la interfaz de usuario en distintos tamaños de pantalla.

#### 3.2…2.1.2 Emulador (emulator)

Un dispositivo del tamaño de un teléfono en el escritorio. Una buena forma de probar la aplicación con todas las características de la plataforma, sin tener que conectar uno real vía USB. El SDK abrirá una instancia del emulador y lanzará la aplicación en él.

#### 3.2…2.1.3 Dispositivo (device)

La aplicación será cargada en el dispositivo Ubuntu conectado y ejecutada. Es la forma más recomendada para testear una app. Al fin y al cabo, el objetivo del desarrollo de apps es ejecutarlas en terminales físicos.

### 3.2…3 WebApp

Las aplicaciones web de Ubuntu son navegadores independientes que restringen la navegación a ciertos patrones de URLs. Este tipo de aplicaciones son muy fáciles de desarrollar: el SDK de Ubuntu permite convertir cualquier página web en aplicación web en pocos minutos. En este apartado se indican los pasos a seguir para hacerlo, usando como ejemplo "*BrowserQuest"*, un juego RPG multijugador patrocinado por Mozilla.

El primer paso es crear el proyecto tal y como se explica en el apartado Creación de un nuevo proyecto en el Ubuntu SDK, teniendo en cuenta lo siguiente:

- Elegir la plantilla 'Web App'.
- El nombre elegido para el proyecto es 'browserquest'.
- El nombre elegido para la app es 'BrowserQuest'.

Una vez creado el proyecto, se selecciona la página '*Edit*' donde se ve que el SDK ha creado los siguientes ficheros (nótese que los nombres pueden variar en función de los nombres de proyecto y app elegidos):

- BrowserQuest.apparmor: las políticas de seguridad (*Security Policy Groups*) que la aplicación necesita para funcionar en el dispositivo.
- BrowserQuest.desktop: información que Ubuntu necesita para lanzar la aplicación y mostrarla al usuario.
- Browserquest.png: el icono de la aplicación. Aparece uno por defecto que se sustituirá más adelante.
- Manifest.json: información necesaria para identificar la aplicación y declararla al sistema.
- README: un archivo con información para el usuario sobre configuraciones de la aplicación.

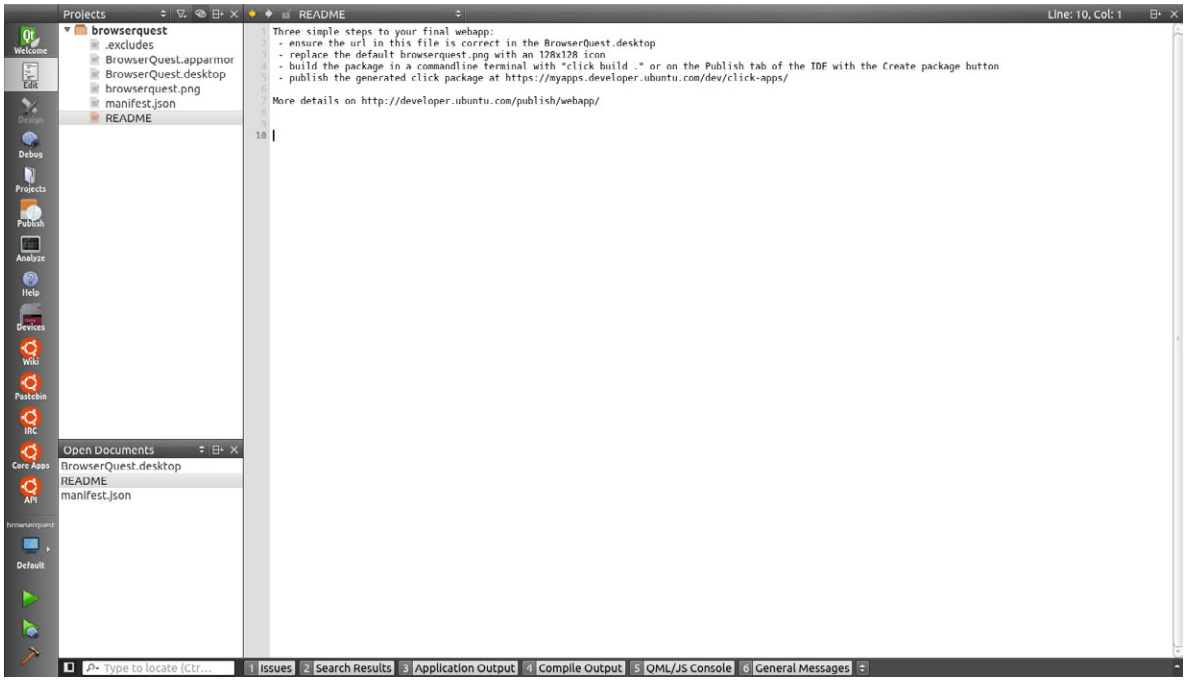



**Ilustración 27. Aplicación Web App - ficheros creados por defecto**

El siguiente paso es reemplazar el icono por defecto por uno que identifique a nuestra app. El tamaño mínimo para que se vea bien es de 256x256. En este caso, el icono elegido es el siguiente:

**Ilustración 28. Aplicación Web App - icono de la app**

Lo último para terminar de desarrollar la aplicación web es definir la URL en el archivo *.desktop*, teniendo la posibilidad de añadir argumentos con los que definir los patrones de URLs a los que la aplicación va a tener acceso. En la línea *Exec* es donde se define todo esto. Con la siguiente línea:

*Exec=webapp-container --enable-back-forward --store-session-cookies*
*--webappUrlPatterns=https?://browserquest.mozilla.org/* http://browserquest.mozilla.org*

Tenemos acceso al juego BrowserQuest y a toda URL que cumpla la regla 'https?://browserquest.mozilla.org/*'.

Se pueden añadir más patrones a la aplicación. Por ejemplo, si se desea que ésta tenga acceso para navegar en el resto de '*mozilla.org*', basta con incluir lo siguiente:

*Exec=webapp-container --enable-back-forward --store-session-cookies*
*--webappUrlPatterns=https?://browserquest.mozilla.org/**
*https?://mozilla.org/* http://browserquest.mozilla.org*


**Ilustración 29. Aplicación Web App - *BrowserQuest.desktop* I**

Con esto la aplicación está completa, a falta de ejecutarla y comprobar su correcto funcionamiento. Ejecutamos la aplicación pulsando el icono de Ubuntu situado encima del botón *play* para elegir un dispositivo, y después el botón *play* (se explica en detalle en el apartado anterior, Ejecutar aplicaciones en el Ubuntu SDK).

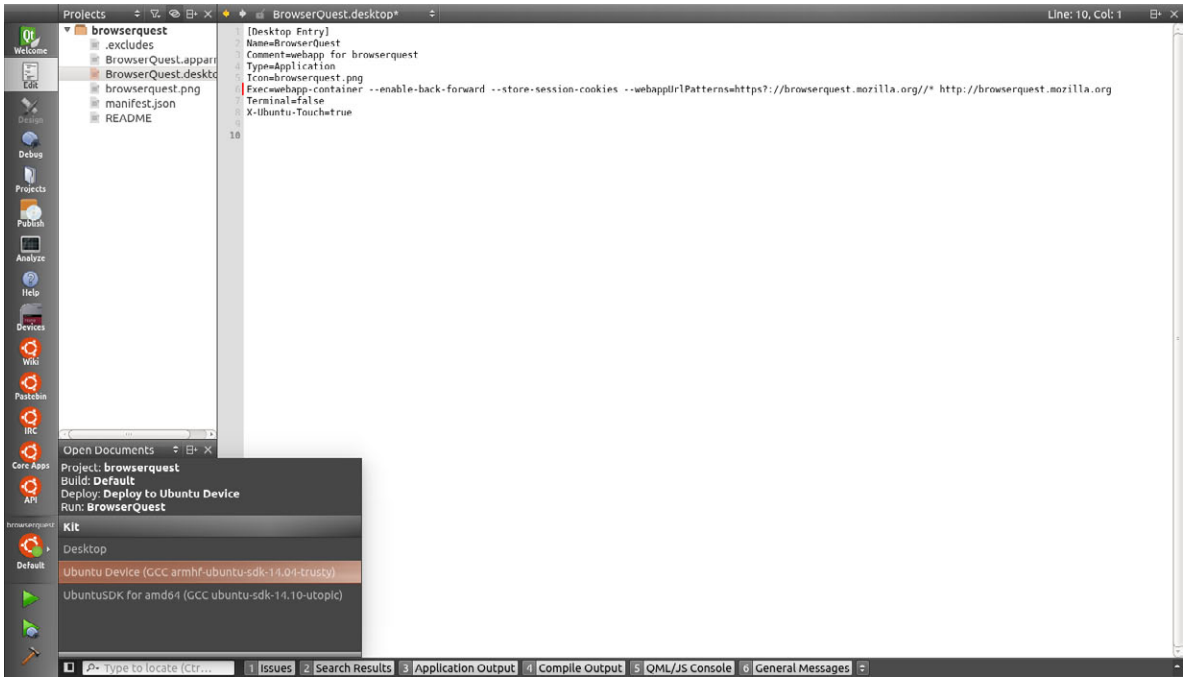



**Ilustración 30. Aplicación Web App - *BrowserQuest.desktop* II**

Si todo va bien, la aplicación se estará cargando en el dispositivo y se debería abrir pasando unos segundos, mostrando la pantalla de inicio del juego:

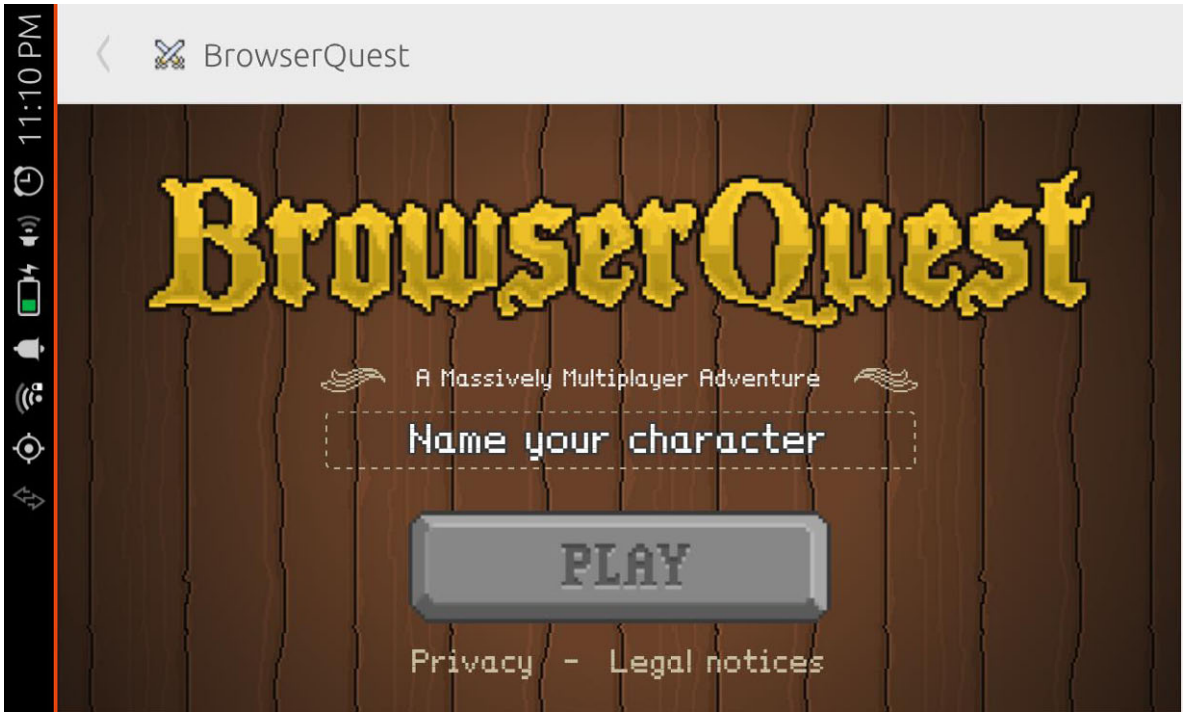



**Ilustración 31. Aplicación Web App - Interfaz principal I**

Las aplicaciones web por defecto cargan más funcionalidad que la necesaria para la mayoría de casos. Esto se puede arreglar cambiando algunos parámetros, con el objetivo de eliminar las flechas de navegación y cambiando a pantalla completa:

*Exec=webapp-container --fullscreen --store-session-cookies*
*--webappUrlPatterns=https?://browserquest.mozilla.org/* http://browserquest.mozilla.org*


**Ilustración 32. Aplicación Web App - Interfaz principal II**

Hay muchos más parámetros que permiten configurar la aplicación. Se puede ver la lista completa en un terminal con la línea '*webapp-container –help*'.

### 3.2…4 Scopes

Un *scope* es una vista adaptada para un conjunto de datos, que pueden usar diseños personalizados. Se puede integrar con las cuentas personales (email, redes sociales,…), dividir el contenido en categorías y agregarse en otras (por ejemplo, un *scope* sobre compras puede generarse a partir varios *stores*) gracias a su flexibilidad.

En este apartado se van a describir los pasos a seguir para desarrollar un *scope* en C++ con el SDK de Ubuntu.

El primer paso es crear un proyecto con la plantilla '*Unity Scope*' (más detalles en Creación de un nuevo proyecto en el Ubuntu SDK).

Este tipo de proyectos es más complejo que resto de los de Ubuntu, por lo que lo que se va a hacer es descargar un proyecto ya existente (un scope sobre el tiempo meteorológico) y cambiarlo para que recupere los datos de SoundCloud. El proyecto se puede descargar desde el terminal mediante la siguiente línea:

*bzr branch lp:~davidc3/ubuntu-sdk-tutorials/scope-tutorial-soundcloud-qjson*

El proyecto contiene una gran cantidad de ficheros, por lo que a lo largo de este apartado se van a describir tan solo aquellos más importantes.

- manifest.json contiene información relativa al sistema y que se almacenará en el paquete resultante. Este archivo normalmente se crea una sola vez de cara a futuros desarrollos.
- <scope>.apparmor contiene las políticas de seguridad de la aplicación.
- Data/<appid>.ini permite personalizar la aplicación (iconos, fondos de pantalla, colores,…).
- Include/api/config.h contiene la configuración HTTP. El primer cambio será llamar a la API de SoundCloud.

  *std::string apiroot { "https://api.soundcloud.com" };*

- Include/api/client.h, /scope/scope.h, /scope/query.h, /scope/preview.h. El resto de las cabeceras C++. El cambio a realizar es sobre client.h, para adaptarlo a la estructura de la API de SoundCloud. El código a incluir es el siguiente: Aplicación Scope – Parte I.
- Src/api/client.cpp: Aplicación Scope – Parte II. El cliente de la API. Proporciona una separación entre el código del scope y el acceso HTTP de la API. Su único objetivo es recuperar datos de SoundCloud.
- Src/scope/scope.cpp: Aplicación Scope – Parte III. Este archivo define una clase de tipo '*unity::scopes::ScopeBase*' que proporciona la API del punto de entrada que el cliente usa para interactuar con el scope. Implementa los métodos *start* y *stop*, y *search* y *preview*, métodos que normalmente se dejan sin modificar, y así se hace para esta aplicación.
- Src/scope/query.cpp: Aplicación Scope – Parte IV. Aquí se mandan las *queries* al cliente API, transformando los resultados devueltos en 'cartas', los elementos que se dispondrán en el layout de la aplicación. Este archivo define una clase del tipo '*unity::scopes::SearchQueryBase*'. Esta clase genera resultados de búsquedas de una *query* que proporciona el cliente y se los devuelve como una respuesta al cliente:
  - Recibe la query desde el cliente.
  - Recibe el objeto respuesta desde el cliente.
  - Envía la consulta a la API del cliente.
  - Crea categorías de los resultados de la búsqueda.
  - Combina cada resultado con su categoría.
  - Inserta los resultados categorizados en el objeto respuesta y se lo devuelve al cliente.
- Src/scope/preview.cpp: Aplicación Scope – Parte V. Este archivo define una clase del tipo '*unity::scopes::PreviewQueryBase*'. Esta clase define los *widgets* y *layouts* usados por cada resultado de búsqueda durante la fase de *preview*:
  - Define los *widgets* usados en las *previews*.
  - Mapea campos del *widget* con campos de datos en cada resultado.
  - Define *layouts* con diferentes números de columnas, dependiendo del tamaño de la pantalla.
  - Asigna *widgets* con columnas por cada *layout*.
  - Recibe un objeto respuesta e inserta los *widgets* y *layouts* en él para que lo use el cliente.

Una vez explicados los archivos principales y básicos de una aplicación scope, se pasa a desarrollar la aplicación específica de SoundCloud.

### *3.2…4.1 Consulta*

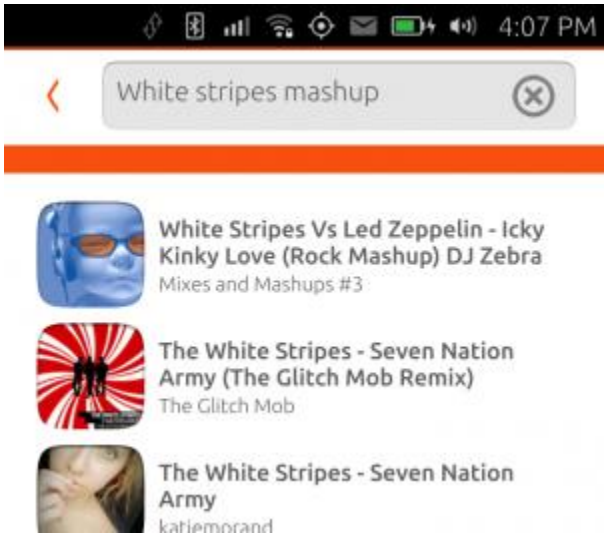



**Ilustración 33. Ejemplo de scope con búsqueda de canciones**

En el archivo 'src/scope/query.cpp', se ve fácilmente de dónde recibe la consulta el scope. Cuando se abre el scope, esta consulta está vacía, por lo que el primer paso será definirla, modificando la función '*Query::run*' con el siguiente código: Aplicación Scope – Parte VI. En este ejemplo se hace una búsqueda del tema "*blur cover*".

### *3.2…4.2 Generación de resultados de búsqueda*

El archivo '*api/client.cpp*' es el que se ha de modificar para recuperar los resultados desde SoundCloud. Para ello, se usa la librería *net-cpp*, aunque se puede usar otra sin problema. Esta librería proporciona un método *get* para manejar cabeceras y errores HTTP, trata las respuestas y devuelve objetos JSON, lo necesario para este ejemplo.

La URI base es traída desde la cabecera de configuración, por lo que solo es necesario añadir el resto de la ruta y de los parámetros:

*get( { "tracks.json"}, { { "client_id", "apigee" }, { "q", query } }, root);*
*// https://api.soundcloud.com/tracks.json?client_id=apigee&q=<query>*

Después, se recorren todos los resultados presentes en el objeto JSON raíz y se extraen los necesarios de la siguiente forma: Aplicación Scope – Parte VII.

### *3.2…4.3 Renderizadores de categorías*

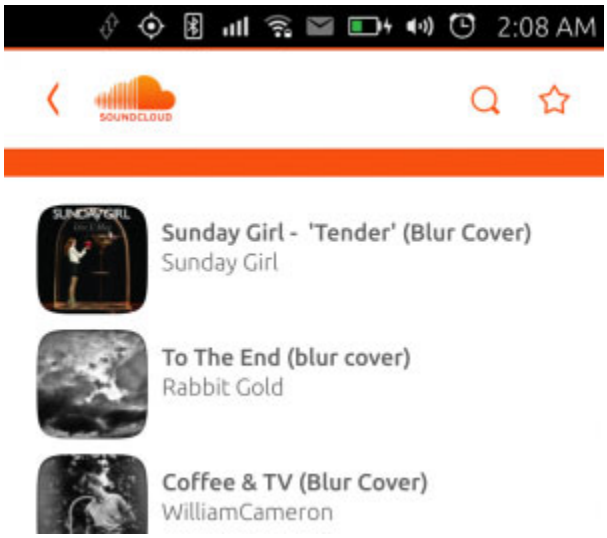



**Ilustración 34. Ejemplo de layout por categorías**

Cada resultado necesita ser mostrado dentro de una categoría. En términos de interfaz de usuario, una categoría puede proporcionar un título a la lista de resultados y un *layout* específico para definir la posición y apariencia de cada resultado.

Los renderizadores son creados como objetos JSON, que tienen dos campos de mayor interés: *templates* y *components*.

El primer paso es modificar el archivo 'src/scope/query.cpp': Aplicación Scope – Parte VIII. Esto muestra una lista simple de resultados con un estilo similar al de muchos *scopes*.

Ahora, en la parte try del método '*Query::run*', se registra la categoría en el objeto respuesta: Aplicación Scope – Parte IX.

### *3.2…4.4 Resultados*

En el scope de SoundCloud, cada resultado consta de:

- Una URI: el link a la canción.
- Una categoría: determina dónde y cómo mostrar el resultado en la interfaz de usuario.
- Un título: el nombre de la canción.
- Un artista: el nombre de la banda/artista.
- Una imagen: la tapa del álbum/canción.

Hay que asegurarse que todos los campos definidos en el *template* estén presentes en el resultado, aunque vengan vacíos, para que estos no sean descartados.

Todavía en el archivo 'src/scope/query.cpp', en la parte try del método '*Query::run*', hay que recorrer la lista de canciones y crear un tipo '*unity::scope::CategorisedResult*' en cada una.

### *3.2…4.5 Previsualizaciones*

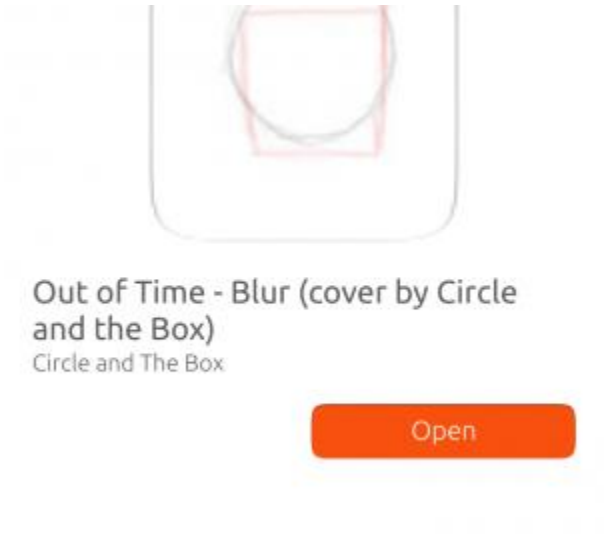



**Ilustración 35. Ejemplo de preview**

La previsualización necesita generar *widgets* y conectar sus campos a los campos de datos del objeto *CategorisedResult*.

También debería generar *layouts* para manejar diferentes entornos visuales. La idea es que tan solo el cliente sepa el contexto del *layout*. El cliente piensa en el contexto visual en términos de número de columnas disponibles. El scope define donde poner los widgets en función de este número de columnas.

#### 3.2…4.5.1 Widgets de previsualización

Hay un lote de *widgets* de previsualización predefinidos. Cada uno tiene un tipo que se usa para crearlo. También tienen campos adicionales que varían por cada tipo.

La aplicación desarrollada en este punto usa tres tipos de *widgets* de previsualización:

- Header: tiene un campo título y subtítulo.
- Image: tiene un campo fuente para recuperar el contenido.
- Actions: usados para proporcionar el botón “Open” y la URI a cargar cuando el usuario pulsa en la previsualización.

De la siguiente forma se crea el widget header en el método ‘*Preview::run*’ del archivo ‘src/scope/preview.cpp’:

*sc::PreviewWidget w_header("headerId", "header");*

- El primer parámetro es el ID. El ID se usa para asignar el *widget* a diferentes *layouts*.
- El segundo parámetro es el tipo del *widget* de previsualización.

Después de la creación del *widget*, sus campos se rellenan con los datos del objeto CategorisedResult, procesados por el cliente. En el caso de este *widget*, se rellenan los campos *title* y *subtitle*.

Hay disponibles dos métodos para insertar los datos en el *widget*:

- Add_attribute_value(FIELD, VALUE): el método más sencillo para inserter datos en el *widget*.
- Add_attribute_mapping(FIELD, CR_FIELD): este método se usa para rellenar los datos desde el objeto CategorisedResult siendo procesados.

Como ya se ha explicado, el *widget* se rellena con los datos procesados de CategorisedResult, por lo que se usa el segundo método:

- *w_header.add_attribute_mapping("title", "title");*
- *w_header.add_attribute_mapping("subtitle", "artist");*

Por tanto, el código resultante es el siguiente para la parte del widget: Aplicación Scope – Parte X.

Y ahora, se puede devolver al cliente con el objeto respuesta:

*reply->push( { w_art, w_header, w_actions });*

Los widgets, llegados a este punto, han sido creados, rellenados y devueltos al cliente, pero este debe saber dónde ponerlos, además de manejar la disposición en diferentes contextos.

#### 3.2…4.5.2 Generating Layouts

En esta aplicación se definen dos layouts: una con una única columna, y otra con dos:

*sc::ColumnLayout layout1col(1), layout2col(2);*

El cliente tiene que conocer cómo manejar esto, aunque parece evidente que el *layout* con una columna se use con la pantalla orientada verticalmente, y el otro *layout* cuando se gire el terminal.

También se ha de definir donde van a ir cada uno de los tres *widgets* en cada uno de estos *layouts*.

En el layout de una columna, todos los widgets van en esa única columna:

*layout1col.add_column( { "imageId", "headerId", "actionsId"});*

En el layout de dos columnas se inserta, por ejemplo, la imagen en la primera columna, y los otros dos en la otra columna:

*layout2col.add_column( { "imageId" });*
*layout2col.add_column( { "headerId", "actionsId" });*

Por último, se registran los *layouts* en el objeto respuesta:

*reply->;register_layout({layout1col, layout2col});*

### *3.2…4.6 Personalización*

Por defecto, el scope tiene el siguiente aspecto:


**Ilustración 36. Interfaz principal aplicación scope**

Las opciones de visualización se pueden cambiar en '*data/<appid>.ini*'. La mayoría de ellas son auto explicativas:

```
[ScopeConfig]
DisplayName = SoundCloud
Description = This is a SoundCloud scope doing SoundCloud things
Art = screenshot.png
Author = Firstname Lastname
Icon = icon.png

[Appearance]
PageHeader.Logo = logo.png
PageHeader.background = color:///#FFFFFF
PageHeader.ForegroundColor = #F8500F
BackgroundColor = #FFFFFF
PageHeader.DividerColor = #F8500F
PreviewButtonColor = #F8500F
```

También se puede cambiar el logo de la aplicación, descargando el que se desee y guardándolo como '*data/logo.png*'.

Este es el aspecto que tendría el scope con los ajustes visuales indicados arriba y con el logo de SoundCloud:


**Ilustración 37. Interfaz aplicación scope - estilo personalizado I**


**Ilustración 38. Interfaz aplicación scope - estilo personalizado II**

Finalmente, el scope de SoundCloud está completo. Se puede ejecutar en el SDK presionando el botón *Start*, seleccionando la forma de despliegue (escritorio, emulador o dispositivo). Si todo funciona correctamente, se puede empaquetar y distribuir (Empaquetar la aplicación y publicarla).

### 3.2…5 QML

Siguiendo los pasos descritos a continuación, se desarrolla un sencillo conversor de moneda con la tecnología QML:

- En el SDK de Ubuntu, pulsar *Ctrl+N* para crear un nuevo proyecto, o ir a *File -> New Project*.
- En *Projects -> Ubuntu*, seleccionar la plantilla "App with Simple UI" y después pulsar en *Choose…*
- Seleccionar un nombre para el proyecto y la ruta en la que crearlo, o marcar la casilla de *default*; a continuación, clicar en *Next* y después en *Finish*.

El contenido está estructurado con una sección de cabecera en la que se incluyen todos los *import* - en el caso de esta aplicación, se importa la librería estándar para aplicaciones QML y los componentes y tipos de Ubuntu – y el cuerpo principal. Se declara el MainView, el componente más esencial, que actúa como raíz de la aplicación. Este ya contiene el *toolbar* estándar y el *header*. Las propiedades que definen esta sección son múltiples y variadas, habiendo usado las descritas a continuación para el desarrollo de esta aplicación.

Con una sintaxis similar a JSON, se definen las propiedades asignándole un id con el que identificarle, y después declarando el resto de características mediante la sintaxis 'propiedad: valor'. Las propiedades que se están definiendo aquí son de carácter visual (*width, height, color*) como si se tratase del fichero *css* de una típica aplicación web. El resto de propiedades, aquellas que no se declaran, tendrán el valor asignado por defecto.

Una diferencia respecto a un archivo css es la declaración de las unidades, que en este caso son units.gu. Este devuelve un valor en *pixels* dependiente del dispositivo sobre el que se ejecute, puesto que al trabajar con entornos móviles este factor hay que tenerlo en muy alta consideración por la gran cantidad de dispositivos que existen en el mercado.

Dentro de nuestra vista principal (*MainView*), añadimos un hijo *Page*, un componente que contiene el resto de componentes, entre otros el título de la página (nótese que el título está incluido dentro de la función *i18n.tr()* para que se pueda traducir en función del idioma del dispositivo).

Todavía dentro del componente *Page*, añadimos un objeto *ListModel* (Aplicación QML – Parte II), que contiene un identificador, una lista de ítems con los pares moneda/tasa (con los valores iniciales EUR y 1.0, que serán tomados como referencia para el resto de divisas) y dos funciones *JavaScript* (tecnología que se integra de una forma transparente) que son los *getters* de moneda y tasa.

El siguiente objetivo es cargar datos XML en un modelo de datos, para lo que usamos un objeto *XmlListModel* (Aplicación QML – Parte III) en el que cargarlos. Las propiedades de este componente son: *source*, para indicar la URL de dónde se recogen los datos; *query*, para especificar una consulta *XPath* absoluta con la que crear el modelo; *namespaceDeclarations*, que contiene el espacio de nombres que se emplean en la query; *onStatusChanged*, un manejador que se dispara cuando la señal *Changed* de cualquier propiedad QML cambie, ejecutando una función *JavaScript* que anexe todos los pares moneda/valor cuando se terminen de cargar; dos objetos *XmlRole*, empleados como parámetros de la propiedad *query* con la que se cargaban los modelos.

El siguiente objeto es *ActivityIndicator* (Aplicación QML – Parte IV), usado para mostrar actividad mientras se cargan los datos de tasas. Típicamente se emplean barras de progreso o círculos animados para que el usuario sepa que hay una tarea en ejecución.

Finalmente, encima del componente Page, se incluye la función JavaScript *convert* (Aplicación QML – Parte V) que realiza la conversión de monedas.

En este punto, se ha añadido todo el código *backend* y toca centrarse en la interacción con el usuario. Para ello, se declara un objeto *Component* (Aplicación QML – Parte VI), un bloque reusable que es una combinación de otros componentes y objetos.

Junto con este nuevo componente es necesario importar dos nuevas librerías: *ListItems* y *Popus*. Al arrancar la aplicación con este nuevo cambio, la interfaz principal es la misma. La nueva funcionalidad que se ha incorporado es la selección de monedas, basado en un objeto *Popover* y un *ListView* (el resultado es la típica lista desplegable). El *popover* muestra la selección de monedas (es el contenedor) y el *ListView* muestra la lista de datos (es el contenido).

Hasta ahora se ha configurado la parte backend y se han construido bloques para la conversión de bloques. Lo restante es juntar todo y observar el resultado. Para ello, se define el layout (Aplicación QML – Parte VII) de la aplicación, compuesto de un objeto *Column* (columna) y dos *Rows* (filas), y cada fila contiene a su vez un botón para seleccionar la moneda y un *input* para mostrar o introducir el valor de conversión de la moneda (es decir, según se introduce un valor en un *input*, automáticamente se actualiza el otro con el valor de la conversión). Además, se incluye un botón para limpiar los dos *input*. A continuación se incluye un *mockup* para ilustrar el *layout*:

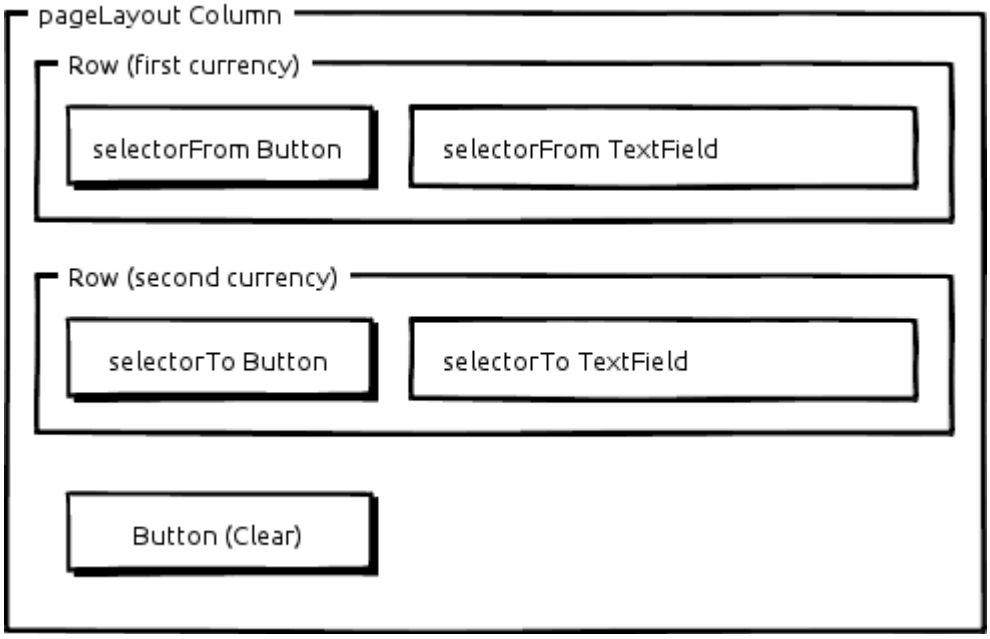



**Ilustración 39. Layout principal de la aplicación QML**

Con esto ya está todo listo para lanzar el SDK y comprobar el correcto funcionamiento de la aplicación. Las dos imágenes siguientes muestran la interfaz principal de la aplicación (Ilustración 40) y la lista desplegable de monedas (Ilustración 41):


**Ilustración 40. Interfaz de la aplicación QML**

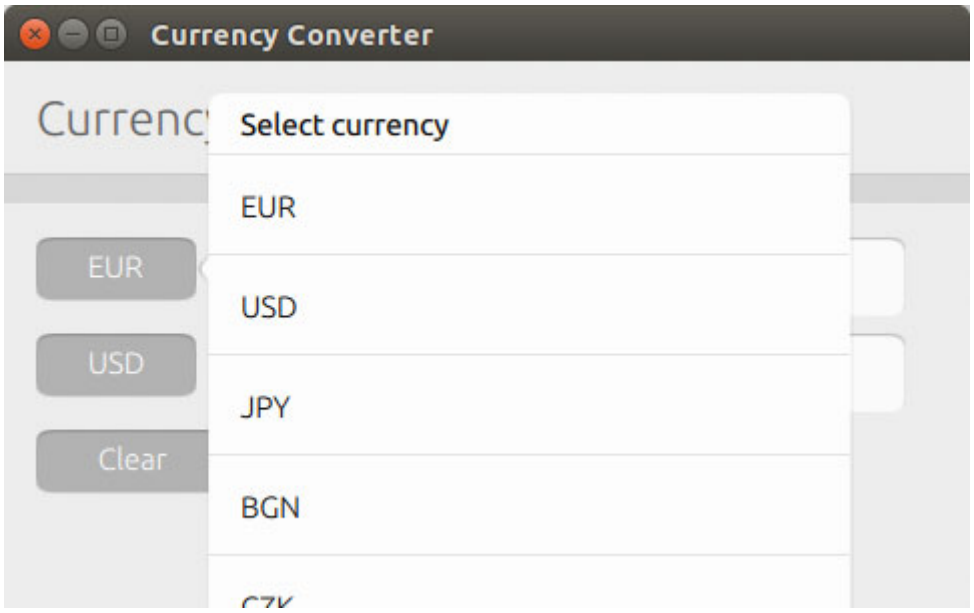



**Ilustración 41. Interfaz de la aplicación QML - lista de monedas**

### 3.2...6 QML con C++

En el apartado anterior se ha desarrollado un sencillo conversor de monedas con QML. Esta tecnología, como se explica en el apartado QML, permite hacer interfaces de usuario muy vistosas con bastante facilidad, pero hay muchas cosas que no se pueden hacer, tales como:

- Tener acceso a funcionalidad fuera del entorno QML/JavaScript.
- Implementar código crítico dónde el código nativo está pensado para la eficiencia.
- Código declarativo de gran complejidad como para implementar en JavaScript.

En cambio, QML permite integrar código C++ con mucha facilidad para suplir esas carencias.

De forma resumida, estos son los pasos a seguir para integrar un componente C++ en un entorno QML:

- Definir una nueva clase derivada de QObject.
- Poner la macro Q_Object en la declaración de la clase para dar soporte a señales y otros servicios del sistema Qt meta-object.
- Declarar cualquier propiedad usando la macro Q_Object.
- En el programa principal del componente C++, llamar a la función *qmlRegisterType()* para registrar el tipo con el motor Qt Quick.

A continuación, se va a desarrollar una pequeña aplicación como ejemplo de la integración de C++ con QML. Esta aplicación consiste en un generador de pares de claves ssh (pública/privada) a través de una sencilla interfaz gráfica. Se presenta al usuario una pantalla con la que seleccionar las opciones deseadas y después se ejecuta la función **ssh-keygen** (https://en.wikipedia.org/wiki/Ssh-keygen) de Linux para generar las claves.

### *3.2…6.1 Generador de claves ssh*

La interfaz de usuario se implementa con QML para darle la apariencia de una sencilla aplicación de escritorio. A continuación se muestra la interfaz:

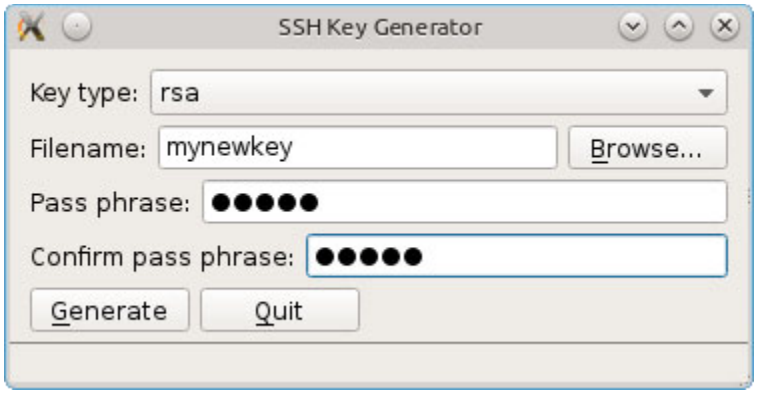



**Ilustración 42. Interfaz principal aplicación QML/C++**

El usuario elige en esta pantalla el tipo de clave, el nombre del archivo dónde generar la clave privada y una contraseña, que debe ser confirmada.

### *3.2…6.2 La clase C++*

Ahora que se tiene la interfaz de usuario, se procede a implementar la funcionalidad *backend*, que se hará con C++.

Lo primero es definir una clase que encapsule la funcionalidad de la generación de la clave. Ésta será mostrada como una clase *KeyGenerator* para la parte QML. El código resultante es el siguiente: Aplicación QML/C++ – Parte I.

Como ya se ha comentado, la clase ha de derivar de QObject. Se deben declarar los atributos y métodos necesarios, y los métodos de notificación se convierten en señales. En el caso del generador de claves, se declaran las propiedades para una lista de tipos de clave, un nombre de archivo y la contraseña. La clave se ha definido como cadena de caracteres, pudiendo hacerlo como un enumerado, pero para simplificar el ejemplo se ha hecho así.

Teniendo los atributos definidos, se declaran los métodos *getters* y *setters* para cada uno de ellos, y para las señales. Además, se añade un slot denominado *generateKey*(). Los slots y las funciones son accesibles desde QML.

#### *3.2…6.3 Implementación C++*

Lo siguiente es desarrollar el generador de claves con C++ una vez que se tiene el modelo. El código fuente, del archivo denominado '*KeyGenerator.cpp*', es el siguiente: Aplicación QML/C++ – Parte II.

El constructor inicializa algunas de las variables. El destructor no hace nada, pero se deja declarado para futuras ampliaciones.

Las funciones *getters* simplemente devuelven el valor de la correspondiente variable privada. Los *setters*, por su parte, los insertan, prestando atención a que el nuevo valor sea diferente del anterior, y de ser así, emitir la correspondiente señal.

La función con mayor complejidad es *generateKey()*: comprueba algunas variables y crea un objeto QProcess para ejecutar el programa externo *ssh-keygen*. Cuando termina, se emite una señal con un argumento booleano que indica que la señal ha sido creada satisfactoriamente o no.

#### *3.2…6.4 Código QML*

El código QML se incluye en '*main.qml*': Aplicación QML/C++ – Parte III. Este archivo genera la interfaz de la aplicación anteriormente mencionada (Ilustración 42. Interfaz principal aplicación QML/C++).

Este código es bastante extenso, pero se puede seguir fácilmente como se va creando el *layout* de la aplicación.

La parte interesante es la integración con C++. Se importa la librería 'com.ics.demo' versión 1.0, la cual permite crear un nuevo tipo *KeyGenerator* que sea accesible, pudiendo tener acceso a las propiedades C++ como propiedades QML, llamando a sus métodos y actuando ante sus

señales. Lo único necesario es registrar el nuevo tipo con el motor QML mediante la siguiente llamada:

*qmlRegisterType<KeyGenerator>("com.ics.demo", 1, 0, "KeyGenerator");*

El último paso es desarrollar el programa principal para ejecutar el código QML desde un ejecutable, declarando, entre otros, los objetos QGuiApplication y QQuickView.

El código fuente del programa principal es el siguiente: Aplicación QML/C++ – Parte IV.

Típicamente, una aplicación QML usa el programa *qmlscene* para ejecutar el código, pero un módulo escrito en C++ no lo permite, motivo por el que se incluye este archivo *main*, que realmente está basado en *qmlscene*.

#### *3.2…6.5 Conclusiones*

La aplicación QML/C++ ya está completa en este punto, y se ha visto lo sencilla que ha sido la integración de las dos tecnologías. Básicamente ha consistido en el desarrollo de un componente escrito en C++ con las propiedades, métodos y *slots* adecuados. A pesar de la cantidad de cosas que se pueden hacer con QML, se demuestra la utilidad de C++ para el desarrollo de aplicaciones menos triviales.

### 3.2…7 Apache Cordova con HTML5

El último tutorial describe los pasos a seguir para desarrollar una sencilla aplicación HTML5 que usa *Apache Cordova* y su API para emplear la cámara del dispositivo.

La aplicación a desarrollar permite lo siguiente:

- Proporciona un botón "*Take a picture*".
- Cuando se clica el botón, se muestra la cámara de *Cordova*.
- El usuario hace una foto.
- La foto es devuelta a través de *Cordova* y mostrada en el HTML de la aplicación.

#### *3.2…7.1 Antes de empezar*

*Cordova* es un proyecto *Apache* que provee, entre otras cosas, un *framework* con utilidades para acceder a la funcionalidad del propio dispositivo desde una aplicación HTML5 a través de *plugins* y APIs de *JavaScript*. En este caso, como ya se ha dicho, se va a acceder a la cámara.

Antes de empezar, se debe configurar el equipo para poder utilizar Cordova. Esto se consigue introduciendo las siguientes instrucciones directamente en el terminal:

- Instalar *cordova* desde la PPA de *Ubuntu Cordova*:
  *sudo apt-add-repository ppa:cordova-ubuntu/ppa; sudo apt-get update*
  *sudo apt-get install cordova-cli*
- Crear un *click chroot* para la arquitectura *armhf*:
  *sudo click chroot -a armhf -f ubuntu-sdk-14.10 create*

- Verificar el entorno mediante el despliegue de una aplicación de ejemplo:
  *cordova create test-app com.ubuntu.developer.myid.test-app "Test App"*
  *cd test-app*
  *cordova platform add ubuntu*
  *vi config.xml*
  (En el fichero mostrado, comprobar la línea '<icon src="img/logo.png" />' puesto que debe existir un archivo 'logo.png' en la ruta '/img')
  *cordova build --device*
  *cordova run --device --debug*

Es posible que durante la ejecución de estas instrucciones se produzcan algunos problemas:

- *""""""""* con la instrucción *cordova create*: omitir el final de la instrucción, dejando solo '*cordova create nombreProyecto*'.
- Si falla la instrucción cordova build con una respuesta similar a lo mostrado en la siguiente imagen:

```
Building Phone Application...
click chroot -a armhf -f ubuntu-sdk-14.10 run dpkg-query -Wf'${db:Status-abbrev}' cmake
libicu-dev:armhf pkg-config qtbase5-dev:armhf qtchooser qtdeclarative5-dev:armhf
qtfeedback5-dev:armhf qtlocation5-dev:armhf qtmultimedia5-dev:armhf qtpim5-dev:armhf
libqt5sensors5-dev:armhf qtsystems5-dev:armhf  FAILED
dpkg-query: no packages found matching libicu-dev:armhf
dpkg-query: no packages found matching qtfeedback5-dev:armhf
dpkg-query: no packages found matching qtpim5-dev:armhf
dpkg-query: no packages found matching qtsystems5-dev:armhf
ii ii ii ii ii ii ii ii Command returned 1: schroot -c click-ubuntu-sdk-14.10-armhf --
env DEB_BUILD_GNU_SYSTEM=linux-gnu DEB_BUILD_GNU_TYPE=i686-linux-gnu
DEB_HOST_ARCH_CPU=arm DEB_BUILD_GNU_CPU=i686 DEB_BUILD_ARCH_CPU=i386
DEB_HOST_MULTIARCH=arm-linux-gnueabihf DEB_HOST_ARCH_ENDIAN=little DEB_HOST_GNU_CPU=arm
DEB_HOST_ARCH_BITS=32 DEB_BUILD_ARCH_BITS=32 DEB_HOST_GNU_TYPE=arm-linux-gnueabihf
DEB_BUILD_ARCH=i386 DEB_BUILD_MULTIARCH=i386-linux-gnu DEB_HOST_GNU_SYSTEM=linux-
gnueabihf DEB_BUILD_ARCH_ENDIAN=little DEB_BUILD_ARCH_OS=linux DEB_HOST_ARCH_OS=linux
DEB_HOST_ARCH=armhf dpkg-query '-Wf${db:Status-abbrev}' cmake libicu-dev:armhf pkg-config
qtbase5-dev:armhf qtchooser qtdeclarative5-dev:armhf qtfeedback5-dev:armhf qtlocation5-
dev:armhf qtmultimedia5-dev:armhf qtpim5-dev:armhf libqt5sensors5-dev:armhf qtsystems5-
dev:armhf
```

**Ilustración 43. Error con instrucción cordova build**

  Ejecutar el siguiente comando:
  *sudo click chroot -a armhf -f ubuntu-sdk-14.10 maint apt-get install cmake libicu-dev:armhf pkg-config qtbase5-dev:armhf qtchooser qtdeclarative5-dev:armhf qtfeedback5-dev:armhf qtlocation5-dev:armhf qtmultimedia5-dev:armhf qtpim5-dev:armhf libqt5sensors5-dev:armhf qtsystems5-dev:armhf*

- Si no se tiene un emulador de dispositivo configurado en el entorno, usar el escritorio como emulador, ejecutando sin parámetros los dos últimos comandos:
  *cordova build*
  *cordova run*

### *3.2…7.2 Desarrollo de la aplicación Cordova*

El primer paso es crear el proyecto Cordova mediante los comandos que ya se han usado en el apartado anterior:

- *cordova create cordova-camera com.ubuntu.developer.cordova-camera*
- *cd cordova-camera*

Como ya se ha explicado, si surge algún problema con la creación del proyecto, omitir la última parte debería solucionarlo (*cordova create cordova-camera*).

Lo siguiente es añadir el código de soporte de la plataforma Ubuntu al proyecto:

- *cordova platform add ubuntu.*

Una vez hecho, añadir el plugin para la API de la cámara:

- *cordova plugin add org.apache.cordova.camera*

A continuación, tal y como se ha explicado en el apartado Antes de empezar, añadir un fichero 'logo.png' en la ruta '/img' con el logo de la aplicación.

#### 3.2…7.2.1 Interfaz HTML5

Una vez que se tiene la base de la aplicación, se procede a definir la interfaz gráfica con HTML5, sustituyendo lo incluido por defecto en el proyecto por lo indicado a continuación:

- En *index.html*, añadir las siguientes declaraciones de hojas de estilo en la sección de cabecera (<*head*>) del documento: Aplicación HTML5 Cordova – Parte I.
- En la misma sección, asegurarse de llamar a los dos siguientes ficheros *JavaScript*: Aplicación HTML5 Cordova – Parte II.
- Por último, borrar el cuerpo entero del documento (<*body*>…</ *body* >) y añadir lo siguiente: Aplicación HTML5 Cordova – Parte III.

Esta es una implementación simple de una aplicación Ubuntu HTML5. Declara lo siguiente:

- Un div *mainview* (obligatorio).
- Una cabecera (*header*) con un único *tabitem*: "*Camera*".
- Un div de contenido (*content*) con dos div internos: cargando (*loading*) y cargado (*loaded*).
    - El div cargando se muestra al lanzar la aplicación e incluye un círculo de progreso. Este está oculto cuando *Cordova* esté listo por código *JavaScript* que se verá más adelante.
    - El div cargado se muestra cuando *Cordova* esté listo por *JavaScript* y contiene:
        - Un botón para hacer una foto (*Take a picture*): se crea un evento *listener* para lanzar la cámara *Cordova*.
        - Una imagen (*img*) vacía: cuando la cámara hace una foto, se usa este elemento para mostrar la foto hecha.

Si se lanza la aplicación en este momento, la interfaz mostrada es la siguiente:

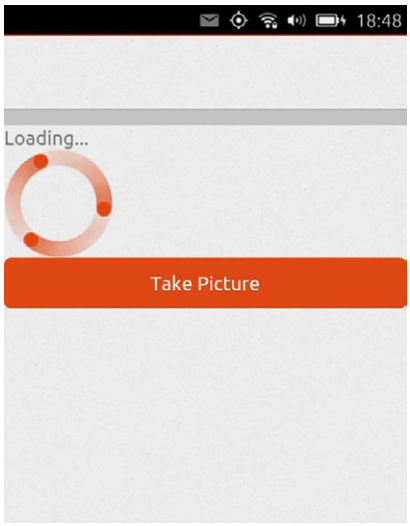



**Ilustración 44. Interfaz de la aplicación HTML5 Cordova – Cargando**

Como se observa, y tal y como se ha detallado anteriormente, este es el div cargando que se muestra hasta que el determinado evento *Cordova* lo reciba por *JavaScript*.

Nota: en Aplicación HTML5 Cordova – Parte II se hace referencia a un archivo *cordova.js* que no aparece en el directorio del proyecto. Este es copiado junto con el resto del código de despliegue de *cordova* durante la fase de construcción (*build*).

En el siguiente apartado se añade el código JavaScript necesario para manejar el evento *Cordova* que oculte el div cargando, muestre el div cargado y proporcione un manejador de eventos para el botón "*Take a Picture*".

### 3.2…7.2.2 Mostrar la cámara Cordova JavaScript

La segunda parte del desarrollo de la aplicación consiste en añadir un manejador de eventos para Cordova, dentro del cual se llamará a la API de la cámara permitiendo al usuario hacer una foto.

Lo primero es renombrar nuestro fichero index.js ('ruta') por app.js, desde donde se hace referencia en Aplicación HTML5 Cordova – Parte II. El primer paso es inicializar un objeto UbuntuUI para configurar las principales partes de la interfaz de usuario: Aplicación HTML5 Cordova – Parte IV.

El siguiente *listener* será disparado en el evento de inicio de la ventana (*window load event*) y prepara el resto de la interfaz de usuario: Aplicación HTML5 Cordova – Parte V.

Dentro de la función declarada se puede instalar un *listener* asociado a pulsar el botón principal, y capturar la imagen con la cámara: Aplicación HTML5 Cordova – Parte VI.

Con esto se tiene el primero paso de lo requerido para nuestra aplicación. Antes de avanzar, se detallan los aspectos del código hasta ahora incluido:

- Se añade un manejador de eventos para el evento de Cordova *deviceready*. Este es llamado cuando el sistema *Cordova* está completamente cargado y preparado, por lo que es un buen lugar donde usar código que use objetos *Cordova*.
- Dentro del manejador del evento *deviceready*, primero se oculta el div cargando y después se muestra el div cargado.
- Después, se obtiene el botón "*Take picture*" con: *UI.button("click")*.
- Su método click(FUNCIÓN) proporciona la FUNCIÓN que se lanza cuando se clica el botón, es decir, el código del manejador del evento del botón.
- Esta función que actúa como manejador del evento llama al método *navigator.camera.getPicture(…)*.
- El objeto *navigator* es el objeto base de *Cordova* y está disponible en el contenedor del *runtime* de HTML5 cuando la aplicación incluye *Cordova*, como en este caso.
- El método getPicture(…) tiene tres argumentos: el nombre de la función a lanzar cuando se hace una foto (aquí se define como onSuccess), el nombre de la función a lanzar cuando falla un intento de hacer una foto (aquí se define como onFail), y otros argumentos opcionales.
- En los argumentos opciones se fija la calidad de imagen, su tamaño, el tipo de la imagen devuelta a DATA_URL, que permite parsear la imagen directamente en JavaScript como una secuencia de datos en base64, y permite la corrección de la orientación.

Como se explica en el párrafo anterior, la función *getPicture* lanza la función *onSuccess* cuando se hace una foto. *Cordova* la lanza y parsea la imagen actual, formateada en el tipo DATA_URL de *Cordova*.

Por tanto, esta aplicación:

- Necesita una función onSuccess.
- Recibe los datos de la imagen parseados.
- Modifica el src de la imagen del html de la aplicación para mostrar la imagen tomada desde los datos parseados.

Esto se realiza con el siguiente código: Aplicación HTML5 Cordova – Parte VII.

Por el otro lado, si la aplicación falla se llama a la función *onFail*, que simplemente manda un mensaje (*log*) a la consola *Cordova*: Aplicación HTML5 Cordova – Parte VIII.

Con todas estas piezas puestas en su sitio, la aplicación debería poder ser lanzada sin problema, permitiendo al usuario tomar una fotografía:

**Ilustración 45. Interfaz de la aplicación HTML5 Cordova – Cámara**

Una vez más, si todo va bien, el sistema traerá de vuelta la aplicación mostrando la imagen tomada debajo del botón.

#### 3.2…7.2.3 Para perfeccionar

Teniendo toda la interfaz gráfica estructurada y el comportamiento definido, se pueden añadir estilos CSS para pulir la aplicación:

- Pintar el botón "*Take Picture*" de naranja.
- Centrarlo.
- Centrar el círculo de progresión del div cargando.

Para ello, se crea el archivo 'app.css' en la ruta raíz con este contenido: Aplicación HTML5 Cordova – Parte IX.

Ahora, en index.html, se añade la siguiente línea para referenciar el archivo de estilos: Aplicación HTML5 Cordova – Parte X.

Finalmente, nuestra aplicación luce de la siguiente manera:

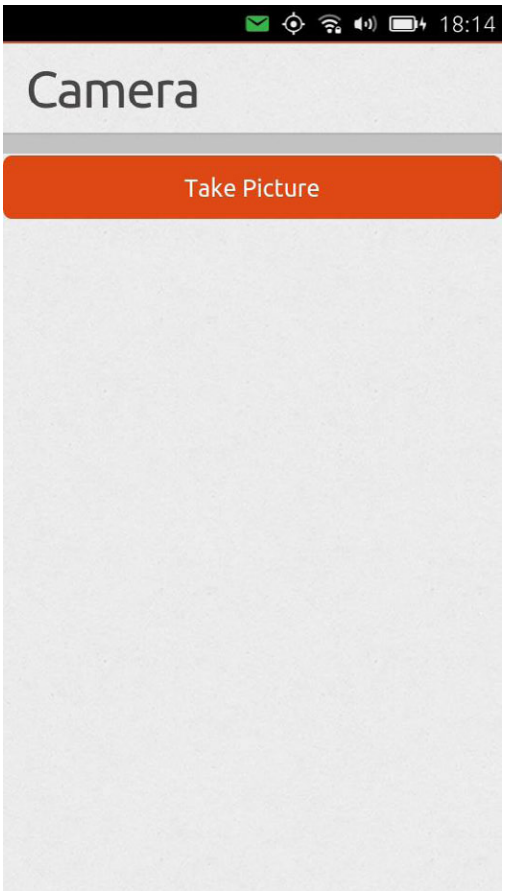
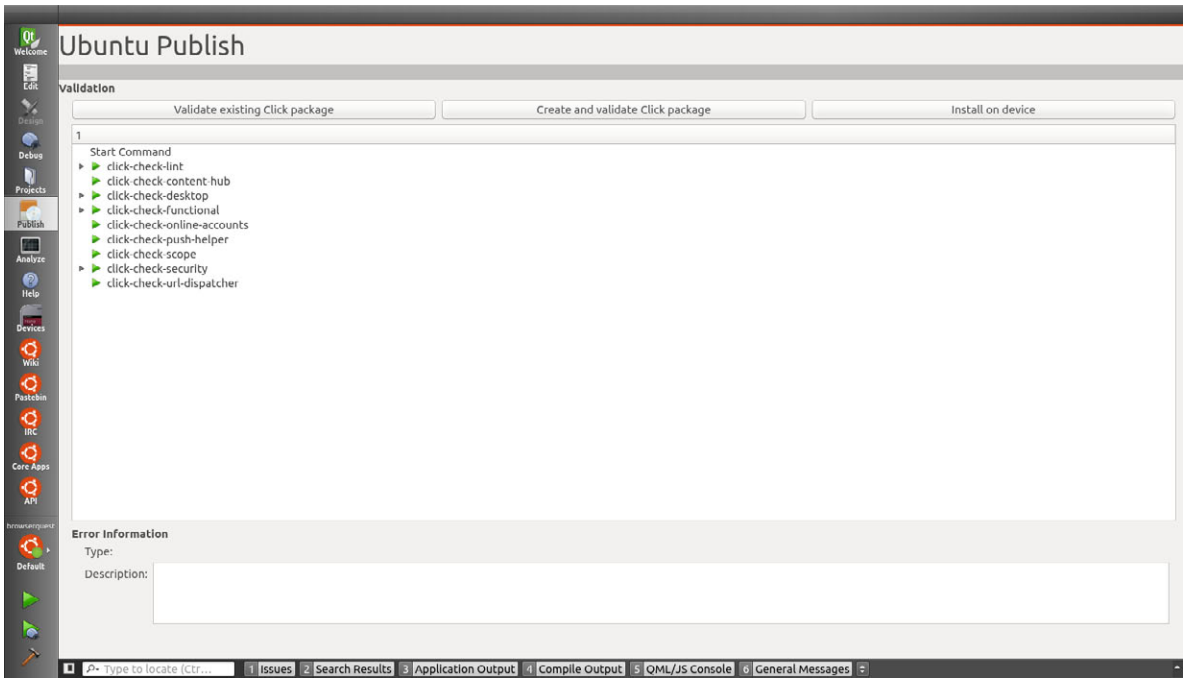



**Ilustración 46. Interfaz de la aplicación HTML5 Cordova – Definitiva con estilos**

### 3.2...8 Empaquetar la aplicación y publicarla

Una vez se crea un proyecto Ubuntu y se desarrolla la aplicación deseada, el siguiente paso es empaquetarla para que otros usuarios la puedan instalar en sus dispositivos, o publicarla en el *store* para que cualquiera tenga acceso.

Para empaquetar la aplicación primero debemos tener el proyecto abierto y después situarse en la página **Publish**. Esta página crear y validar el paquete a publicar. Este se crea en la ruta principal del proyecto.




Si todos los *tests* son válidos (aparecen en verde), se puede comprobar cómo se vería la app en el menú del dispositivo junto a otras mediante el botón ***Install on device*** de la misma página.


En este caso se ha instalado el juego *BrowserQuest* desarrollado anteriormente (*WebApp*).

El siguiente paso es publicar la app en el *store* de Ubuntu. Esto se hace con una serie muy simple de pasos:

- Lo primero es registrarse en el portal de desarrolladores de Ubuntu (developer.ubuntu.com, pestaña *Log in*), prestando mucha atención al namespace elegido para las publicaciones, puesto que debe ser único y no podrá ser cambiado una vez introducido (normalmente se sigue el formato <nombre de app>.<nombre de usuario>).
- Empaquetar la aplicación en el caso de que no se haya hecho.
- Elegir con cuidado el icono de la app (de tamaño 256x256) e incluir pantallazos representativos para conseguir más descargas.
- Añadir los detalles para que los usuarios puedan encontrar la app, tales como la categoría, una descripción o una URL de contacto entre otros.

Con todo esto la aplicación empezará a publicarse, pasando por un proceso automático de revisión. Si todo está correcto, en pocos minutos la app estará disponible en el mercado.

# 4 Conclusión

El objetivo de este apartado consiste en recapitular las aportaciones principales de este proyecto, comentar posibles trabajos futuros en base al desarrollo actual, listar los problemas encontrados durante todas las fases del proyecto y, para finalizar, se aportan opiniones personales sobre el trabajo realizado.

## 4.1 Aportaciones realizadas

La solución adoptada para la resolución de esta problemática en el presente trabajo fin de máster es un documento que detalla paso a paso cómo instalar el sistema operativo Ubuntu en el ordenador personal del usuario final, configurar el entorno, instalar el entorno de desarrollo de aplicaciones móviles para Ubuntu, el Ubuntu SDK, configurarlo, crear proyectos nuevos para cada tipo de aplicación, desarrollarlas, empaquetarlas y publicarlas en el *market* para su libre distribución.

## 4.2 Trabajos futuros

El desarrollo de un tutorial siempre es complicado debido a los distintos conocimientos y capacidades de los usuarios. Por tanto, sería productivo que este documento estuviese distribuido y fuese usado, de cara a solucionar las carencias y especificar el modo de actuación ante los numerosos errores que siempre ocurren al trabajar con equipos informáticos.

Por último, también se podría procesar este documento en un video, que siempre es más útil para el usuario ante las típicas situaciones en las que no se sabe cómo avanzar.

## 4.3 Problemas encontrados

En resumen, los problemas fundamentales han sido dos:

- Adquirir los conocimientos sobre desarrollo de aplicaciones móviles en Ubuntu, de forma que se verifica la utilidad de realizar este documento tutorial.
- Intentar empatizar con diversos tipos de usuarios de cara a exponer las situaciones y problemas, tanto en terreno de lenguaje técnico como formal.

## 4.4 Opiniones personales

El desarrollo del presente trabajo fin de máster ha supuesto una satisfacción general en todos sus aspectos. En primer lugar porque supone la última puerta que cruzar para finalizar el máster, pero sobre todo supone una satisfacción a nivel profesional y personal. Los conocimientos adquiridos a lo largo de este tiempo han sido muy amplios, sobre todo teniendo en cuenta que el desarrollo de aplicaciones móviles está a la orden del día.

El conocimiento adquirido no ha sido sólo de carácter técnico, sino también de un carácter más práctico, al haber afrontado este aprendizaje de una forma más autónoma a la que se

acostumbra durante el máster. El objetivo de esta consiste en formar buenos profesionales, capaces de encontrar soluciones sin esperar que alguien lo haga por nosotros (como funciona la vida misma).

Además, este proyecto ha permitido poner en práctica el conocimiento de varias asignaturas, al haber realizado un proyecto de forma completa, pasando por todas sus fases. Esto también permite conocer la dificultad que suponen y la importancia fundamental de planificar las tareas y de seguirlas todas con cuidado: no todo consiste en desarrollar, sino también asentar bien las bases, definiendo qué se quiera desarrollar, por qué, etc. Cabe destacar que también se ha aprendido la importancia de realizar las tareas con tiempo en vez de dejarlo todo para el último momento. Para complementar lo dicho, se puede generalizar este aspecto: es básico ampliar y diversificar tus conocimientos, de forma que se sepan analizar los problemas y escoger las mejores herramientas para su solución.

En resumen, y para finalizar este apartado, se reitera la satisfacción general conseguida: es gratificante ver cómo partiendo de prácticamente “nada”, es decir, unas ideas no del todo definidas, se consigue elaborar todo un proyecto tangible.

# 5 Bibliografía

[1] "Ubuntu Developer". Canonical Ltd. Ubuntu. 2014. Disponible [Internet]: https://developer.ubuntu.com/en/
[2] "Aplicaciones móviles, la evolución". Wordpress. 2013. Disponible [Internet]: https://upsasoyyo.wordpress.com/2013/09/17/aplicaciones-moviles-la-evolucion/
[3] "Tabla comparative de Sistemas operativos móviles". Slideshare. 2014. Disponible [Internet]: http://es.slideshare.net/kpwalkin/tabla-comparativa-de-sistemas-operativos-mviles

# Anexo I. Fragmentos de código empleados en el desarrollo de las apps

## Aplicación HTML5 Cordova – Parte I

```
<meta name="viewport" content="width=device-width, initial-scale=1, maximum-scale=1, user-scalable=0">
<!-- Ubuntu UI Style imports - Ambiance theme -->
<link href="/usr/share/ubuntu-html5-ui-toolkit/0.1/ambiance/css/appTemplate.css" rel="stylesheet" type="text/css" />

<!-- Ubuntu UI javascript imports - Ambiance theme -->
<script src="/usr/share/ubuntu-html5-ui-toolkit/0.1/ambiance/js/fast-buttons.js"></script>
<script src="/usr/share/ubuntu-html5-ui-toolkit/0.1/ambiance/js/core.js"></script>
<script src="/usr/share/ubuntu-html5-ui-toolkit/0.1/ambiance/js/buttons.js"></script>
<script src="/usr/share/ubuntu-html5-ui-toolkit/0.1/ambiance/js/dialogs.js"></script>
<script src="/usr/share/ubuntu-html5-ui-toolkit/0.1/ambiance/js/page.js"></script>
<script src="/usr/share/ubuntu-html5-ui-toolkit/0.1/ambiance/js/pagestacks.js"></script>
<script src="/usr/share/ubuntu-html5-ui-toolkit/0.1/ambiance/js/tabs.js"></script>
```

## Aplicación HTML5 Cordova – Parte II

```
<!-- Cordova platform API access - Uncomment this to have access to the Javascript APIs -->
<script src="cordova.js"></script>

<!-- Application script and css -->
<script src="js/app.js"></script
```

## Aplicación HTML5 Cordova – Parte III

```
<div data-role="mainview">
    <header data-role="header">
        <ul data-role="tabs">
            <li data-role="tabitem" data-page="camera">Camera</li>
        </ul>
    </header>

    <div data-role="content">
        <div data-role="tab" id="camera">
            <div id="loading">
                <header>Loading...</header>
                <progress class="bigger">Loading...</progress>
            </div>
            <div id="loaded">
                <button data-role="button" class="ubuntu" id="click">Take Picture</button>
```

```
                    <img id="image" src="" />
                </div>
            </div> <!-- tab: camera -->
        </div> <!-- content -->
</div> <!-- mainview -->
```

## Aplicación HTML5 Cordova – Parte IV

```
/**
 * Wait before the DOM has been loaded before initializing the Ubuntu UI layer
 */
var UI = new UbuntuUI();
```

## Aplicación HTML5 Cordova – Parte V

```
window.onload = function () {
        UI.init();

        document.addEventListener("deviceready", function() {
                if (console && console.log)
                        console.log('Platform layer API ready');

                //hide the loading div and display the loaded div
                document.getElementById("loading").style.display = "none";
                document.getElementById("loaded").style.display = "block";
```

## Aplicación HTML5 Cordova – Parte VI

```
                // event listener to take picture
                UI.button("click").click( function() {
                        navigator.camera.getPicture(onSuccess, onFail, {
                                quality: 100,
                                targetWidth: 400,
                                targetHeight: 400,
                                destinationType: Camera.DestinationType.DATA_URL,
                                correctOrientation: true
                        });
                        console.log("Take Picture button clicked");
                }); // "click" button event handler

        }, false); // deviceready event handler

}; // window.onload event handler
```

## Aplicación HTML5 Cordova – Parte VII

```
function onSuccess(imageData) {
        var image = document.getElementById('image');
        image.src = "" + imageData;
        image.style.margin = "10px";
        image.style.display = "block";
}
```

## Aplicación HTML5 Cordova – Parte VIII

```
function onFail(message) {
        console.log("Picture failure: " + message);
}
```

## Aplicación HTML5 Cordova – Parte IX

```
#loading {
        position: absolute;
        left:45%;
}
#loaded {
        display: none;
}
```

## Aplicación HTML5 Cordova – Parte X

```
<link href="app.css" rel="stylesheet" type="text/css"/>
```

## Aplicación Scope – Parte I

```
class Client {
    public:

    /**
    * Our Artist object.
    */
    struct Artist {
        unsigned int id;
        std::string username;
        std::string avatar_url;
    };

    /**
    * Track info, including the artist.
    */
    struct Track {
```

```
        unsigned int id;
        std::string title;
        std::string uri;
        std::string artwork_url;
        std::string stream_url;
        std::string description;
        std::string genre;
        Artist artist;
    };

    /**
    * A list of Track objects.
    */
    typedef std::deque<Track> TrackList;

    /**
    * Track results.
    */
    struct TrackRes {
        TrackList tracks;
    };

    Client(Config::Ptr config);

    virtual ~Client() = default;

    /**
    * Get the track list for a query
    */
    virtual TrackRes tracks(const std::string &query);

    /**
    * Cancel any pending queries (this method can be called from a different thread)
    */
    virtual void cancel();

    virtual Config::Ptr config();

protected:
    void get(const core::net::Uri::Path &path,
             const core::net::Uri::QueryParameters &parameters,
             QJsonDocument &root);
    /**
    * Progress callback that allows the query to cancel pending HTTP requests.
    */
    core::net::http::Request::Progress::Next progress_report(
             const core::net::http::Request::Progress& progress);
```

```
    /**
    * Hang onto the configuration information
    */
    Config::Ptr config_;

    /**
    * Thread-safe cancelled flag
    */
    std::atomic<bool> cancelled_;
};
```

## Aplicación Scope – Parte II

```
#include <api/client.h>

#include <core/net/error.h>
#include <core/net/http/client.h>
#include <core/net/http/content_type.h>
#include <core/net/http/response.h>
#include <QVariantMap>
#include <iostream>

namespace http = core::net::http;
namespace net = core::net;

using namespace api;
using namespace std;

Client::Client(Config::Ptr config) :
   config_(config), cancelled_(false) {
}

void Client::get(const net::Uri::Path &path,
            const net::Uri::QueryParameters &parameters, QJsonDocument &root) {
   // Create a new HTTP client
   auto client = http::make_client();

   // Start building the request configuration
   http::Request::Configuration configuration;

   // Build the URI from its components
   net::Uri uri = net::make_uri(config_->apiroot, path, parameters);
   configuration.uri = client->uri_to_string(uri);

   // Give out a user agent string
   configuration.header.add("User-Agent", config_->user_agent);
```

```
    // Build a HTTP request object from our configuration
    auto request = client->head(configuration);

    try {
        // Synchronously make the HTTP request
        // We bind the cancellable callback to #progress_report
        auto response = request->execute(
                bind(&Client::progress_report, this, placeholders::_1));

        // Check that we got a sensible HTTP status code
        if (response.status != http::Status::ok) {
            throw domain_error(response.body);
        }
        // Parse the JSON from the response
        root = QJsonDocument::fromJson(response.body.c_str());
    } catch (net::Error &) {
    }
}

Client::TrackRes Client::tracks(const string& query) {
    QJsonDocument root;

    // Build a URI and get the contents.
    // The fist parameter forms the path part of the URI.
    // The second parameter forms the CGI parameters.
    get( { "tracks.json"}, { { "client_id", "apigee" }, { "q", query } }, root);
    // https://api.soundcloud.com/tracks.json?client_id=apigee&q=<query>

    // My “list of tracks” object (as seen in the corresponding header file)
    TrackRes result;

    QVariantList variant = root.toVariant().toList();
    for (const QVariant &i : variant) {
        QVariantMap item = i.toMap();
        QVariantMap user = item["user"].toMap();
        string art;
        // If the track artwork is empty, we use the artist picture
        if (item["artwork_url"].toString().toStdString() == "") {
            art = user["avatar_url"].toString().toStdString();
        } else {
            art = item["artwork_url"].toString().toStdString();
        }
        cout << item["title"].toString().toStdString();
        // We add each result to our list
        result.tracks.emplace_back(
            Track {
```

```
            item["id"].toUInt(), item["title"].toString().toStdString(),
            item["uri"].toString().toStdString(), art,
            item["stream_url"].toString().toStdString(),
            item["description"].toString().toStdString(),
            item["genre"].toString().toStdString(),
            Artist {
               user["id"].toUInt(),
               user["username"].toString().toStdString(),
               user["avatar_url"].toString().toStdString()
            }
         }
      );
   }
   return result;
}

http::Request::Progress::Next Client::progress_report(
      const http::Request::Progress&) {

   return cancelled_ ?
            http::Request::Progress::Next::abort_operation :
            http::Request::Progress::Next::continue_operation;
}

void Client::cancel() {
   cancelled_ = true;
}

Config::Ptr Client::config() {
   return config_;
}
```

## Aplicación Scope – Parte III

```
#include <scope/localization.h>
#include <scope/preview.h>
#include <scope/query.h>
#include <scope/scope.h>

#include <iostream>
#include <sstream>
#include <fstream>

namespace sc = unity::scopes;
using namespace std;
using namespace api;
using namespace scope;
```

```
void Scope::start(string const&) {
   config_ = make_shared<Config>();

   setlocale(LC_ALL, "");
   string translation_directory = ScopeBase::scope_directory()
         + "/../share/locale/";
   bindtextdomain(GETTEXT_PACKAGE, translation_directory.c_str());

   // Under test we set a different API root
   char *apiroot = getenv("NETWORK_SCOPE_APIROOT");
   if (apiroot) {
      config_->apiroot = apiroot;
   }
}

void Scope::stop() {
}

sc::SearchQueryBase::UPtr Scope::search(const sc::CannedQuery &query,
                                        const sc::SearchMetadata &metadata) {
   // Boilerplate construction of Query
   return sc::SearchQueryBase::UPtr(new Query(query, metadata, config_));
}

sc::PreviewQueryBase::UPtr Scope::preview(sc::Result const& result,
                                          sc::ActionMetadata const& metadata) {
   // Boilerplate construction of Preview
   return sc::PreviewQueryBase::UPtr(new Preview(result, metadata));
}

#define EXPORT __attribute__ ((visibility ("default")))

// These functions define the entry points for the scope plugin
extern "C" {

EXPORT
unity::scopes::ScopeBase*
// cppcheck-suppress unusedFunction
UNITY_SCOPE_CREATE_FUNCTION() {
   return new Scope();
}

EXPORT
void
// cppcheck-suppress unusedFunction
UNITY_SCOPE_DESTROY_FUNCTION(unity::scopes::ScopeBase* scope_base) {
```

```
    delete scope_base;
}

}
```

## Aplicación Scope – Parte IV

```
#include <boost/algorithm/string/trim.hpp>

#include <scope/localization.h>
#include <scope/query.h>

#include <unity/scopes/Annotation.h>
#include <unity/scopes/CategorisedResult.h>
#include <unity/scopes/CategoryRenderer.h>
#include <unity/scopes/QueryBase.h>
#include <unity/scopes/SearchReply.h>

#include <iomanip>
#include <sstream>

namespace sc = unity::scopes;
namespace alg = boost::algorithm;

using namespace std;
using namespace api;
using namespace scope;


/**
 * Define the layout for the forecast results
 *
 * The icon size is small, and ask for the card layout
 * itself to be horizontal. I.e. the text will be placed
 * next to the image.
 */
const static string TRACKS_TEMPLATE =
    R"(
      {
        "schema-version": 1,
        "template": {
          "category-layout": "grid",
          "card-layout": "horizontal",
          "card-size": "large"
        },
        "components": {
          "title": "title",
```

```
            "art" : {
               "field": "art"
            },
            "subtitle": "artist"
         }
      }
   )";


Query::Query(const sc::CannedQuery &query, const sc::SearchMetadata &metadata,
         Config::Ptr config) :
   sc::SearchQueryBase(query, metadata), client_(config) {
}

void Query::cancelled() {
   client_.cancel();
}


void Query::run(sc::SearchReplyProxy const& reply) {
   try {
      // Start by getting information about the query
      const sc::CannedQuery &query(sc::SearchQueryBase::query());

      // Trim the query string of whitespace
      string query_string = alg::trim_copy(query.query_string());

      Client::TrackRes trackslist;
      if (query_string.empty()) {
         // If the string is empty, provide a specific one
         trackslist = client_.tracks("blur cover");
      } else {
         // otherwise, use the query string
         trackslist = client_.tracks(query_string);
      }

      // Register a category for tracks
      auto tracks_cat = reply->register_category("tracks", "", "",
         sc::CategoryRenderer(TRACKS_TEMPLATE));
      // register_category(arbitrary category id, header title, header icon, template)
      // In this case, since this is the only category used by our scope,
      // it doesn't need to display a header title, we leave it as a blank string.


      for (const auto &track : trackslist.tracks) {
      // Iterate over the trackslist
         sc::CategorisedResult res(tracks_cat);
```

```
            // We must have a URI
            res.set_uri(track.uri);

            // We also need the track title
            res.set_title(track.title);

            // Set the rest of the attributes, art, artist, etc
            res.set_art(track.artwork_url);
            res["artist"] = track.artist.username;
            res["stream"] = track.stream_url;


            // Push the result
            if (!reply->push(res)) {
               // If we fail to push, it means the query has been cancelled.
               // So don't continue;
               return;
            }
         }
      } catch (domain_error &e) {
         // Handle exceptions being thrown by the client API
         cerr << e.what() << endl;
         reply->error(current_exception());
      }
}
```

## Aplicación Scope – Parte V

```
#include <scope/preview.h>

#include <unity/scopes/ColumnLayout.h>
#include <unity/scopes/PreviewWidget.h>
#include <unity/scopes/PreviewReply.h>
#include <unity/scopes/Result.h>
#include <unity/scopes/VariantBuilder.h>

#include <iostream>

namespace sc = unity::scopes;

using namespace std;
using namespace scope;

Preview::Preview(const sc::Result &result, const sc::ActionMetadata &metadata) :
   sc::PreviewQueryBase(result, metadata) {
}
```

```
void Preview::cancelled() {
}

void Preview::run(sc::PreviewReplyProxy const& reply) {
   sc::Result result = PreviewQueryBase::result();

   // Support three different column layouts
   sc::ColumnLayout layout1col(1), layout2col(2);

   // Single column layout
   layout1col.add_column( { "imageId", "headerId", "actionsId"});

   // Two column layout
   layout2col.add_column( { "imageId" });
   layout2col.add_column( { "headerId", "actionsId" });

   // Register the layouts we just created
   reply->register_layout( { layout1col, layout2col });

   // Define the header section
   sc::PreviewWidget w_header("headerId", "header");
   // It has title and a subtitle properties
   w_header.add_attribute_mapping("title", "title");
   w_header.add_attribute_mapping("subtitle", "artist");

   // Define the image section
   sc::PreviewWidget w_art("imageId", "image");
   // It has a single source property, mapped to the result's art property
   w_art.add_attribute_mapping("source", "art");

   // Define the actions section
   sc::PreviewWidget w_actions("actionsId", "actions");
   sc::VariantBuilder builder;
   builder.add_tuple({
      {"id", sc::Variant("open")},
      {"label", sc::Variant("Open")},
      {"uri", result["uri"]}
   });
   w_actions.add_attribute_value("actions", builder.end());

   // Push each of the sections
   reply->push( { w_art, w_header, w_actions });
}
```

## Aplicación Scope – Parte VI

```
void Query::run(sc::SearchReplyProxy const& reply) {
    try {
        // Start by getting information about the query
        const sc::CannedQuery &query(sc::SearchQueryBase::query());

        // Trim the query string of whitespace
        string query_string = alg::trim_copy(query.query_string());

        Client::TrackRes trackslist;
        if (query_string.empty()) {
            // If the string is empty, provide a specific one
            trackslist = client_.tracks("blur cover");
        } else {
            // otherwise, use the query string
            trackslist = client_.tracks(query_string);
        }
(...)
```

## Aplicación Scope – Parte VII

```
   Client::TrackRes Client::tracks(const string& query) {
       QJsonDocument root;

       // Build a URI and get the contents.
       // The fist parameter forms the path part of the URI.
       // The second parameter forms the CGI parameters.
       get( { "tracks.json"}, { { "client_id", "apigee" }, { "q", query }
}, root);
    //https://api.soundcloud.com/tracks.json?client_id=apigee&q=<query>

       // My "list of tracks" object (as seen in the corresponding header
file)
       TrackRes result;

       QVariantList variant = root.toVariant().toList();
       for (const QVariant &i : variant) {
           QVariantMap item = i.toMap();
           QVariantMap user = item["user"].toMap();
           string art;
           // If the track artwork is empty, we use the artist picture
           if (item["artwork_url"].toString().toStdString() == "") {
               art = user["avatar_url"].toString().toStdString();
           } else {
               art = item["artwork_url"].toString().toStdString();
           }
           cout << item["title"].toString().toStdString();
           // We add each result to our list
           result.tracks.emplace_back(
               Track {
                   item["id"].toUInt(),
item["title"].toString().toStdString(),
                   item["uri"].toString().toStdString(), art,
                   item["stream_url"].toString().toStdString(),
                   item["description"].toString().toStdString(),
```

```
                    item["genre"].toString().toStdString(),
                    Artist {
                        user["id"].toUInt(),
                        user["username"].toString().toStdString(),
                        user["avatar_url"].toString().toStdString()
                    }
                }
            );
        }
        return result;
    }
```

## Aplicación Scope – Parte VIII

```
const static string TRACKS_TEMPLATE =
    R"(
        {
            "schema-version": 1,
            "template": {
                "category-layout": "grid",
                "card-layout": "horizontal",
                "card-size": "large"
            },
            "components": {
                "title": "title",
                "art" : {
                    "field": "art"
                },
                "subtitle": "artist"
            }
        }
    )";
```

## Aplicación Scope – Parte IX

```
// Register a category for tracks
auto tracks_cat = reply->register_category("tracks", "", "",
    sc::CategoryRenderer(TRACKS_TEMPLATE));
// register_category(arbitrary category id, header title, header icon,
template)
// In this case, since this is the only category used by our scope,
// it doesn't need to display a header title, we leave it as a blank
string.
```

## Aplicación Scope – Parte X

```
// Define the header section
sc::PreviewWidget w_header("headerId", "header");

// It has title and a subtitle properties
w_header.add_attribute_mapping("title", "title");
w_header.add_attribute_mapping("subtitle", "artist");
```

```
// Define the image section
sc::PreviewWidget w_art("imageId", "image");

// It has a single source property, mapped to the result's art property
w_art.add_attribute_mapping("source", "art");

// Define the actions section
sc::PreviewWidget w_actions("actionsId", "actions");

// Actions are built using tuples with an id, a label and a URI
sc::VariantBuilder builder;
builder.add_tuple({
    {"id", sc::Variant("open")},
    {"label", sc::Variant("Open")},
    {"uri", result["uri"]}
});
w_actions.add_attribute_value("actions", builder.end());
```

## Aplicación QML – Parte I

```
MainView {
    id: root
    // objectName for functional testing purposes (autopilot-qt5)
    objectName: "mainView"

    // Note! applicationName needs to match the "name" field of the click
manifest
    applicationName: "currencyconverter.yourname"

    /*
     This property enables the application to change orientation
     when the device is rotated. The default is false.
    */
    //automaticOrientation: true

    // Removes the old toolbar and enables new features of the new
header.
    useDeprecatedToolbar: false

    width: units.gu(100)
    height: units.gu(75)

    property real margins: units.gu(2)
    property real buttonWidth: units.gu(9)

    Page {
        title: i18n.tr("Currency Converter")

    }
}
```

## Aplicación QML – Parte II

```
ListModel {
    id: currencies
    ListElement {
        currency: "EUR"
        rate: 1.0
    }

    function getCurrency(idx) {
        return (idx >= 0 && idx < count) ? get(idx).currency: ""
    }

    function getRate(idx) {
        return (idx >= 0 && idx < count) ? get(idx).rate: 0.0
    }
}
```

## Aplicación QML – Parte III

```
   XmlListModel {
       id: ratesFetcher
       source: "http://www.ecb.int/stats/eurofxref/eurofxref-daily.xml"
       namespaceDeclarations:                "declare                namespace
gesmes='http://www.gesmes.org/xml/2002-08-01';"
                                +"declare    default    element    namespace
'http://www.ecb.int/vocabulary/2002-08-01/eurofxref';"
       query: "/gesmes:Envelope/Cube/Cube/Cube"

       onStatusChanged: {
           if (status === XmlListModel.Ready) {
               for (var i = 0; i < count; i++)
                   currencies.append({"currency":         get(i).currency,
"rate": parseFloat(get(i).rate)})
           }
       }

       XmlRole { name: "currency"; query: "@currency/string()" }
       XmlRole { name: "rate"; query: "@rate/string()" }
   }
```

## Aplicación QML – Parte IV

```
ActivityIndicator {
    objectName: "activityIndicator"
    anchors.right: parent.right
    running: ratesFetcher.status === XmlListModel.Loading
}
```

## Aplicación QML – Parte V

```
function convert(from, fromRateIndex, toRateIndex) {
    var fromRate = currencies.getRate(fromRateIndex);
```

```
    if (from.length <= 0 || fromRate <= 0.0)
        return "";
    return currencies.getRate(toRateIndex) * (parseFloat(from) /
fromRate);
}
```

## Aplicación QML – Parte VI

```
Component {
    id: currencySelector
    Popover {
        Column {
            anchors {
                top: parent.top
                left: parent.left
                right: parent.right
            }
            height: pageLayout.height
            Header {
                id: header
                text: i18n.tr("Select currency")
            }
            ListView {
                clip: true
                width: parent.width
                height: parent.height - header.height
                model: currencies
                delegate: Standard {
                    objectName: "popoverCurrencySelector"
                    text: currency
                    onClicked: {
                        caller.currencyIndex = index
                        caller.input.update()
                        hide()
                    }
                }
            }
        }
    }
}
```

## Aplicación QML – Parte VII

```
Column {
    id: pageLayout

    anchors {
        fill: parent
        margins: root.margins
    }
    spacing: units.gu(1)

    Row {
        spacing: units.gu(1)
```

```
        Button {
            id: selectorFrom
            objectName: "selectorFrom"
            property int currencyIndex: 0
            property TextField input: inputFrom
            text: currencies.getCurrency(currencyIndex)
            onClicked: PopupUtils.open(currencySelector, selectorFrom)
        }

        TextField {
            id: inputFrom
            objectName: "inputFrom"
            errorHighlight: false
            validator: DoubleValidator {notation:
DoubleValidator.StandardNotation}
            width: pageLayout.width - 2 * root.margins - root.buttonWidth
            height: units.gu(5)
            font.pixelSize: FontUtils.sizeToPixels("medium")
            text: '0.0'
            onTextChanged: {
                if (activeFocus) {
                    inputTo.text = convert(inputFrom.text,
selectorFrom.currencyIndex, selectorTo.currencyIndex)
                }
            }
            function update() {
                text = convert(inputTo.text, selectorTo.currencyIndex,
selectorFrom.currencyIndex)
            }
        }
    }

    Row {
        spacing: units.gu(1)
        Button {
            id: selectorTo
            objectName: "selectorTo"
            property int currencyIndex: 1
            property TextField input: inputTo
            text: currencies.getCurrency(currencyIndex)
            onClicked: PopupUtils.open(currencySelector, selectorTo)
        }

        TextField {
            id: inputTo
            objectName: "inputTo"
            errorHighlight: false
            validator: DoubleValidator {notation:
DoubleValidator.StandardNotation}
            width: pageLayout.width - 2 * root.margins - root.buttonWidth
            height: units.gu(5)
            font.pixelSize: FontUtils.sizeToPixels("medium")
            text: '0.0'
            onTextChanged: {
```

```
                if (activeFocus) {
                    inputFrom.text = convert(inputTo.text,
selectorTo.currencyIndex, selectorFrom.currencyIndex)
                }
            }
            function update() {
                text = convert(inputFrom.text,
selectorFrom.currencyIndex, selectorTo.currencyIndex)
            }
        }
    }

    Button {
        id: clearBtn
        objectName: "clearBtn"
        text: i18n.tr("Clear")
        width: units.gu(12)
        onClicked: {
            inputTo.text = '0.0';
            inputFrom.text = '0.0';
        }
    }
}
```

## Aplicación QML/C++ – Parte I

```
#ifndef KEYGENERATOR_H
#define KEYGENERATOR_H

#include <QObject>
#include <QString>
#include <QStringList>

// Simple QML object to generate SSH key pairs by calling ssh-keygen.

class KeyGenerator : public QObject
{
  Q_OBJECT
  Q_PROPERTY(QString type READ type WRITE setType NOTIFY typeChanged)
  Q_PROPERTY(QStringList types READ types NOTIFY typesChanged)
  Q_PROPERTY(QString filename READ filename WRITE setFilename NOTIFY filenameChanged)
  Q_PROPERTY(QString passphrase READ filename WRITE setPassphrase NOTIFY
passphraseChanged)

public:
  KeyGenerator();
  ~KeyGenerator();

  QString type();
  void setType(const QString &t);
```

```
    QStringList types();

    QString filename();
    void setFilename(const QString &f);

    QString passphrase();
    void setPassphrase(const QString &p);

public slots:
    void generateKey();

signals:
    void typeChanged();
    void typesChanged();
    void filenameChanged();
    void passphraseChanged();
    void keyGenerated(bool success);

private:
    QString _type;
    QString _filename;
    QString _passphrase;
    QStringList _types;
};
#endif
```

## Aplicación QML/C++ – Parte II

```
#include <QFile>
#include <QProcess>
#include "KeyGenerator.h"

KeyGenerator::KeyGenerator()
    : _type("rsa"), _types{"dsa", "ecdsa", "rsa", "rsa1"}
{
}

KeyGenerator::~KeyGenerator()
{
}

QString KeyGenerator::type()
{
    return _type;
}
```

```
void KeyGenerator::setType(const QString &t)
{
   // Check for valid type.
   if (!_types.contains(t))
      return;

   if (t != _type) {
      _type = t;
      emit typeChanged();
   }
}

QStringList KeyGenerator::types()
{
   return _types;
}

QString KeyGenerator::filename()
{
   return _filename;
}

void KeyGenerator::setFilename(const QString &f)
{
   if (f != _filename) {
      _filename = f;
      emit filenameChanged();
   }
}

QString KeyGenerator::passphrase()
{
   return _passphrase;
}

void KeyGenerator::setPassphrase(const QString &p)
{
   if (p != _passphrase) {
      _passphrase = p;
      emit passphraseChanged();
   }
}

void KeyGenerator::generateKey()
{
   // Sanity check on arguments
   if (_type.isEmpty() or _filename.isEmpty() or
```

```
    (_passphrase.length() > 0 and _passphrase.length() < 5)) {
    emit keyGenerated(false);
    return;
  }

  // Remove key file if it already exists
  if (QFile::exists(_filename)) {
    QFile::remove(_filename);
  }

  // Execute ssh-keygen -t type -N passphrase -f keyfileq
  QProcess *proc = new QProcess;
  QString prog = "ssh-keygen";
  QStringList args{"-t", _type, "-N", _passphrase, "-f", _filename};
  proc->start(prog, args);
  proc->waitForFinished();
  emit keyGenerated(proc->exitCode() == 0);
  delete proc;
}
```

## Aplicación QML/C++ – Parte III

```
// SSH key generator UI

import QtQuick 2.1
import QtQuick.Controls 1.0
import QtQuick.Layouts 1.0
import QtQuick.Dialogs 1.0
import com.ics.demo 1.0

ApplicationWindow {
  title: qsTr("SSH Key Generator")

  statusBar: StatusBar {
  RowLayout {
    Label {
      id: status
      }
    }
  }

  width: 369
  height: 166

  ColumnLayout {
    x: 10
    y: 10
```

```
// Key type
RowLayout {
   Label {
      text: qsTr("Key type:")
   }
   ComboBox {
      id: combobox
      Layout.fillWidth: true
      model: keygen.types
      currentIndex: 2
   }
}

// Filename
RowLayout {
   Label {
      text: qsTr("Filename:")
   }
   TextField {
      id: filename
      implicitWidth: 200
      onTextChanged: updateStatusBar()
   }
   Button {
      text: qsTr("&Browse...")
      onClicked: filedialog.visible = true
   }
}

// Passphrase
RowLayout {
   Label {
      text: qsTr("Pass phrase:")
   }
   TextField {
      id: passphrase
      Layout.fillWidth: true
      echoMode: TextInput.Password
      onTextChanged: updateStatusBar()
   }

}

// Confirm Passphrase
RowLayout {
   Label {
```

```
            text: qsTr("Confirm pass phrase:")
        }
        TextField {
            id: confirm
            Layout.fillWidth: true
            echoMode: TextInput.Password
            onTextChanged: updateStatusBar()
        }
    }

    // Buttons: Generate, Quit
    RowLayout {
        Button {
            id: generate
            text: qsTr("&Generate")
            onClicked: keygen.generateKey()
        }
        Button {
            text: qsTr("&Quit")
            onClicked: Qt.quit()
        }
    }

}

FileDialog {
    id: filedialog
    title: qsTr("Select a file")
    selectMultiple: false
    selectFolder: false
    nameFilters: [ "All files (*)" ]
    selectedNameFilter: "All files (*)"
    onAccepted: {
        filename.text = fileUrl.toString().replace("file://", "")
    }
}

KeyGenerator {
    id: keygen
    filename: filename.text
    passphrase: passphrase.text
    type: combobox.currentText
    onKeyGenerated: {
        if (success) {
            status.text = qsTr('<font color="green">Key generation succeeded.</font>')
        } else {
            status.text = qsTr('<font color="red">Key generation failed</font>')
```

```
        }
      }
    }

    function updateStatusBar() {
      if (passphrase.text != confirm.text) {
        status.text = qsTr('<font color="red">Pass phrase does not match.</font>')
        generate.enabled = false
      } else if (passphrase.text.length > 0 && passphrase.text.length < 5) {
        status.text = qsTr('<font color="red">Pass phrase too short.</font>')
        generate.enabled = false
      } else if (filename.text == "") {
        status.text = qsTr('<font color="red">Enter a filename.</font>')
        generate.enabled = false
      } else {
        status.text = ""
        generate.enabled = true
      }
    }

    Component.onCompleted: updateStatusBar()
}
```

## Aplicación QML/C++ – Parte IV

```
#include <QApplication>
#include <QObject>
#include <QQmlComponent>
#include <QQmlEngine>
#include <QQuickWindow>
#include <QSurfaceFormat>
#include "KeyGenerator.h"

// Main wrapper program.
// Special handling is needed when using Qt Quick Controls for the top window.
// The code here is based on what qmlscene does.

int main(int argc, char ** argv)
{
    QApplication app(argc, argv);

    // Register our component type with QML.
    qmlRegisterType<KeyGenerator>("com.ics.demo", 1, 0, "KeyGenerator");

    int rc = 0;

    QQmlEngine engine;
```

```
    QQmlComponent *component = new QQmlComponent(&engine);

    QObject::connect(&engine, SIGNAL(quit()), QCoreApplication::instance(), SLOT(quit()));

    component->loadUrl(QUrl("main.qml"));

    if (!component->isReady() ) {
        qWarning("%s", qPrintable(component->errorString()));
        return -1;
    }

    QObject *topLevel = component->create();
    QQuickWindow *window = qobject_cast<QQuickWindow *>(topLevel);

    QSurfaceFormat surfaceFormat = window->requestedFormat();
    window->setFormat(surfaceFormat);
    window->show();

    rc = app.exec();

    delete component;
    return rc;
}
```