Desarrollo de Aplicaciones Móviles Multiplataforma

Alumno: Lic. Lisandro Nahuel Delía
Lic. en Sistemas – UNLP
ldelia@lidi.info.unlp.edu.ar

Director: Mg. Pablo Javier Thomas
Prof. Asociado. - Facultad de Informática – UNLP
pthomas@lidi.info.unlp.edu.ar

Trabajo Final presentado para obtener el grado de Especialista en Ingeniería de Software

Facultad de Informática

Universidad Nacional de La Plata

Marzo 2017

Agradecimientos

A Pablo Thomas, por la confianza depositada día a día.

Al III-LIDI, por permitirme crecer profesionalmente.

A todas aquellas personas, que en forma directa o indirecta, aportaron a este trabajo, y que contribuyen al crecimiento de la Ingeniería de Software como disciplina.

Y especialmente a mi familia por estar siempre a mi lado y alentarme a que continúe con mi formación académica.

Índice de Contenidos

Índice de Figuras

Objetivo

Existen diversos enfoques para desarrollar aplicaciones móviles multiplataforma. A partir de ello, el objetivo de este trabajo final consiste en elaborar un análisis comparativo sobre los enfoques predominantes, y realizar experimentos con el desarrollo de aplicaciones móviles específicas para cada elemento analizado.

Los objetivos específicos de este trabajo son:

a) Realizar un análisis comparativo entre los enfoques de desarrollo de aplicaciones móviles multiplataforma.
b) Llevar a cabo el desarrollo de una aplicación móvil concreta en cada enfoque de desarrollo multiplataforma. Analizar ventajas y desventajas de los enfoques según el experimento.
c) Analizar la performance de las aplicaciones que se generan con cada enfoque de desarrollo multiplataforma.
d) Estudiar los impactos que se producen en el proceso de desarrollo de software según el enfoque a utilizar.

Prefacio

La computación móvil se puede definir como un entorno de cómputo con movilidad física. El usuario de un entorno de computación móvil será capaz de acceder a datos, información u otros objetos lógicos desde cualquier dispositivo en cualquier red mientras está en movimiento [1].

Estos dispositivos tienen características físicas distintivas, entre las cuales se destacan su tamaño, peso, tamaño de pantalla, su mecanismo de ingreso de datos y su capacidad de expansión. Además, tienen un rol esencial los aspectos técnicos, incluyendo el poder de procesamiento, espacio de memoria, autonomía de batería, sistema operativo, entre otros.

El desarrollo de software para dispositivos móviles plantea nuevos desafíos originados en las características únicas de esta actividad. La necesidad de tratar con diversas plataformas, estándares, protocolos y tecnologías de red; las capacidades limitadas, aunque en continua evolución, de los dispositivos y las exigencias de tiempo del mercado, son sólo algunos de los problemas a tratar [2].

Las aplicaciones móviles son generadas en un entorno dinámico e incierto. Generalmente, son pequeñas, no críticas, aunque no menos importantes. Están destinadas a un gran número de usuarios finales y son liberadas en versiones rápidas para poder satisfacer las demandas del mercado [3].

Por todo lo expuesto, el desarrollo de software para dispositivos móviles difiere considerablemente del tradicional [4], y acompaña el crecimiento y evolución de la Ingeniería de Software como disciplina.

Para maximizar su presencia en el mercado, un producto de software debe ejecutarse en la mayor cantidad de dispositivos posible. Una solución consiste en el desarrollo nativo de la aplicación en cada una de las plataformas existentes utilizando el entorno de desarrollo integrado (IDE por sus siglas en inglés), el lenguaje y las herramientas propias de cada plataforma [5]. Sin embargo, al no ser posible la reutilización de código fuente entre diferentes plataformas, el esfuerzo se multiplica y se elevan los costos de desarrollo, actualización y distribución de nuevas versiones [6].

El desarrollo multiplataforma, a diferencia del desarrollo nativo, se centra en el reúso de código. La construcción de aplicaciones web móviles constituye un ejemplo que representa este enfoque. Sin embargo, las limitaciones derivadas de su ejecución dentro de un navegador, ha motivado a los ingenieros de software a dirigir su atención hacia otro tipo de aplicaciones multiplataforma con el que se obtienen resultados más cercanos a las soluciones nativas. En este contexto, existen diversas sub-clasificaciones [5] [7] [8] y es de interés analizar las características inherentes a cada una de ellas, a través de la construcción de un prototipo experimental.

Trabajos publicados

Los resultados parciales del presente trabajo dieron lugar al siguiente capítulo de libro:

- **"An Experimental Analysis of Application Types for Mobile Devices"** [9]
  Lisandro Delía; Nicolás Galdámez; Pablo Thomas, Patricia Pesado
  Computer Science & Technology Series - XIX Argentine Congress of Computer Science - Selected Papers.: Editorial de la Universidad de La Plata. 2014. p173 - 183. ISBN 978-987-1985-49-4

Así mismo, se realizaron publicaciones en los siguientes eventos científicos:

- **"Multi-Platform Mobile Application Development Analysis"** [10]
  Lisandro Delía, Nicolás Galdamez, Pablo Thomas, Leonardo Corbalán, Patricia Pesado
  IEEE Ninth International Conference on Research Challenges in Information Science - IEEE RCIS
  Atenas, Grecia. 13 al 15 de mayo de 2015.
  Forma de Publicación: Proceedings - CD ROM
  ISBN: 978-1-4673-6630-4

Organización del trabajo

Este trabajo se organiza de la siguiente manera:

- El capítulo 1 reseña la evolución de los dispositivos móviles.

- En el capítulo 2 se describen conceptos esenciales del desarrollo de aplicaciones para dispositivos móviles.
- En el capítulo 3 se profundizan conceptos relacionados al desarrollo de aplicaciones móviles multiplataforma, y se estudian diferentes metodologías y herramientas.
- El capítulo 4 contiene un conjunto de experimentos sobre desarrollos de aplicaciones móviles multiplataforma y el análisis de los resultados obtenidos.
- El capítulo 5 presenta las conclusiones obtenidas y el capítulo 6 los trabajos futuros propuestos en relación al tema.
- Finalmente, se detalla la bibliografía utilizada en el presente trabajo.

Capítulo 1. Tecnología Móvil

Hubo una época en el que los teléfonos móviles no tenían pantalla táctil, ni permitían grabar vídeo, ni siquiera podían conectarse a Internet. Tampoco podían enviar y recibir mensajes. Simplemente permitían la función básica de hablar por teléfono.

Dos décadas atrás, los dispositivos móviles eran aparatos de gran tamaño y pesados. Sólo algunas pocas personas podían acceder a estos dispositivos, ya que se consideraban artículos de lujo. Pero este escenario fue cambiando paulatinamente; llevando a que el celular se transforme en un elemento imprescindible en nuestras vidas. En Argentina, en el año 2003 había 4 millones de líneas activas. En la actualidad la cifra llega a 37 millones de líneas activas y 62 millones de líneas declaradas [11].

1.1 Los inicios

La Segunda Guerra Mundial originó la necesidad de comunicarse a distancia, por lo que Motorola creó un equipo militar llamado Handie Talkie H12-16 para comunicaciones vía ondas de radio.

Su salto a los sistemas civiles sucedió a finales de la década de los 40 con sistemas de radio analógicos en frecuencias FM principalmente y con servicios en las bandas HF y VHF ofrecidos por la americana Bell.

En 1955, Ericsson comercializó el Mobile Telephone System A (MTA) Phone (Figura 1), un teléfono que pesaba 40 kilogramos y que se instalaba en automóviles. Como curiosidad, tuvo un total de 125 usuarios hasta 1967.

Figura 1: Mobile Telephone System A (MTA)

La primera llamada móvil data del 3 de abril de 1973, cuando Martin Cooper (Figura 2), en aquel entonces ingeniero electrónico de Motorola, realizó una llamada desde la calle en Nueva York. La llamada fue realizada mediante un prototipo del Motorola DynaTAC 8000x. Considerado el primer móvil del mundo, pesaba cerca de un kilo y medía 33x4,5x8,9 centímetros, su batería tenía únicamente la autonomía de una hora en conversación, necesitando diez horas para cargarse. Su costo era de USD 4.000.

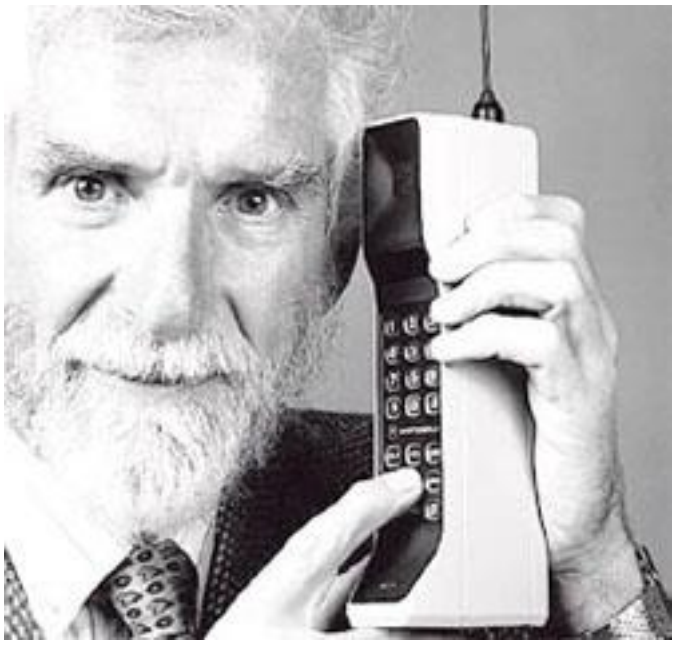


Figura 2: Martin Cooper, el autor de la primer llamada móvil

El verdadero conocimiento popular del teléfono móvil no llegó hasta la década de los 90, con la segunda generación de estos dispositivos.

La comunicación digital permitió mejorar la calidad de la escucha de voz, la reducción del tamaño y peso de los dispositivos utilizados y, sobre todo, el incremento de compañías operadoras que dieron paso a un mercado más competitivo y accesible.

Los usuarios fueron testigos de avances sorprendentes como la identificación de llamada o los mensajes de texto, que constituyeron a una nueva forma de comunicación.

Vertiginosamente, el celular pasó de ser considerado una herramienta de trabajo a generalizarse en todos los estratos de la sociedad.

Según avanzaba la década del 90, los teléfonos móviles iban adaptándose a esas novedades, incorporando pantallas cada vez de mayor tamaño y, en los más avanzados, haciendo desaparecer la antena que era necesario desplegar para poder realizar una comunicación en condiciones adecuadas.

Poco a poco, los dispositivos empezaron a integrar cada vez más funciones: aviso por vibración, juegos, bluetooth, infrarrojos, politonos y, ya con el cambio de siglo, pantallas a color y cámaras fotográficas. Más adelante llegaría la grabación de vídeo, el acceso a Internet, las pantallas táctiles, entre otras [12].

1.2 Evolución de los protocolos para redes de comunicaciones

Hubiera sido imposible el auge de los teléfonos móviles sin la estandarización, mejora y evolución de los protocolos para redes de comunicaciones y su soporte por las operadoras. Así, tras las primeras comunicaciones vía ondas de radio con banda de frecuencias por debajo de los 600 kHz, las posteriores en AM y FM, los servicios de Bell y Ericsson en los años 50 y 60, llegó esa primera llamada de 1973 que popularizó todo el sector.

La primera generación 1G fue responsabilidad de Ericsson con el sistema Telefonía Móvil Nórdica (NMT) y seguía utilizando canales analógicos. En 1986, la compañía modernizó el sistema funcionando a frecuencias superiores de 900 MHz posibilitando servicio para un mayor número de usuarios. Además del NMT, en los 80 se desarrollaron otros sistemas de telefonía móvil, como el Sistema Telefónico Móvil Avanzado (AMP) desarrollado por los laboratorios Bell.

La segunda generación 2G llegó en la década de los 90 con sistemas como Global System for Mobile Communications (GSM), Estándar Interno 136 (IS-136), Red Mejorada Digital Integrada (iDEN) y el Estándar Interno 95 (IS-95). GSM fue el desarrollo más relevante ya que fue el estándar europeo de telefonía móvil digital. En el proyecto participaron 26 compañías europeas de telecomunicaciones y en 1992 se pusieron en marcha las primeras redes europeas de GSM-900 y los primeros teléfonos móviles GSM. Además de Europa, GSM ha terminado imponiéndose también en Asia, América Latina, Oceanía y una parte de América del Norte.

La necesidad de mayores velocidades de transmisión de datos y mayores capacidades que permitieran nuevos servicios dieron paso a la Tercera Generación (3G), no sin antes pasar por el 2.5G que proporcionó el General Packet Radio Service (GPRS). El estándar europeo es el Universal Mobile Telecommunications System (UMTS) basado en la tecnología W-CDMA y está gestionado por la organización 3GPP, también responsable de GSM, GPRS y Enhanced Data Rates for GSM Evolution (EDGE).

La Cuarta Generación (4G) sucede a las tecnologías 2G y 3G y ofrece, entre otras mejoras, mayor seguridad y Calidad de Servicio (QoS), junto a velocidades de acceso muy superiores a las anteriores (mayores a 100 Mbit/s en movimiento y a 1 Gbit/s en

reposo). Está basada completamente en el protocolo IP, siendo un sistema de sistemas y una red de redes, que se alcanza gracias a la convergencia entre las redes de cables e inalámbricas. Según Vodafone "el 4G tiene finalmente una razón y es el entretenimiento" [13]. La norma Long Term Evolution (LTE) es la más extendida aunque no la única existente.

5G será la nueva revisión del sistema de conexión de red sin cables y se espera que esté disponible en el año 2020 (Figura 3) en países como Corea del Sur, como así también en Europa. Como 4G, la mayor velocidad de transferencia será su mayor avance con velocidades de descarga de hasta 10 Gbps (1,25 GB/s) [14].

Figura 3: Evolución de los protocolos para redes de comunicación móviles

1.3 La actualidad

Muchas de las actividades que un tiempo atrás solían hacerse desde una computadora, tales como revisar el correo electrónico, escuchar música, agendar una nueva cita en el calendario, compartir fotos en alguna red social, entre muchísimas otras, en nuestros días se llevan a cabo de forma natural desde los dispositivos móviles. La Figura 4 muestra cómo los dispositivos móviles les han ganado terreno a los equipos

tradicionales denominados ‘de escritorio’.

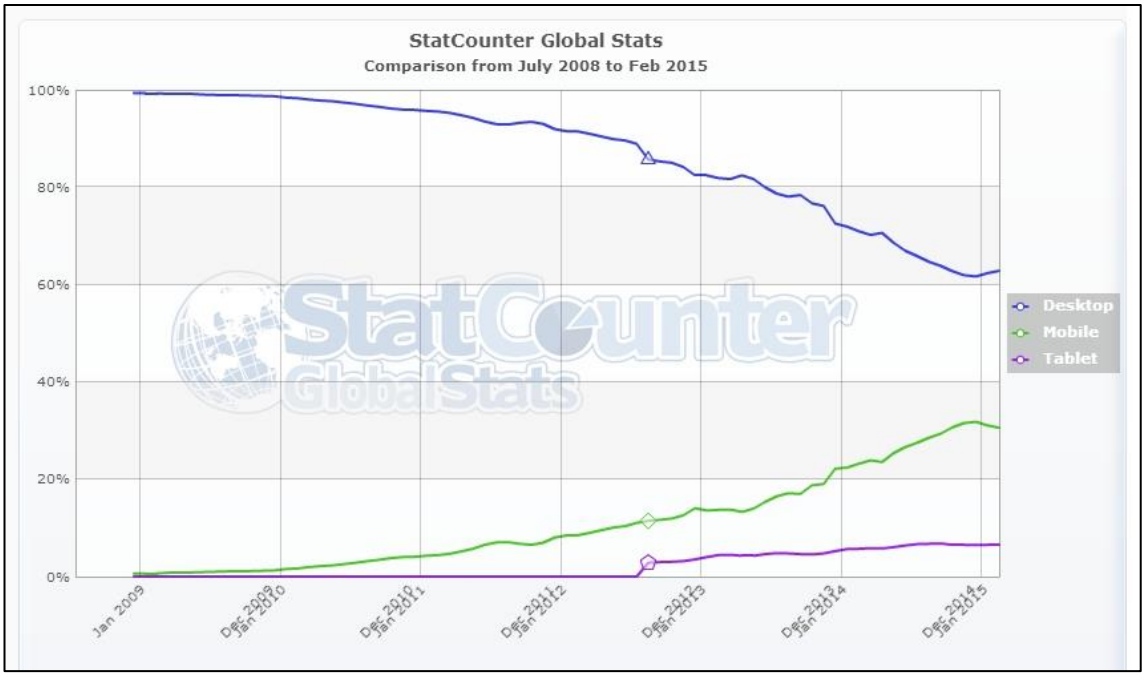


Figura 4: Comparación entre plataformas [15]

En la actualidad, existen cientos de modelos diferentes de dispositivos móviles y los fabricantes siguen innovando, intentando destacarse sobre la competencia. Hoy en día no nos sorprende encontrar dispositivos móviles con procesadores quad-core de 2.2 Ghz. Incluso, ya se comienzan a conocer prototipos de dispositivos con ocho o diez núcleos.

De la mano de los procesadores, la memoria RAM de los dispositivos móviles también ha crecido considerablemente, llegando a comercializarse dispositivos con 3 GB de memoria.

La cámara de los dispositivos es otra de las capacidades que más ha progresado, llegando a encontrarse sensores de 20 MP, y la posibilidad de grabar videos en formato UHD 4K (3840 x 2160).

El tamaño de las pantallas y sus resoluciones fueron otra de las características que más han evolucionado. Es posible encontrar dispositivos con pantallas de 5 pulgadas, con posibilidad de visualizar 16 millones de colores y con resoluciones Quad HD (2560 x 1440).

La tecnología de las baterías es un aspecto en que no se ha logrado grandes avances en el sector, ni se pronostican grandes progresos a corto plazo. Incluso, la llegada de chips

más potentes, mayores pantallas o el 4G disminuirán aún más la autonomía de las baterías.

El último aspecto a mencionar es el de las capacidades del dispositivo. Además de los ya clásicos sensores Global Positioning System (GPS), la capacidad de conectividad vía bluetooth, infrarrojo y WIFI, hoy surge una diversidad de opciones que pueden disponer los teléfonos inteligentes.

Por el lado de la conectividad, cada vez es más común encontrar terminales con mecanismos de comunicación Near Field Communication (NFC) o WIFI Direct.

Con respecto a los sensores, actualmente, existen dispositivos que cuentan con acelerómetros, giroscopios y magnetómetros, los cuales brindan a los desarrolladores de aplicaciones la posibilidad de controlar la posición del aparato, la aceleración que sufre el dispositivo con respecto a la fuerza de la gravedad, etc. También es común el uso de sensores de proximidad, por ejemplo, para apagar la pantalla mientras se atiende una llamada. El sensor de luminosidad permite medir el brillo de la luz ambiental. Así, el software del teléfono utiliza estos datos para ajustar el brillo de la pantalla automáticamente, haciéndola más clara o más oscura.

Los últimos smartphones de gama alta también son capaces de medir la presión atmosférica a través de un barómetro. Los datos que mide el dispositivo sirven para determinar en qué nivel por encima del mar se encuentra el teléfono, lo que se traduce en la mejora de la precisión del GPS.

Incluso, ya existen dispositivos que cuentan con un sensor de humedad del aire, el cual permite controlar los niveles de humedad para, por ejemplo, ayudar a las personas con problemas respiratorios.

Como si todo esto fuese poco, existen modelos con podómetro -para medir con precisión los pasos que da el usuario-, pulsómetro -para medir las pulsaciones a través de los vasos sanguíneos de los dedos- y hasta sensor de huellas digitales para cuestiones relativas a la seguridad.

Por lo anteriormente dicho, los desarrolladores de aplicaciones para dispositivos móviles de la actualidad, tienen el desafío de poder manipular desde sus aplicaciones todas las posibilidades descriptas.

Capítulo 2. Desarrollo de Aplicaciones Móviles Nativas

Si bien el desarrollo de aplicaciones para dispositivos móviles se remonta por lo menos a 10 años atrás, ha habido un crecimiento exponencial desde que se abrió la tienda de aplicaciones de iPhone en julio de 2008. Desde entonces, los fabricantes de dispositivos han creado tiendas de aplicaciones para otros dispositivos móviles, incluyendo Android, BlackBerry, Nokia Ovi, Windows Phone, entre otros [16].

Actualmente, el desarrollo de aplicaciones para dispositivos móviles es un campo en evolución con gran interés económico y científico. Una prueba de esto, es que para Julio del 2014, se disponían de más de 3 millones de aplicaciones entre las principales tiendas virtuales de aplicaciones. Y solamente en el año 2013, se contabilizaron 101 billones de descargas de aplicaciones móviles en todo el mundo, de las cuales 9 billones fueron descargas pagas y 92 billones descargas gratuitas [17].

Con el actual creciente número de plataformas móviles, desarrollar aplicaciones móviles se hizo muy difícil para las empresas, ya que necesitan desarrollar las mismas aplicaciones para cada plataforma de destino.

El desarrollo de aplicaciones nativas es la forma natural de implementar aplicaciones móviles. Las aplicaciones nativas son concebidas para ejecutarse en una plataforma específica, es decir, se debe considerar el tipo de dispositivo, el sistema operativo a utilizar y su versión.

Las aplicaciones nativas se desarrollan utilizando un entorno de desarrollo integrado (IDE) que proporciona las herramientas de desarrollo necesarias para la construcción y depuración de aplicaciones. El código fuente se compila para obtener código ejecutable, proceso similar que el utilizado para las tradicionales aplicaciones de escritorio.

Cuando la aplicación está lista para ser distribuida, debe ser transferida a las App stores (tiendas de aplicaciones) específicas de cada sistema operativo. Estas tiendas tienen un proceso de auditoría para evaluar si la aplicación se adecúa a los requerimientos de la

plataforma a operar. Cumplido este paso, la aplicación se pone a disposición de los usuarios.

La principal ventaja de este tipo de aplicaciones es la posibilidad de interactuar con todas las capacidades del dispositivo (cámara, GPS, acelerómetro, agenda, entre otras). Además no es estrictamente necesario poseer acceso a internet. Su ejecución es rápida, puede ejecutarse en modo background y notificar al usuario cuando ocurra un evento que necesite su atención.

Claramente estas ventajas tienen como contrapartida un mayor costo de desarrollo, pues se debe utilizar un lenguaje de programación diferente según la plataforma. Por ende, si se desea cubrir varias plataformas, se deberá generar una aplicación para cada una de ellas. Esto conlleva a mayores costos de desarrollo, actualización y distribución de nuevas versiones [9] [18] [19].

En la actualidad, hay una gran cantidad de sistemas operativos para dispositivos móviles. La Figura 5 muestra la utilización de los sistemas operativos en los últimos años en todo el mundo, mientras que la Figura 6 muestra la misma comparativa en Argentina. En ambos casos, el ranking es liderado por Android, seguido por iOS y en tercer lugar Windows Phone [20].

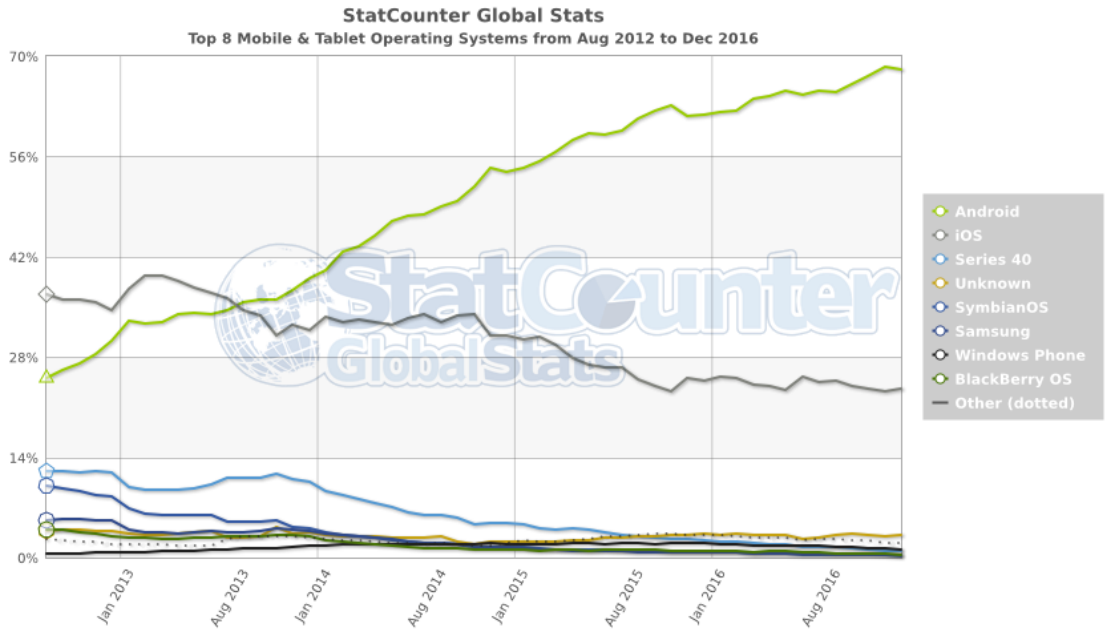


Figura 5- Utilización de Sistemas Operativos Móviles en el mundo

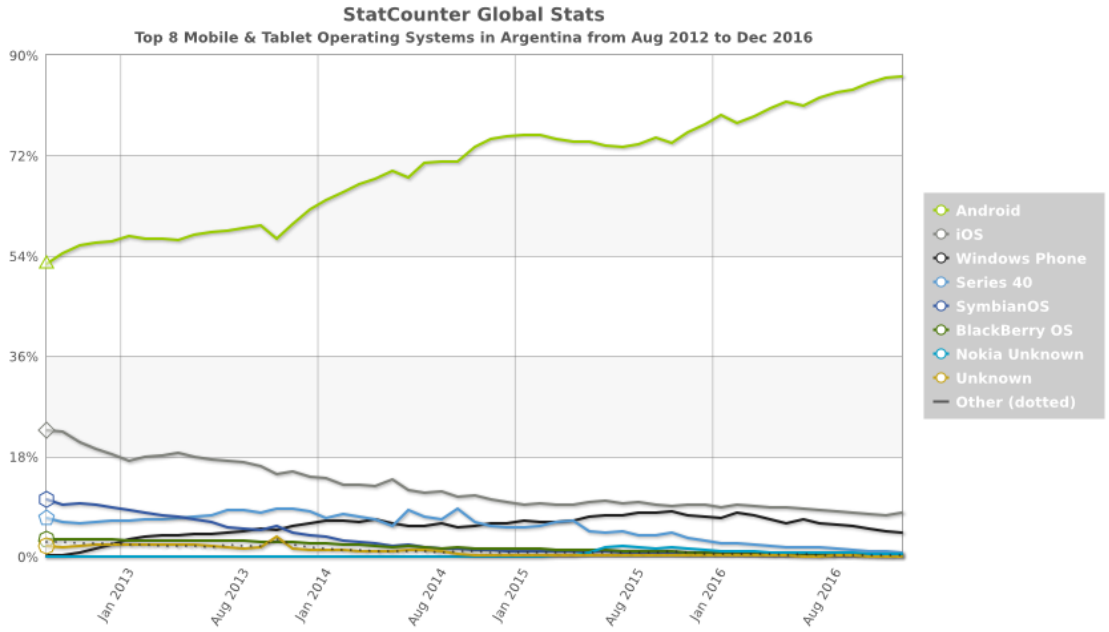


Figura 6- Utilización de Sistemas Operativos Móviles en Argentina

A continuación se analizará cómo es el proceso de desarrollo de aplicaciones móviles nativas en los sistemas operativos más relevantes de la actualidad; y luego, en la Tabla 1 se compararán aspectos técnicos de estas plataformas.

2.1 Desarrollo de Aplicaciones Nativas en Android

Android es el sistema operativo más utilizado en la actualidad. Puede ser utilizado en teléfonos inteligentes, tablets, relojes inteligentes, televisores y automóviles.

Está basado en Linux y es respaldado por Google.

2.1.1 Evolución

Inicialmente, el desarrollo de Android era llevado a cabo por la empresa Android Inc., la cual fue fundada en el año 2003 y recibía respaldo económico por parte de Google. Años más tarde, más precisamente en Julio del 2005, Google adquiere Android y da lugar a la primer versión del sistema operativo en el año 2007: **Android 1.0 Apple Pie**[1], desarrollado sobre el kernel de Linux 2.6. En su presentación se hizo hincapié en que era un sistema operativo para dispositivos móviles totalmente gratuito y open source, a

[1] Desde la salida de la primera versión del sistema operativo hasta el presente, Google ha seguido la tradición de ponerle un nombre de postre a cada una de sus versiones, siguiendo un orden alfabético.

diferencia de iOS.

La primera versión comercial tenía mucho margen de mejora y apenas inquietó a la competencia, pero ya introducía algunos conceptos que años después son un estándar de los sistemas operativos móviles:

- Menú desplegable de notificaciones
- Widgets de escritorio
- Android Market, la tienda de apps (no contaba con ningún sistema de pago para usuarios. Todo el catálogo era gratuito)
- Integración con Google Mail, Contacts y Calendar
- Navegador, Maps, Google Talk, reproductor de YouTube y soporte para cámaras.

En febrero del 2009, apenas 3 meses después del lanzamiento de Android 1.0 Apple Pie llega la primera actualización, **Android 1.1 Banana Bread**. No añadía grandes novedades, pero sí corregía numerosos fallos detectados en la primera versión e introducía un concepto por entonces poco usado por los rivales que buscaba facilitar la vida al usuario: las actualizaciones automáticas, con las que resultaba muy simple mantener todo el software al día.

El 30 de abril del 2009 lanzan una nueva actualización de gran importancia, con muchas novedades en cuanto a usabilidad: **Android 1.5 Cupcake.** Esta nueva versión introdujo las siguientes características:

- Teclado táctil QWERTY en pantalla con predicción de texto, y diccionario de usuarios para palabras personalizadas.
- Grabación y reproducción en formatos MPEG-4 y 3GP
- Widget de escritorio de Google para realizar búsquedas directamente
- SDK para el desarrollo de widgets de escritorio por parte de terceros
- Funciones del portapapeles ampliadas
- Interfaz para grabar y reproducir vídeos mejorada
- Posibilidad de auto-rotación.

Android 1.6 Donut aparecía en septiembre del 2009 con algunas novedades adicionales:

- Soporte para CDMA/EVDO, 802.1x, VPN, que ampliaba los mercados al alcance de Android
- Compatibilidad con distintas resoluciones de pantalla. Soporte WVGA
- Actualización y nuevo diseño del Android Market
- Utilidad de búsqueda universal en Internet y en el mismo dispositivo
- Galería, cámara y videocámara con mejor integración, con rápido acceso a la cámara
- Motor multi-lenguaje de Síntesis de habla para permitir a cualquier aplicación de Android "hablar" una cadena de texto
- Mejoras en las búsquedas por texto y voz

Apenas dos meses después, en noviembre del 2009, lanzan **Android 2.0 Eclair**, uno de los cambios más sustanciales sufridos por Android tanto a nivel de diseño como de arquitectura interna. Era una versión dirigida a dispositivos de mayor tamaño en un tiempo en el que los fabricantes empezaban a diversificar su oferta. La versión 2.1 mantuvo la misma nomenclatura y sólo corrigió algunos fallos, pero su uso fue mayor entre los fabricantes que la versión anterior. A continuación se mencionan algunas de las nuevas características:

- Sincronización de cuenta expandida, permitiendo a los usuarios agregar múltiples cuentas al dispositivo para sincronización de correo y contactos
- Google Maps Navigation, sistema de navegación GPS gratuito
- Compatibilidad con Microsoft Exchange
- Optimización en velocidad de hardware y GUI renovada.
- Soporte para más tamaños de pantalla y resoluciones, con una mejor tasa de contraste.
- Navegador actualizado, soporte para HTML5 y barra de dirección y búsqueda unificada
- Función Speech to Text para escribir textos mediante el uso de la voz
- Nueva pantalla de desbloqueo.

- Nuevas características para la cámara, incluyendo soporte de flash, zoom digital, modo escena, balance de blancos, efecto de colores y enfoque macro.
- Adición de fondos de pantalla animados, permitiendo la animación de imágenes de fondo de la pantalla inicio para mostrar movimiento.

El 20 de mayo del 2010 aparecía una nueva actualización del sistema operativo, **Android 2.2 Froyo**. Traía numerosos cambios, algunos copiados de otras ROMs y otros con el uso empresarial en mente:

- Pantalla Home totalmente rediseñada con 5 paneles en lugar de 3
- Nueva galería de imágenes
- Soporte para actuar como hotspot[2] para otros dispositivos, tethering[3] de datos
- Soporte para Flash 10.1
- Función copiar y pegar mejorada en Google Mail
- Nueva pantalla alternativa de desbloqueo mediante código PIN
- Grabación de vídeo en 720p
- Optimizaciones en velocidad, memoria y rendimiento. Compilador JIT
- Integración del motor de JavaScript V8 de Chrome en el navegador.
- Soporte para el servicio Android Cloud to Device Messaging (C2DM), habilitando notificaciones push

El año 2010 iba a concluir con el lanzamiento de una versión sumamente popular que prevaleció gran tiempo: **Android 2.3 Gingerbread**. Esta versión contó con las siguientes características:

- Diseño de la interfaz de usuario actualizado, con incremento en velocidad y simpleza
- Nuevo diseño para el teclado numérico en pantalla
- Soporte para tamaños y resoluciones de pantalla extra-grandes (WXGA y mayores)
- Función para copiar y pegar mejorada con soporte para caracteres individuales en lugar de cajas de texto

[2] lugar que ofrece acceso a Internet a través de una red inalámbrica y un enrutador conectado a un proveedor de servicios de Internet
[3] proceso por el cual un dispositivo móvil con conexión a Internet actúa como pasarela para ofrecer acceso a la red a otros dispositivos

- Soporte para NFC
- Herramientas de visualización de consumo y uso de la batería mejoradas
- Soporte para cámaras frontales
- Acceso de bajo nivel para los desarrolladores de juegos
- Recolector basura concurrente para incrementar el rendimiento.
- Sustitución del sistema de archivos YAFFS por ext4
- Soporte nativo para más sensores (tales como giroscopio y barómetro).
- Soporte de chat de video o voz, usando Google Talk.

En febrero del 2011 Google lanza **Android 3.0 Honeycomb**. Se trata de una actualización específica para tablets, no compatible con teléfonos, que introducía las líneas maestras de la interfaz en el futuro. Las versiones 3.1 y 3.2 mantuvieron el mismo nombre y fueron básicamente un conjunto de correcciones. A continuación se listan los cambios significativos que tuvieron lugar en estas versiones:

- Pantalla de Inicio rediseñada
- Inclusión de tonos azules en la interfaz en detrimento del verde tradicional
- Nuevas funcionalidades para el emplazamiento y uso de widgets
- Fin de los botones físicos. Adaptación automática del SO según el dispositivo
- Agregado de barra de sistema, con características de acceso rápido a notificaciones, estados y botones de navegación suavizados, disponible en la parte inferior de la pantalla.
- Añadida la barra de acción (Action Bar en inglés), entregando acceso a opciones contextuales, navegación, widgets u otros tipos de contenido en la parte superior de la pantalla.
- Multitarea simplificada – tocando Aplicaciones recientes en la barra del sistema permite a los usuarios ver instantáneas de las tareas en curso y saltar rápidamente de una aplicación a otra.
- Aceleración gráfica mediante hardware
- Optimización del renderizado de gráficos 3D
- Soporte para periféricos USB

Octubre del 2011 fue el mes del lanzamiento de **Android 4.0 Ice Cream Sandwich**, basado en el núcleo de Linux 3.0.1. Esta nueva versión introdujo las siguientes mejoras:

- Nueva Interfaz Holo y fuente tipográfica Roboto
- Sistema de gestión de notificaciones mejorado
- Multitarea mejorada
- Android Beam, funcionalidad para transferir datos entre dos dispositivos vía NFC
- Función de desbloqueo mediante el rostro
- Nuevas funciones para la visualización y gestión del consumo de datos
- Nuevas aplicaciones de correo y calendario
- Habilidad de acceder a aplicaciones directamente desde la pantalla de bloqueo.
- Herramienta integrada de captura de pantalla
- Soporte MKV
- Aceleración por hardware de la interfaz de usuario
- Soporte Stylus (lápiz táctil)

En Julio del 2012 se presenta **Android 4.1 Jelly Bean**, que incluye las siguientes novedades:

- Sistema de detección de entrada de datos táctiles optimizado
- Estreno de Google Now, el servicio-asistente de voz inteligente de Google
- Navegador Google Chrome
- Búsqueda mediante voz mejorada
- Nuevas posibilidades para las notificaciones interactivas de escritorio
- Dictado de voz offline
- Se deja de dar soporte al Flash Player

Meses después, en noviembre del 2012 se lanza la versión **Android 4.2 Jelly Bean**, la cual incluye habituales correcciones, rendimiento mejorado y cambios puntuales en ciertas aplicaciones.

Android 4.3 Jelly Bean es anunciado el 24 de julio del 2013. Uno de los objetivos de esta versión fue tratar de consolidar a Android como un sistema operativo capaz de ejecutar juegos. Esta nueva versión cuenta con las siguientes novedades:

- Soporte multiusuario y de perfiles mejorado
- Soporte OpenGL ES 3.0
- Bluetooth Smart
- Plataforma Google Games

- Servicios de localización Wi-Fi mejorados
- Ya no es necesario pulsar el icono del micrófono para realizar una búsqueda de voz. Solo hay que decir "OK Google" y en seguida ordenar al equipo lo que se necesite.

Android 4.4 KitKat es anunciado en octubre del 2013. Esta nueva versión intenta corregir el problema de fragmentación: se cuenta con muchas versiones y los fabricantes tienen dificultades para adaptar sus productos a los requerimientos de cada nueva versión. Se destaca de esta versión:

- Menores requisitos de hardware para corregir la fragmentación de versiones
- Compatible con terminales con 512 MB de memoria RAM
- Reducción del consumo de batería mediante la optimización de los sensores
- Inclusión de la suite ofimática QuickOffice
- Servicios de almacenamiento online integrados: Google Drive, Box...
- Soporte para infrarrojos. Usar el móvil como mando de TV
- Aplicaciones a pantalla completa
- Captura de pantalla en vídeo

En octubre del 2014 se lanza **Android 5 Lollipop**, dándole soporte a otros dispositivos como Android TV, relojes inteligentes, autos, etc. Esta nueva versión incluye:

- Nuevo diseño basado en Material Design que logra un flujo de trabajo más fluido
- Interfaz que se adapta a cualquier tamaño de dispositivo
- Renovado sistema de notificaciones inteligente
- Interesante vista multitarea que muestra capas con las diferentes aplicaciones abiertas
- Función Android Smart Lock, que permite emparejar un dispositivo Android con otro, ya sea un reloj inteligente o un automóvil.
- Modo "Invitado" para que puedas prestar el dispositivo sin que otros usuarios tengan acceso a la información privada.

- Soporte para sistemas de 64 bits

En octubre del 2015 se anuncia **Android 6 Marshmallow** que incluye las siguientes novedades:

- Soporte para autenticación vía huella digital.
- Desinstalación rápida de las aplicaciones desde la pantalla de inicio.
- Nuevo esquema de gestión de energía llamado Doze.
- Mayor duración de la batería cuando el dispositivo está en reposo.
- Inclusión de Android Pay que empleará el chip NFC.
- Mejoras en Google Now.
- Mayor control sobre los permisos requeridos por las aplicaciones.

2.1.2 Proceso de Desarrollo

Las aplicaciones nativas para esta plataforma se implementan utilizando el lenguaje de programación Java.

El estándar sugiere utilizar el IDE oficial Android Studio (Figura 7), aunque es posible utilizar otros entornos. Independientemente del IDE utilizado, se necesita disponer del Software Development Kit (SDK) de la versión de Android en la que se desea trabajar. El SDK provee todas las herramientas necesarias para desarrollar, compilar, depurar, y simular aplicaciones.

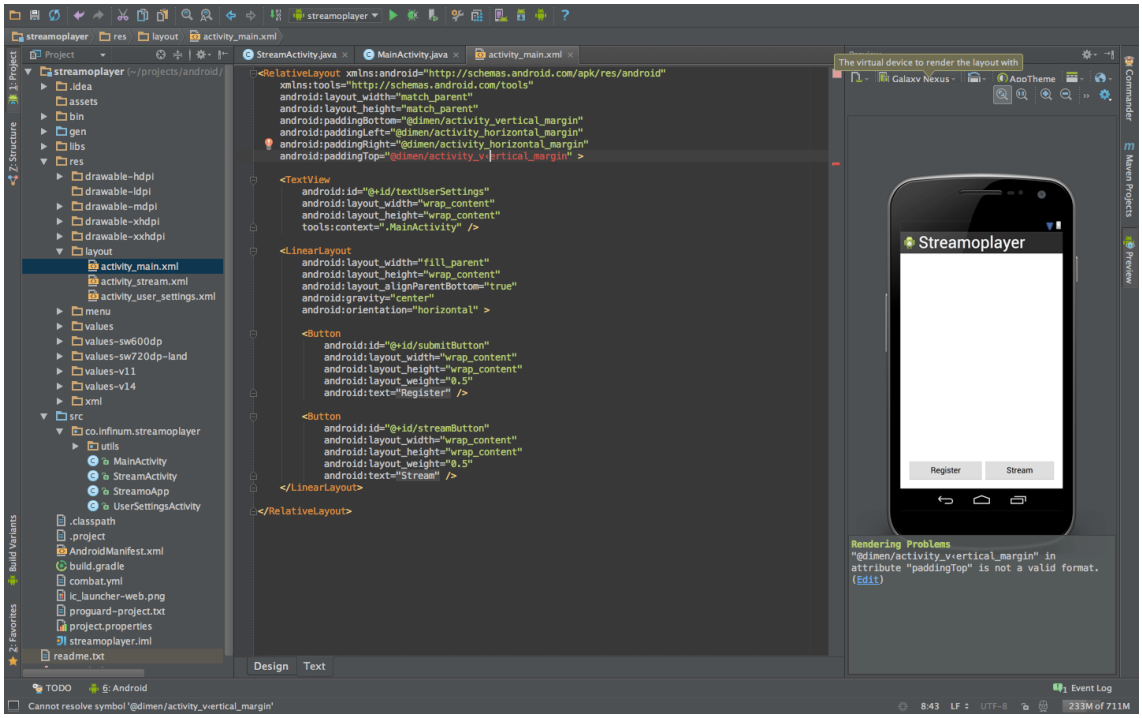
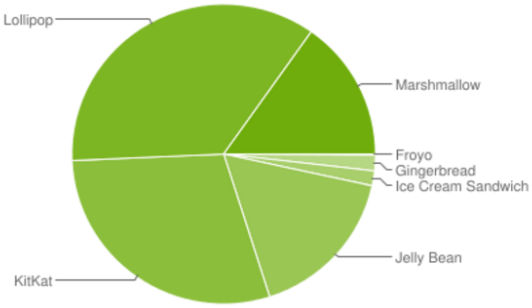


Figura 7: Entorno de desarrollo Android Studio

Uno de los interrogantes que surgen en el momento de crear una aplicación es con cual versión de la Application Programming Interface (API) de Android trabajar. Se debe tener en cuenta que al trabajar con una versión específica se está limitando cuales son los dispositivos que soportarán la aplicación a crear. No se recomienda utilizar una API antigua -ya que no se estaría utilizando muchos de los avances de la plataforma- ni tampoco la última en el mercado –debido a que pocos serían los usuarios que podrían utilizarla-. En la Figura 8 se visualiza la distribución de versiones según los dispositivos activos, a la fecha de Diciembre 2016 [21]. De esa información, se desprende que si se optara por trabajar con la API 16 (Jelly Bean), el 96,6% de los celulares activos podrán utilizar la aplicación a crear.

Version	Codename	API	Distribution
2.2	Froyo	8	0.1%
2.3.3 - 2.3.7	Gingerbread	10	1.7%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	1.6%
4.1.x	Jelly Bean	16	6.0%
4.2.x		17	8.3%
4.3		18	2.4%
4.4	KitKat	19	29.2%
5.0	Lollipop	21	14.1%
5.1		22	21.4%
6.0	Marshmallow	23	15.2%



Datos recopilados durante un período de 7 días hasta el 1 de agosto de 2016.
No se muestran versiones con una distribución inferior al 0,1%.

Figura 8: Distribución de versiones de Android - Diciembre 2016

2.2 Desarrollo de Aplicaciones Nativas en iOS

iOS es un sistema operativo para dispositivos móviles de la empresa multinacional Apple. Si bien fue originalmente desarrollado para el dispositivo iPhone, luego el sistema operativo alcanzó otros dispositivos, como iPod Touch iPad. A diferencias de otros sistemas operativos para dispositivos móviles, iOS solo funciona en dispositivos creados por la empresa Apple, pero no así en otro hardware de terceros.

2.2.1 Evolución

El sistema operativo móvil iOS se deriva del sistema operativo OS X, el cual está basado en UNIX.

La primera versión del sistema operativo se presentó en enero del 2007. A diferencia de sus sucesores, la primera versión no se llamó iOS, sino **iPhone OS**. Se trataba de una plataforma bastante cerrada, ya que las únicas aplicaciones disponibles eran las que estaban integradas al sistema operativo y fueron creadas por Apple, como los Mapas, Mail, Fotos, Calendario, entre otras.

El desarrollo de aplicaciones nativas por parte de terceros no tuvo lugar hasta marzo del 2008, momento en el cual Apple liberó el primer kit de desarrollo.

Meses después se lanza la segunda versión del sistema operativo: **iPhone OS 2**. En esta oportunidad, introducen la App Store, la cual permitió la descarga de aplicaciones de terceros. Además, la aplicación de Mapas recibió compatibilidad con el GPS y el Mail notificaciones push.

En 2009, Apple lanzó **iPhone OS 3**, y en su versión 3.2 se le cambió el nombre a **iOS 3.2**. iOS 3 incorporó grandes novedades: búsqueda Spotlight, botones Copiar y Pegar, grabación de vídeo, edición de vídeo, mensajes MMS, comandos de voz, modo landscape y otras muchas apps. Además, se siguió explotando las ventajas de las notificaciones push y se continuó evolucionando su SDK, dándole mayor poder a los desarrolladores de aplicaciones.

iOS 4 llegó a mediados del 2010, y trajo consigo nuevos cambios a la plataforma. Los más destacables fueron la inclusión de fondos de pantalla y la posibilidad de poder abrir varias aplicaciones en simultáneo. Adicionalmente, las aplicaciones podían ser agrupadas mediante carpetas, algo simple pero útil.

En Junio del 2011 se libera **iOS 5**. Esta nueva versión ofrecía grandes novedades: un Centro de Notificaciones, Siri –el asistente virtual que marcó tendencia en los últimos años-, iMessage, iCloud, Game Center, Quiosco, Recordatorios, integración social para Twitter, y se mejoró notablemente la app de navegación web nativa, Safari.

iOS 6 fue presentado en Junio del 2012 y trajo como novedades la inclusión de Apple Maps (en reemplazo de Google Maps), mejoras en Siri con información mucho más precisa, la integración de Facebook y Twitter en el sistema, el modo no molestar, el soporte para LTE, entre otras.

Los usuarios de este sistema operativo recibieron una de las mayores actualizaciones en Junio del 2013, cuando se presenta **iOS 7**. Esta nueva versión contó con un rediseño total de la interfaz de usuario, basándose en transparencias y degradados. Además, se introdujo el Centro de Control, AirDrop, una renovada app de fotos, iTunes Radio y CarPlay, mejoras en el App Switcher, videos a cámara lenta, entre otras cosas.

Junio del 2014 fue el mes donde quedó disponible **iOS 8**, una nueva versión del sistema operativo que intenta resolver algunos errores de iOS 7 y añadir ciertas mejoras. Las extensiones y la posibilidad de conectar apps entre sí mediante el menú compartir hicieron que iOS 8 fuese mucho más productivo. Además incorporó widgets para el

Centro de Notificaciones o las notificaciones interactivas. Además, se implementaron los contactos favoritos, los teclados de terceros, la función Reachability y la utilidad Continuity, y las aplicaciones Apple Pay, Salud, HandOff, QuickType, Compartir en Familia, iCloud Drive y Apple Music.

En el 2015 se presenta **iOS 9**, que introdujo todo un cambio en el uso del sistema gracias a la compatibilidad con el 3D Touch del iPhone 6s y el iPhone 6s Plus. Se actualizó Notas con nuevas funciones, se sustituyó Newsstand por News, se incorporaron mejoras en Passbook que pasó a llamarse Wallet y se mejoró Mapas con direcciones y tráfico. Se añadió Split View, Picture-in-Picture y atajos en teclados de terceros. Además, el sistema ahora podía aprovechar mejor la autonomía del dispositivo gracias al Modo ahorro de batería que limita las conexiones y características del sistema para no consumir tanta energía.

Por último, en Junio del 2016 quedó disponible **iOS 10**, versión que se caracteriza por abrirse a los desarrolladores. Con esta nueva versión se puede tener acceso a Siri, al 3D Touch y a apps directas del sistema operativo. Además, el centro de widgets ahora es más intuitivo y la pantalla de bloqueo más útil. Como es habitual, se aprecian mejoras en distintas aplicaciones del sistema operativo y se permite borrar apps de Apple que no sean utilizadas por los usuarios.

2.2.2 Proceso de desarrollo

Las aplicaciones móviles para la plataforma iOS se desarrollan utilizando el lenguaje de programación Objetive C. El entorno de desarrollo oficial es la plataforma Xcode [22], la cual opera en conjunto con Interface Builder, una herramienta gráfica para la creación de interfaces de usuario.

Actualmente, hay dos grandes versiones del sistema operativo móvil en curso, como se puede apreciar en la Figura 10. Es algo a destacar que la última versión del sistema operativo es soportada por el 63% de los dispositivos activos. Claramente, la fragmentación interna es mucho menor que la encontrada en Android.

Figura 9: Entorno de desarrollo Xcode

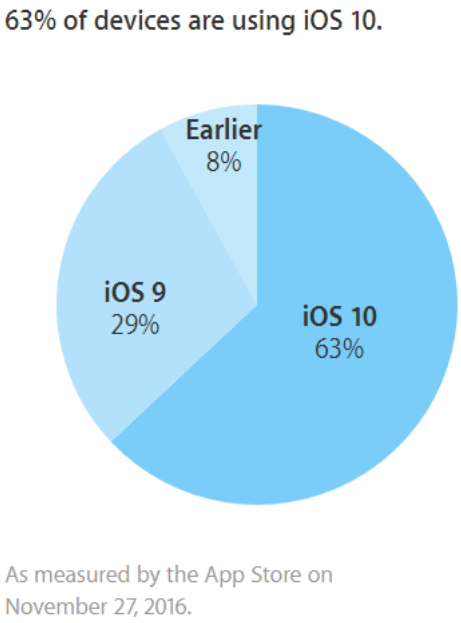


Figura 10: Distribución de versiones de iOS - Noviembre 2016

2.3 Desarrollo de Aplicaciones Nativas en Windows Phone

El sistema operativo móvil Windows Phone fue desarrollado por Microsoft, y sucede a Windows Mobile. A diferencia de su predecesor está enfocado en el mercado de consumo en lugar del mercado empresarial.

2.3.1 Evolución

Su primera versión, lanzada en septiembre del 2010, fue denominada **Windows Phone 7**. Fue presentado como un nuevo sistema operativo que incluye funciones de integración con los servicios Xbox Live y Zune. La interfaz, conocida como "Metro", ha sido revisada en su totalidad y comparte características visuales similares a la interfaz del dispositivo Zune HD.

Windows Phone 7.5 es una actualización para Windows Phone lanzada en septiembre de 2011. Esta nueva versión incorporó numerosas características, como por ejemplo la inclusión de Internet Explorer 9, con soporte para CSS3 Media Queries, y soporte para usar GPS cuando se trabaje con aplicaciones de ubicación geográfica, entre otros.

En enero del 2013 se presenta **Windows Phone 7.8**, una actualización que trae mejoras como la nueva interfaz de usuario y fondos personalizados para la pantalla de bloqueo. Fue la última gran actualización de Windows Phone 7, ya que Microsoft se centró en Windows Phone 8.

Windows Phone 8 es la siguiente versión del sistema operativo, la cual fue presentada a finales de 2012. Entre las nuevas características se incluyen:

- Nuevas pantallas de inicio y de bloqueo más personalizables
- Rincón infantil: un espacio exclusivo y controlado para los niños.
- Cartera: para almacenar tarjetas de fidelización y de crédito
- NFC
- Internet Explorer 10
- Integración con Skype
- Nuevo núcleo Windows NT, con soporte para procesadores de varios

núcleos

- Captura de pantalla

Su próxima versión es **Windows Phone 8.1** y fue presentada en abril del 2014. Entre sus características más relevantes, se encuentran:

- Centro de notificaciones
- Asistente de voz conocido con el nombre de Cortana
- Sensores (de Wi-Fi, de datos y de batería)
- Aplicaciones que vienen en el paquete de instalación, como: Salud y ejercicios, Comida y bebida, Viajes y Mapas (esta última compite con Here Maps, la cual es la aplicación de mapas de Nokia).
- Mejoras en la pantalla de inicio: posibilidad de agregar fondos de pantalla y una tercera columna de mosaicos personalizables (Tiles).

A mediados del 2014 Microsoft decide cambiar de estrategia, tratando de unificar todas las plataformas como PC, tablets, smartphones, Xbox One, entre otros. Windows 10 (móvil) se dio a conocer públicamente durante un evento el 21 de enero para 2015. Al igual que su homólogo, Windows 10 Mobile utiliza como navegador predeterminado Microsoft Edge, reemplazando Internet Explorer de Windows Phone 8.1, ya que este no se adecuaba a las nuevas líneas de diseño del sistema operativo.

2.3.2 Proceso de desarrollo

El entorno de desarrollo más popular para el desarrollo de aplicaciones para Windows Phone es Visual Studio (Figura 11). En la actualidad, es posible utilizar diversos lenguajes de programación: VB.NET y C#.NET, C++ y JavaScript.

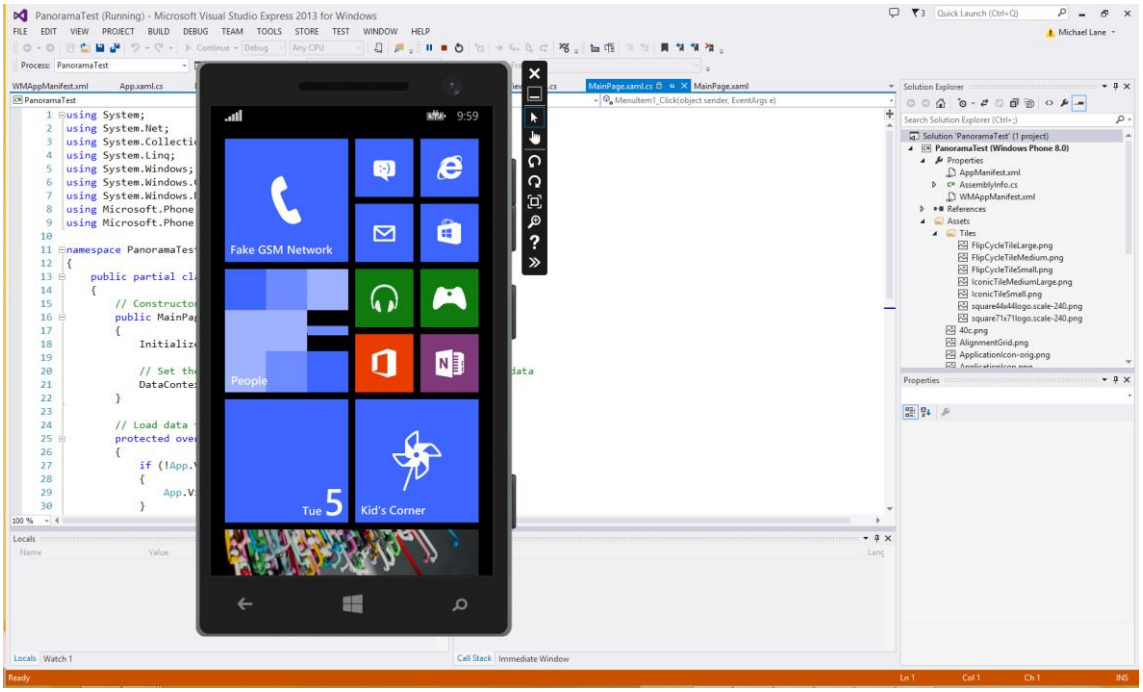


Figura 11: Entorno de desarrollo Visual Studio

2.4 Diferencias técnicas entre Android, iOS y Windows Phone

Desarrollar aplicaciones móviles para los tres sistemas operativos más populares implica conocer ciertos aspectos técnicos de cada plataforma. Algunos de estos aspectos técnicos difieren sustancialmente entre cada sistema operativo.

Una de las características a tener en cuenta es si las aplicaciones móviles se ejecutan dentro de una máquina virtual específicamente diseñada, o si se ejecutan directamente sobre el sistema operativo.

Otro aspecto que los desarrolladores deben tener en cuenta es el lenguaje de programación a utilizar para crear aplicaciones nativas. Este aspecto es importante porque incide en otros aspectos, como por ejemplo la gestión de memoria.

Además del lenguaje de programación a utilizar, es importante conocer la tecnología requerida para diseñar interfaces de usuario.

Asimismo, los desarrolladores deben familiarizarse con diferentes IDEs según la plataforma en la que se desea desarrollar, como así también con diferentes tiendas de aplicaciones donde alojar los productos finales.

Por último, la tecnología de hardware requerida para soportar el sistema operativo también difiere según cada plataforma.

La Tabla 1 compara cada uno de estos aspectos en los sistemas operativos móviles Android, iOS y Windows Phone.

Sistema Operativo	Máquina Virtual	Lenguaje de Programación	Interfaces de usuario	Gestión de memoria	IDE	Plataforma de desarrollo	Dispositivos	Tienda de Aplicaciones
Android	Dalvik VM	Java	XML files	Garbage collector	Android Studio	Multiplatforma	Heterogeneos	Google Play Store
iOS	No	Objective-C	Cocoa Touch	Reference counting	XCode	Mac OS X	Homogeneos	iTunes Apps Store
Windows Phone	CLR	C# and .Net	XAML files	Garbage collector	Visual Studio	Windows Vista/7	Homogeneos	Windows Phone Store

Tabla 1: Diferencias técnicas entre las plataformas Android, iOS y Windows Phone

Capítulo 3. Desarrollo de Aplicaciones Móviles Multiplataforma

En los últimos años el mercado de los dispositivos móviles, en especial smartphones, ha mostrado un crecimiento notable tanto en Argentina como en todo el mundo. En particular, en nuestro país, las plataformas que más han crecido son Android e iOS [15,20].

Actualmente gran parte de la industria del software se concentra en desarrollar soluciones para dispositivos móviles, proveyendo especialmente aplicaciones nativas.

Las aplicaciones nativas, tal como se mencionó en el capítulo anterior, ofrecen la posibilidad de acceder a todas las capacidades del dispositivo (cámara, GPS, acelerómetro y agenda, entre otras), su rendimiento es alto, el acceso a Internet no es estrictamente necesario y pueden ejecutarse en segundo plano notificando al usuario cuando se requiera su atención. Estas aplicaciones pueden distribuirse/comercializarse a través de las tiendas en línea correspondientes.

El principal reto para los proveedores de aplicaciones es proporcionar soluciones para todas las plataformas, pero esto conlleva un alto costo [6]: no es posible reusar el código fuente entre plataformas diferentes, multiplicando esfuerzos y elevando los costos de desarrollo, actualización y distribución de nuevas versiones.

El desarrollo multiplataforma procura optimizar la relación costo/beneficio compartiendo la misma codificación entre las versiones para las distintas plataformas. Entre otras ventajas sobresalen: menor tiempo y costo de desarrollo; prestaciones similares a las nativas con acceso al hardware del dispositivo, y disponibilidad de entornos potentes de desarrollo (Delphi, Visual Studio, etc.) o; en su lugar, utilización de tecnologías (HTML5, Javascript y CSS) bien conocidas por los desarrolladores web quienes pueden trasladar sus conocimientos y experiencias al paradigma móvil. Sin embargo, el rendimiento de las aplicaciones y sus interfaces de usuario, pueden afectar la experiencia de usuario.

Las aplicaciones multiplataforma pueden clasificarse en: aplicaciones web móviles, híbridas, interpretadas y generadas por compilación cruzada [5]. En las secciones siguientes se analizaran cada una de estas clasifcaciones.

3.1 Aplicaciones Web Móviles

Las Aplicaciones Web Móviles, diseñadas para ejecutarse dentro de un navegador, se desarrollan con tecnología web estándar (HTML, CSS y JavaScript), y cuentan con una serie de características favorables: no necesitan adecuarse a ningún entorno operativo, son independientes de la plataforma y su puesta en marcha es rápida y sencilla.

Por contrapartida, sus tiempos de respuesta decaen debido a la interacción cliente-servidor. Al mismo tiempo, resultan ser menos atractivas que las aplicaciones nativas ya que no se encuentran instaladas en el dispositivo, lo que implica acceder previamente a un navegador. Además, las restricciones de seguridad impuestas a la ejecución de código por medio de un navegador, limitan el acceso a todas las capacidades del dispositivo [23]. Asimismo, no es posible desarrollar aplicaciones web que corran en segundo plano ni tampoco aplicaciones offline, ya que es requisito disponer de una conexión a internet para su funcionamiento.

El proceso de desarrollo de una aplicación web móvil debe contemplar una serie de características inherentes al entorno de ejecución. Como ya ha sido mencionado, las aplicaciones web móviles pueden ser accesibles desde cualquier dispositivo móvil que cuente con un navegador y acceso a internet, pero esto no significa que la navegación en la aplicación resulte ser cómoda u óptima. Por ejemplo, una aplicación web con un menú compuesto de 20 ítems puede implicar una usabilidad correcta desde una PC. Por lo contrario, para un dispositivo móvil con una pantalla limitada, no resulta práctico que se le presente al usuario tantas opciones en el menú. Con este ejemplo, lo que se pretende mostrar es que para que una aplicación sea accesible desde diferentes dispositivos móviles, posiblemente se tengan que realizar, al menos, dos diseños de presentación diferentes: un diseño para PC y otro diseño para dispositivos móviles, con una estructura organizativa de la aplicación más simplificada, con respecto al diseño tradicional. Por ejemplo, algunas de las funcionalidades de la aplicación para PC, pueden ser eliminadas para la versión para dispositivos móviles o presentadas de forma diferente.

Para desarrollar diferentes presentaciones de una misma aplicación web,

actualmente existen dos estrategias distintas, ambas válidas según el contexto. En los siguientes apartados se analizarán cada una de ellas.

3.1.1 Aplicación Web dedicada y exclusiva para dispositivos móviles

Esta metodología ha sido el estándar hasta el año 2012. En ella, la dirección URL y el código HTML de la versión móvil son distintos de la versión de escritorio. Por lo general, las direcciones URL tienen el formato m.nombresitio.com.

Algunas de las ventajas de este método:

- En general, la carga es más rápida y la navegación más cómoda.
- Es posible adaptar de manera más fácil el contenido de las secciones.

Desventajas de este método:

- Mantenimiento más costoso. Se debe mantener dos versiones distintas del mismo sitio.
- Dificulta el posicionamiento del sitio en los buscadores.
- Estos desarrollos al final tienden a ser versiones muy reducidas de la web para PC, llevando al usuario a frustrarse y a preferir ver la versión original de la web.

3.1.2 Aplicación Web con Diseño Adaptable

Con esta metodología la aplicación web se adapta al dispositivo desde el cual se la está accediendo. El código HTML y la dirección URL de la aplicación son únicos. El Diseño Adaptable, o Responsive Design (Figura 12), tiene sus orígenes en el año 2008, cuando la W3C discutió y describió sus propósitos. Desde el año 2012 -momento en el cual Google recomendó fuertemente su implicancia [24]- viene en ascenso, y posiblemente llegue a convertirse en un standard en un corto plazo.

Mediante un Diseño Adaptable todos los elementos de la web se reajustan en ancho y altura adaptándose al tamaño de tu pantalla. Incluso es posible ocultar secciones cuando se accede desde un dispositivo móvil.

Figura 12: Diseño Adaptable

Las principales ventajas del desarrollo de aplicaciones web con Diseño Adaptable son:

- El mantenimiento es menos costoso ya que solo hay que mantener una única versión de la aplicación.
- Se simplifica el proceso de posicionar la aplicación en los buscadores.

Como contrapartida, el Diseño Adaptable presenta una serie de desventajas:

- Requiere de mayores niveles técnicos para su desarrollo.
- El contenido de las aplicaciones con Diseño Adaptable puede ser mayor que el contenido de las aplicaciones web específicas para dispositivos móviles, perjudicando el tiempo de carga cuando se utiliza un tipo de conexión no muy potente y generando un mayor consumo de datos.

En la actualidad, existe una gran cantidad de frameworks que simplifican el desarrollo de aplicaciones web con diseño adaptable, destacándose por su popularidad y extensa documentación los frameworks Bootstrap [25] y Foundation [26].

3.2 Aplicaciones Híbridas

Las aplicaciones híbridas utilizan tecnologías web (HTML, Javascript y CSS) pero no son ejecutadas por un navegador. En su lugar, se ejecutan en un contenedor web (webview), como parte de una aplicación nativa, la cual está instalada en el dispositivo

móvil. Desde una aplicación híbrida es posible acceder a las capacidades del dispositivo, a través de diversas API.

Las aplicaciones híbridas ofrecen grandes ventajas permitiendo la reutilización de código en las distintas plataformas, el acceso al hardware del dispositivo, y la distribución a través de las tiendas de aplicaciones. En contrapartida, se observan dos desventajas de las aplicaciones híbridas respecto del caso nativo: i) la experiencia de usuario se ve perjudicada al no utilizar componentes nativos en la interfaz, y ii) la ejecución se ve ralentizada por la carga asociada al contenedor web.

Afortunadamente, existe una diversidad de frameworks que permiten desarrollar aplicaciones híbridas. A continuación, se hará un breve resumen de los más populares.

3.2.1 PhoneGap

PhoneGap [27] es un framework gratuito y open source, creado en el año 2008 por la empresa Nitobi Software, que permite crear aplicaciones móviles usando tecnología web estándar: HTML, JavaScript y CSS. En octubre del 2011 Adobe adquiere la empresa Nitobi, y aporta el código de PhoneGap a Apache Software Foundation, para iniciar un nuevo proyecto por encima de PhoneGap llamado Apache Cordova [28]. PhoneGap pasó a ser una de las varias distribuciones de Apache Cordova (Figura 13), brindando servicios extras, como por ejemplo Adobe PhoneGap Build [29], un servicio de compilación en la nube proporcionado por Adobe Creative Cloud, PhoneGap Desktop y PhoneGap Developer.

PhoneGap permite compilar aplicaciones para las siguientes plataformas: Android, IOS, Amazon Fire OS, Windows Phone, Windows 8, Ubuntu, Tizen, Firefox OS, Blackberry 10.

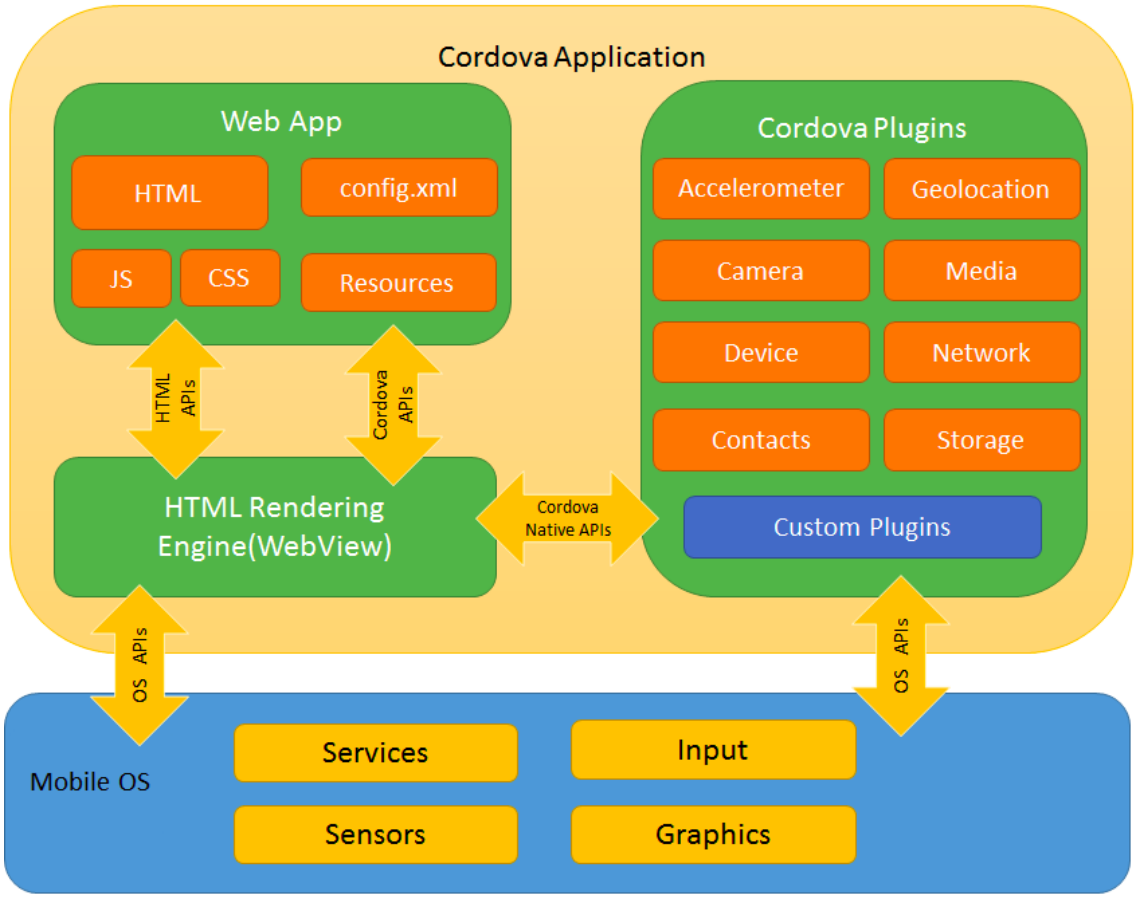


Figura 13: Funcionamiento de Apache Cordova

3.2.2 CocoonJS

CocoonJS [30] es un framework para el desarrollo de aplicaciones móviles, producido por Ludei [31]. Actualmente cuenta con una licencia propietaria y ofrece diferente tipos de planes pagos y gratuitos. El mencionado framework está basado en Apache Cordova, al igual que PhoneGap, pero se diferencia de este último en que hace un mayor hincapié en la performance de las aplicaciones resultantes. CocoonJS ofrece dos tipos de webview:

- Canvas+, un webview liviano optimizado para video juegos. Provee solo un subconjunto de las funcionalidades del HTML5 (canvas, audio, motion, geolocalización, etc) necesarias para desarrollar video juegos, con el objetivo de brindar la mejor performance posible.
- Webview+, un potente y completo webview basado en el webview del proyecto Chromium, el cual cuenta de una mejor performance en comparación con el webview por defecto de Apache Cordova.

CocoonJs permite crear aplicaciones tanto para iOS, como para Android.

3.2.3 Ionic

Ionic [32] es un completo SDK gratuito y open source para el desarrollo de aplicaciones híbridas, producido por Max Lynch, Ben Sperry, y Adam Bradley de la compañia Drifty Co en el año 2013. En un tiempo relativamente corto alcanzó una popularidad a destacar. Está basado en Apache Cordova, permitiendo desarrollar aplicaciones con tecnología web. A diferencia de otros frameworks para el desarrollo de aplicaciones hibridas, Ionic provee un framework para la interfaz de usuario y herramientas visuales para diseñar interfaces.

Mediante Ionic es posible crear aplicaciones para Android, IOS, Windows Phone y Firefox OS.

3.2.4 Sencha Touch

Sencha Touch [33] es un framework gratuito Model-View-Controller (MVC) JavaScript construido sobre el sistema de clases de Ext JS, diseñado especialmente para el desarrollo de aplicaciones web móviles para dispositivos táctiles [34].

Al igual que PhoneGap, Sencha Touch, permite a los desarrolladores crear aplicaciones móviles a partir de un desarrollo web con HTML5, CSS y Javascript. A diferencia de PhoneGap, Sencha no necesita de ninguna librería de terceros para construir interfaces similares a las nativas, ya que el framework provee consigo una gran cantidad de componentes visuales listos para utilizar.

Sencha empaqueta el código del proyecto mediante PhoneGap, para luego poder construir aplicaciones en todas las plataformas soportadas por PhoneGap.

3.3 Aplicaciones Interpretadas

Las aplicaciones interpretadas son implementadas utilizando un lenguaje base, el cual se traduce en su mayor parte a código nativo, mientras el resto es interpretado en tiempo de ejecución. Estas aplicaciones son implementadas de manera independiente de las plataformas utilizando diversas tecnologías y lenguajes, tales como Javascript, Java, Ruby y XML, entre otros.

Una de las principales ventajas de este tipo de aplicaciones es que se obtienen

interfaces de usuario totalmente nativas. Sin embargo, los desarrolladores experimentan una dependencia total con el entorno de desarrollo elegido.

Seguidamente, se presentaran algunos de frameworks más reconocidos para el desarrollo de aplicaciones interpretadas.

3.3.1 Appcelerator Titanium

Desarrollado por Appcelerator Inc., Appcelerator Titanium [35] es un framework de código abierto que permite la creación de aplicaciones móviles para las plataformas iOS y Android. Dicho framework está formado por Titanium Studio, un entorno de desarrollo de código libre para la codificación de aplicaciones móviles multiplataforma, y el SDK Titanium, una serie de herramientas para el desarrollo, testeo, análisis, depuración y compilación de aplicaciones.

Las aplicaciones desarrolladas con Appcelerator Titanium se codifican utilizando el lenguaje Javascript, el cual es evaluado en tiempo de ejecución mediante un intérprete de Javascript, que se ejecuta en el sistema operativo del dispositivo. Al iniciar la aplicación, desde el código nativo se ejecuta el código Javascript, y mediante la API de Titanium se mapea uno-a-uno cada elemento del código Javascript con los elementos nativos. De esta manera, la API de Titanium actúa como puente, logrando interfaces de usuario compuestas de controles nativos. Este entorno de desarrollo utiliza el framework Alloy diseñado para el desarrollo ágil de aplicaciones móviles. Alloy está basado en la arquitectura MVC y soporta el uso de tecnologías populares como Backbone.js [36] y Underscore.js [37].

Los creadores de Titanium destacan que mediante este framework es posible reutilizar entre el 60% y 90% del código entre distintas plataformas.

3.3.2 NativeScript

El 5 de marzo del 2015 la empresa Telerik lanzó NativeScript [38], un proyecto de código abierto, el cual permite generar aplicaciones nativas utilizando JavaScript. Asimismo, es posible desarrollar la aplicación utilizando el lenguaje TypeScript [39], lenguaje libre y de código abierto desarrollado por Microsoft, que extiende a JavaScript, esencialmente añadiendo tipado estático y objetos basados en clases. En este sentido, cuando se compila la aplicación se realiza la traducción del código TypeScript a código JavaScript.

NativeScript provee un módulo multiplataforma que permite obtener aplicaciones nativas desde código JavaScript. Este módulo expone las capacidades del dispositivo y de la plataforma subyacente, de una manera consistente y permite accederlas desde el código JavaScript. De igual manera, las interfaces de usuario pueden ser definidas mediante código JavaScript, documentos HTML y archivos CSS, abstrayéndose de los componentes nativos reales.

Cuando la aplicación es compilada, parte del código multiplataforma es traducido a código nativo, mientras que el resto es interpretado en tiempo de ejecución (Figura 14).

Por el momento, NativeScript permite generar aplicaciones para Android e iOS, y se prevé dar soporte a Windows Phone. NativeScript renderiza controles nativos, sin utilizar WebViews, logrando una performance y experiencia de usuario similar al de las aplicaciones nativas.

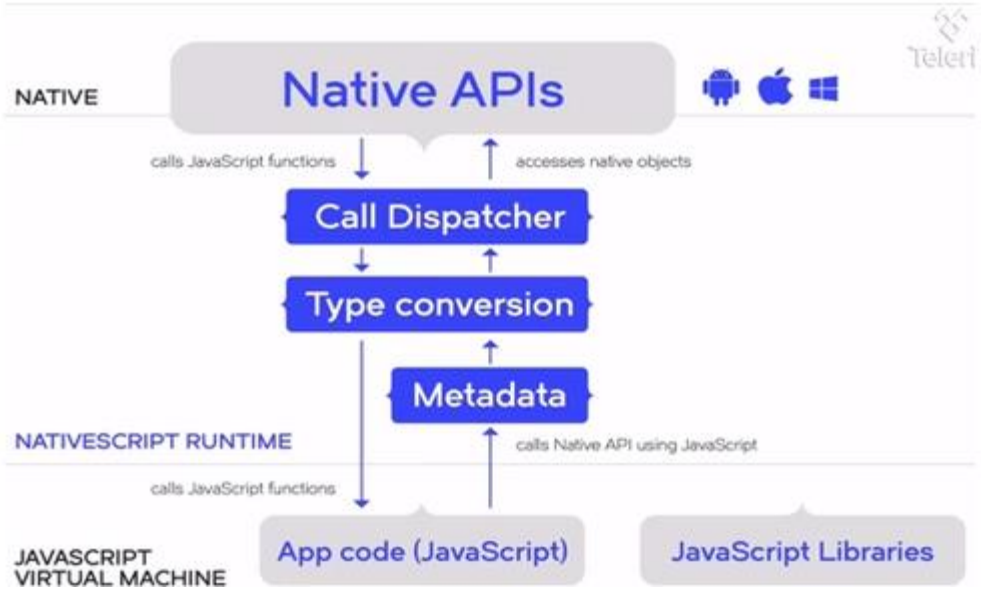


Figura 14: Proceso de interpretación mediante NativeScript

3.4 Aplicaciones Generadas por Compilación Cruzada

Estas aplicaciones se compilan de manera nativa creando una versión específica de alto rendimiento para cada plataforma destino. Ejemplos de entornos de desarrollo para generar aplicaciones por compilación cruzada son Applause [40], Xamarin [41], Embarcadero Delphi 10 Seattle [42] y RubyMotion [43].

El entorno de desarrollo abierto Applause utiliza como entrada un lenguaje

específico del dominio basado en el framework Xtext [44], diseñado explícitamente para aplicaciones móviles orientadas a datos, y genera código fuente en Objective C, Java, C# o Python. Las características de Xamarin, Embarcadero Delphi 10 Seattle y RubyMotion se presentan en secciones posteriores.

3.4.1 Xamarin

Xamarin es una plataforma de desarrollo de aplicaciones móviles para generar aplicaciones 100% nativas para iOS, Android y Windows, a partir de una base de código C#/.NET común para conseguir entre un 75 % y hasta casi un 100 % de reutilización de código entre plataformas. Las aplicaciones escritas con Xamarin y C# disponen de acceso completo a las API de la plataforma subyacente, así como de la capacidad de crear interfaces de usuario nativas y de realizar la compilación en código nativo, por lo que el impacto en el rendimiento en tiempo de ejecución es escaso.

Xamarin es una compañía que se estableció en mayo de 2011, por los mismos ingenieros que crearon el proyecto Mono [45], consistente en una implementación libre de la plataforma de desarrollo .NET para dispositivos Android, iOS y GNU/Linux. Anteriormente, este proyecto se llamaba MonoTouch y MonoDroid.

Si bien Xamarin cuenta con su propio IDE denominado Xamarin Studio (Figura 15), es posible integrarlo con Microsoft Visual Studio, y de esta manera generar también aplicaciones para Windows, incluido Windows RT para tablets y Windows Phone para celulares.

En febrero del 2016, Xamarin es comprada por Microsoft y desde ese momento la plataforma Xamarin y el IDE Xamarin Studio pasaron a ser gratuitas.

Xamarin propone un enfoque de desarrollo multiplataforma en el que se comparte la codificación completa de la lógica del negocio. Sin embargo, las interfaces deben ser programadas de manera independiente para cada una de las plataformas destino. Así, la reutilización de código, según estudios estadísticos de la compañía Xamarin, es cercana al 85%.

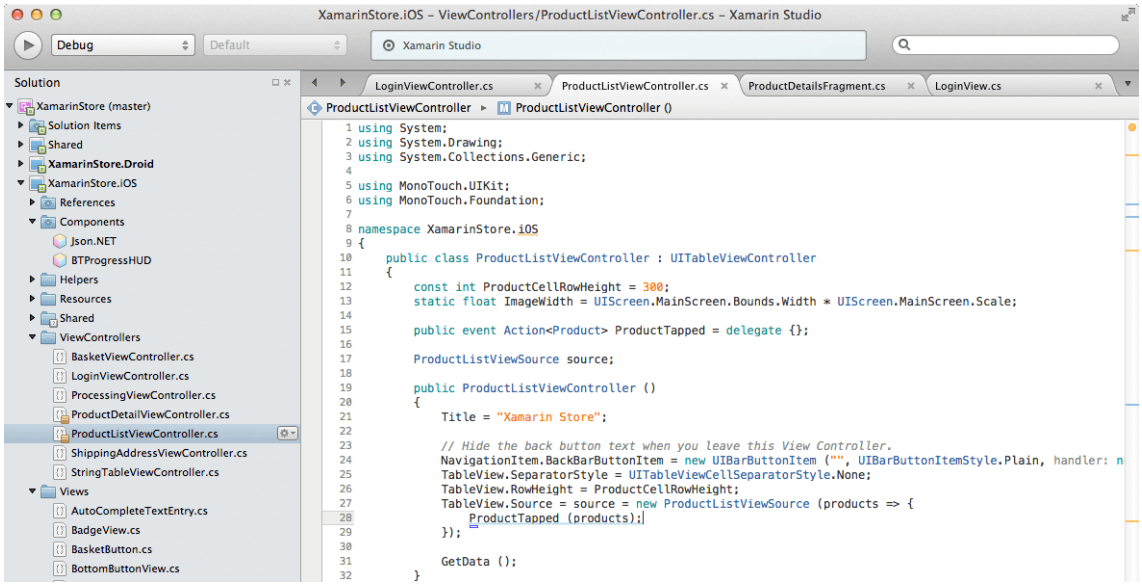


Figura 15: Entorno de Desarrollo Xamarin Studio

3.4.2 Embarcadero Delphi 10 Seattle

Embarcadero Delphi 10 Seattle [42] es una plataforma de desarrollo propietaria y no gratuita, que permite a los desarrolladores crear aplicaciones rápidamente a través de un entorno visual cómodo e intuitivo (Figura 16). La aplicación desarrollada puede ser compilada para múltiples plataformas, incluyendo Windows, Mac, Android e iOS.

Delphi 10 Seattle, al igual que Xamarin, aventaja a otros tipos de aplicaciones multiplataformas con resultados 100% nativos, con acceso total a los sensores y capacidades del dispositivo (cámara, notificaciones, GPS, acelerómetro, entre otras).

Un punto a destacar es que Delphi 10 Seattle, a través del framework de interfaces de usuario FireUI Multi-Device Designer, permite crear interfaces de usuario totalmente nativas, sin la necesidad de tener que diseñar interfaces específicas para cada plataforma en particular.

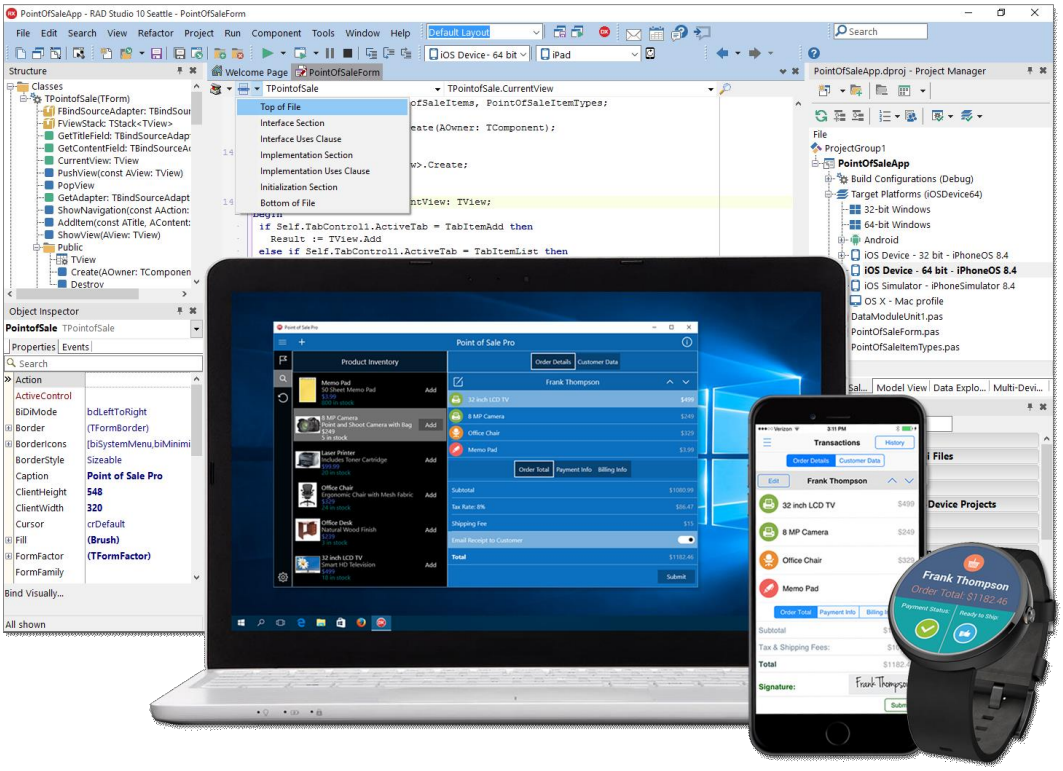


Figura 16: Entorno de Desarrollo Delphi 10 Seattle

3.4.3 RubyMotion

En Mayo de 2012, la empresa HipByte lanza RubyMontion, un proyecto que permite generar aplicaciones para iOS, Android y OS X, utilizando el lenguaje de programación Ruby.

Para poder generar las aplicaciones multiplataforma, RubyMotion posee implementaciones de Ruby sobre los sistemas operativos destinos (Android e iOS) que permiten acceder a todas las características provistas por cada plataforma. Las herramientas de compilación provistas por RubyMotion, se encargan de embeber la correspondiente implementación de Ruby al momento de compilar para cada plataforma.

La principal desventaja de RubyMotion es que sólo funciona bajo el sistema operativo OS X. Otro factor a tener en cuenta es que no posee un entorno de desarrollo integrado que facilite todo el proceso al desarrollador; esto se ve solventado gracias a que ofrece complementos para los IDE y editores de código más utilizados.

Finalmente, si bien el producto es un servicio pago, ofrece una posibilidad de probarlo gratuitamente aunque con soporte únicamente para compilar a la última versión

de cada plataforma destino.

Capítulo 4. Experimentación

Con el objetivo de profundizar la investigación de las diferentes metodologías de desarrollo de aplicaciones móviles multiplataforma, es que se llevaron a cabo diferentes experimentos en simultáneo. El primer experimento consiste en el desarrollo de una aplicación móvil utilizando cada una de las metodologías estudiadas en el capítulo anterior, con el objetivo de analizar el proceso de desarrollo. El segundo experimento tiene como objetivo analizar el rendimiento de las aplicaciones móviles generadas mediante las metodologías estudiadas, a través de un cálculo matemático. Por último, se presenta un informe comparativo que pueda ayudar en la toma de decisiones sobre qué metodología utilizar en futuros desarrollos.

4.1 Aplicación Móvil para la plataforma de E-Learning WebUNLP

4.1.1 Descripción del problema

WebUNLP es un entorno virtual de enseñanza y aprendizaje que permite a los docentes mediar sus propuestas educativas. Alumnos y docentes pueden encontrarse en ese espacio para compartir materiales de estudio, comunicarse y generar una experiencia educativa en forma virtual [46].

Actualmente, WebUNLP cuenta con una versión web enfocada a computadoras de escritorio o portátiles, pero no está adaptada para ser utilizada desde dispositivos móviles.

El desarrollo planteado consiste en extender WebUNLP, con la construcción de una aplicación móvil que permita acceder a determinadas funcionalidades del sistema, a través de un dispositivo móvil. El enfoque propuesto incluye un análisis de la misma solución, comparando el desarrollo nativo, web, híbrido, interpretado y de compilación cruzada, a fin de establecer cuál de ellos es conveniente.

Como ocurre con cualquier desarrollo de software, la construcción de una aplicación móvil implica tener claramente definido su propósito y cuáles requerimientos debe cumplir. En particular, para el caso de software para dispositivos móviles resulta esencial tener objetivos más acotados que en su versión de escritorio [47].

Para el caso específico de WebUNLP se realizó el desarrollo incremental de una aplicación móvil, y esta primera versión se enfocó en una de sus herramientas de comunicación: la cartelera. Esta herramienta permite comunicar las novedades de un curso, como, por ejemplo, el cambio de horario de una cursada, recordar las fechas de entrega de un trabajo práctico, entre otras [46].

4.1.2 Análisis

Uno de los primeros interrogantes planteados fue la elección de la plataforma. En términos de mercado los sistemas operativos que predominan en Argentina son Android e iOS [48,15], con lo cual se decidió dar soporte a estos dos sistemas operativos.

Se analizaron los requerimientos funcionales y no funcionales de forma aislada a la plataforma y luego de forma específica para cada una de ellas.

A continuación, se presentan algunos requerimientos a cumplir por la aplicación:

- El usuario debe poder ingresar a la aplicación con las mismas credenciales que las utilizadas para acceder a la versión web de escritorio.
- El usuario debe poder acceder a la cartelera de todos los cursos en los que participa, ya sea como docente o alumno.
- El usuario debe recibir una notificación en su dispositivo cuando una novedad es publicada en la cartelera. Este requerimiento no se puede cumplir en la versión web accesible desde computadoras de escritorio y/o portables.
- El usuario debe tener la misma experiencia de uso en todas las plataformas operativas.
- La aplicación web existente debe estar sincronizada con la aplicación móvil a desarrollar, esto significa que cualquier cambio realizado desde la aplicación móvil debe verse reflejado en la versión web y viceversa.

4.1.3 Diseño

Para satisfacer los requerimientos planteados en el punto anterior, a excepción de la notificación, el diseño de la aplicación móvil web consiste en una réplica de lo ofrecido por WEBUNLP, adaptándose solamente la interfaz al tamaño de pantalla del dispositivo móvil.

Para el diseño de las aplicaciones móviles nativa, híbrida, interpretada y de compilación cruzada la situación es más compleja. En la Figura 17 se presenta la arquitectura genérica de todos los componentes que participan en este escenario de desarrollo. En ella se observa el acceso desde una PC a WebUNLP, y el acceso desde dispositivos móviles a la información de WebUNLP, mediante servicios web. Para el desarrollo de los servicios web se diseñó una API Restful dada su simpleza, escalabilidad e interoperabilidad [49].

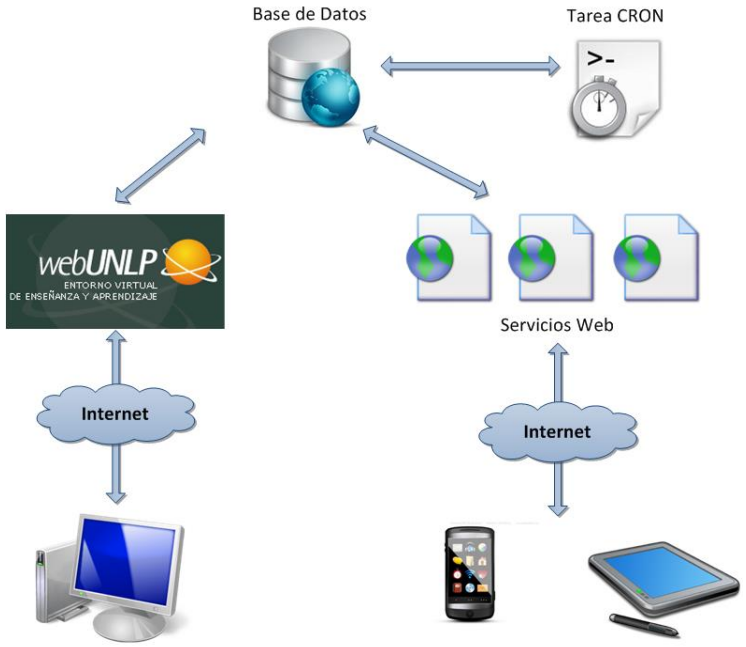


Figura 17: Arquitectura genérica para aplicación nativa e híbrida

Asimismo existe una tarea programada (Cron) de ejecución intermitente a intervalos regulares de tiempo, la cual notifica a los dispositivos móviles correspondientes cuando se crea una novedad en la cartelera de WebUNLP. El Cron identifica el sistema operativo del dispositivo a notificar y genera dicha notificación.

En cuanto a la interfaz gráfica, se planteó un diseño independiente de la plataforma para analizar los aspectos de usabilidad de la aplicación, y luego se realizaron los ajustes necesarios para cada tipo de aplicación.

En las interfaces diseñadas la navegación es en serie, en el orden en que se presentan en el mockup [50] de la Figura 18.

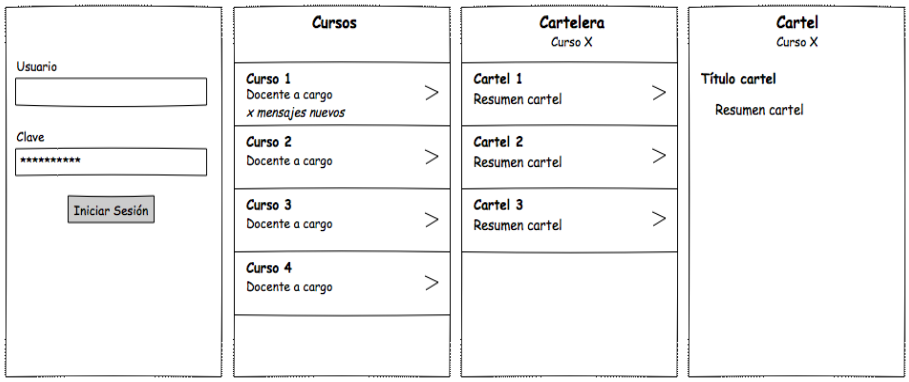


Figura 18: Mockup independiente del tipo de aplicación

4.1.4 Desarrollo

4.1.4.1 Aplicación Nativa para Android

El desarrollo de aplicaciones para la plataforma Android requiere disponer del Java Development Kit (JDK) y su entorno de programación, conocido como Android SDK. Este último provee librerías y herramientas necesarias para construir, testear y depurar aplicaciones para Android.

El desarrollo de la aplicación de WebUNLP para Android se realizó según las convenciones adoptadas por su comunidad, porque aplica buenas prácticas en la construcción de interfaces y de la lógica detrás de ellas, el acceso a datos y a servicios web. Para cumplir el requerimiento de las notificaciones en el dispositivo correspondiente, el Cron genera la notificación mediante el uso del servicio Google Cloud Messaging (GCM) [51]. En la Figura 19, se presentan las interfaces de la aplicación desarrollada.

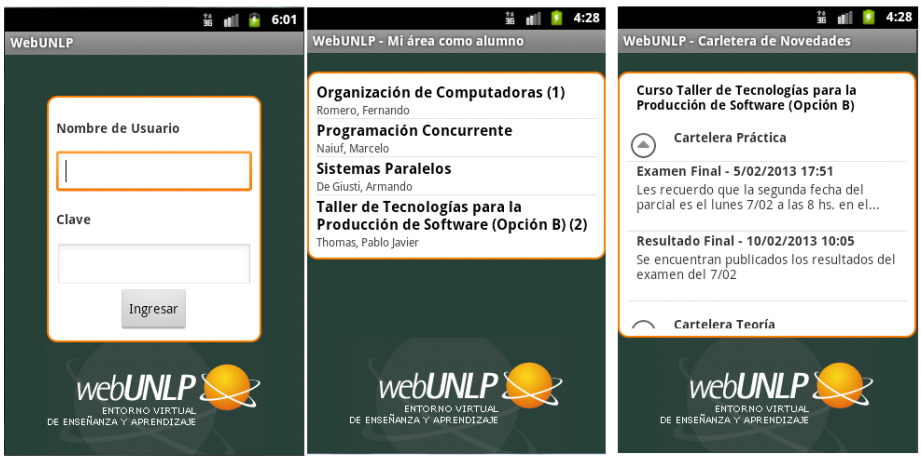


Figura 19: Aplicación nativa para Android

4.1.4.2 Aplicación Nativa para iOS

La plataforma Apple iOS está basada en un modelo propietario, es por ello que el desarrollo de una aplicación nativa iOS implica contar con una Apple Mac ejecutando OS X con Xcode instalado. El lenguaje de programación principal es Objective C. Xcode es el entorno de desarrollo de Apple para todos sus dispositivos y es el encargado de proporcionar el iOS SDK con las herramientas, compiladores y frameworks necesarios. Además, Xcode viene integrado con simuladores para dispositivos iOS (iPhones y iPads) que facilitan las etapas de prueba del sistema desarrollado.

Para los aspectos referentes a la interfaz gráfica y la interacción con el usuario, se siguieron las convenciones propuestas por Apple [52] para lograr una mejor integración de la aplicación con el sistema operativo y mejorar la experiencia del usuario.

Por último, para satisfacer el requerimiento de notificaciones, el Cron notifica al dispositivo iOS correspondiente mediante el uso del servicio Apple Push Notification Service (APNs) [53]. En la Figura 20, se presentan las interfaces de la aplicación desarrollada.

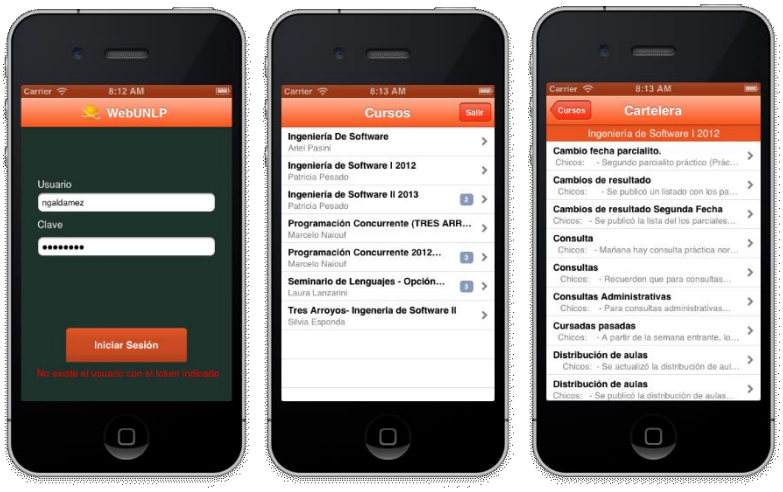


Figura 20: Aplicación nativa para iOS

4.1.4.3 Aplicación Web Móvil

Se desarrolló una aplicación web capaz de acceder a la cartelera de WebUNLP, disponible para cualquier dispositivo móvil que cuente con un navegador que soporte las características utilizadas para el desarrollo: HTML5, CSS3 y Javascript.

Como la velocidad de transmisión/recepción de datos de un dispositivo móvil a través de WiFi, y en particular 3G, es inferior a la velocidad de una computadora de

escritorio, la versión web de la cartelera de WebUNLP es liviana y gran parte de los requerimientos son implementados a través de Ajax [54] para evitar, ante algún cambio, la recarga de la página completa.

Debido a las limitaciones que tienen las aplicaciones que se ejecutan sobre un navegador, no es posible implementar la recepción de una notificación en el dispositivo cuando una novedad es publicada en la cartelera.

4.1.4.4 Aplicación Híbrida desarrollada con PhoneGap y Jquery Mobile

Para la construcción de la aplicación WebUNLP híbrida se utilizó el framework PhoneGap [27], el cual permite desarrollar aplicaciones móviles que utiliza tecnologías comunes a todos los dispositivos: HTML5, CSS y Javascript.

Asimismo, se utilizó el framework Javascript Jquery Mobile [55] para lograr interfaces con apariencia y comportamiento consistente a través de las diferentes plataformas móviles.

Con el objetivo de implementar el patrón de diseño MVC se utilizó la librería Backbone.js [36].

Por último, para satisfacer el requerimiento de notificaciones, se utilizó el plugin Pushwoosh [56].

En la figura 5, se presentan las interfaces de la aplicación desarrollada.

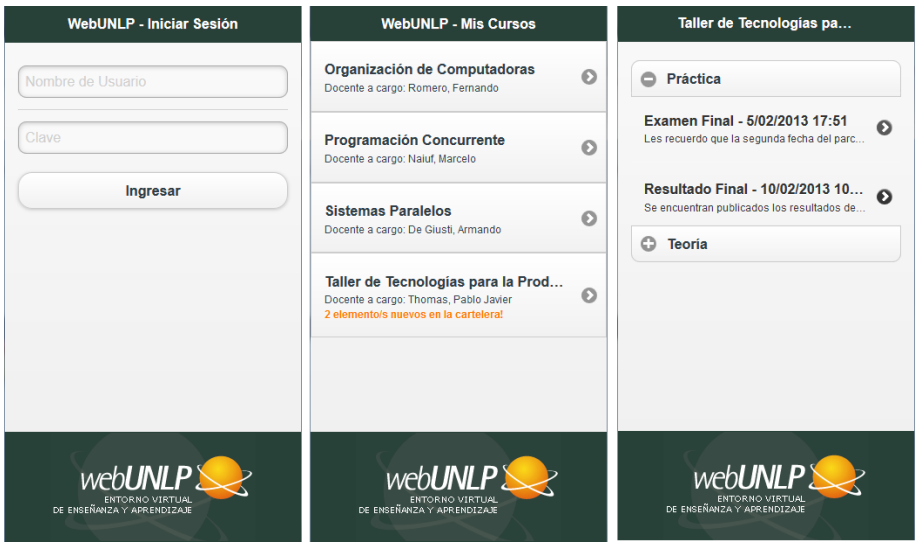


Figura 21: Aplicación híbrida desarrollada con PhoneGap

4.1.4.5 Aplicación Híbrida desarrollada con Sencha Touch

El desarrollo de WebUNLP utilizando Sencha Touch se vio favorecido por el uso del patrón de diseño MVC. Esto permitió mayor flexibilidad y legibilidad en el código [57] [34].

Sencha ofrece Sencha Command, una herramienta multiplataforma de línea de comandos que provee tareas automatizadas para todo el ciclo completo de una aplicación. En el caso de WebUNLP se utilizó tanto para la generación del proyecto como para el empaquetado de la aplicación.

En la Figura 22, se presentan las interfaces de la aplicación desarrollada.

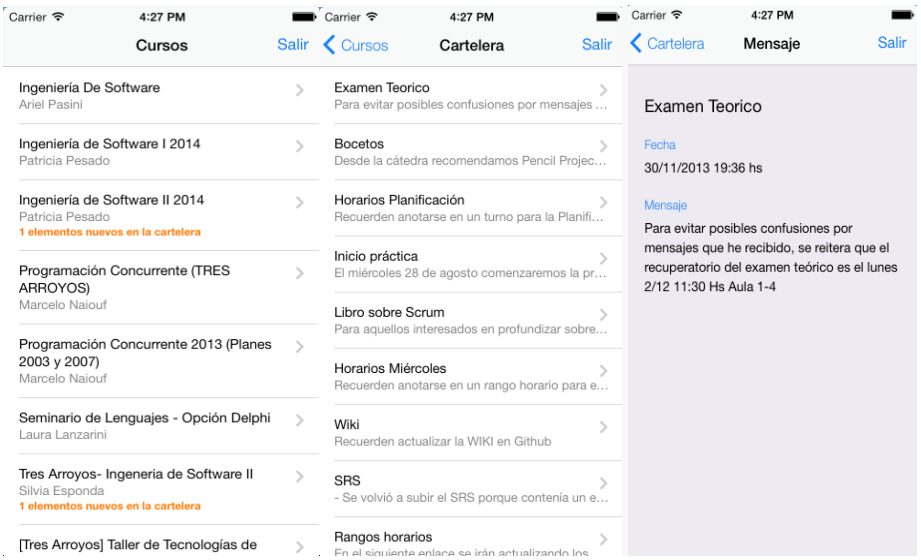


Figura 22: Aplicación desarrollada con Sencha Touch

4.1.4.6 Aplicación Interpretada de WebUNLP con Appcelerator Titanium 3

Para desarrollar una aplicación móvil experimental según el enfoque interpretado se utilizó el entorno de desarrollo gratuito y open source Appcelerator Titanium 3 [35].

Se destaca la simplicidad y legibilidad de los controladores y modelos de la aplicación desarrollada, luego de utilizar Alloy, el framework de Titanium que permite diseñar aplicaciones según el patrón de diseño MVC. Para la construcción de las vistas se puede optar por la programación mediante Javascript y la API de Titanium, o bien por una especificación XML con estilos TSS (Titanium Style Sheets). Esta última opción simplifica el proceso de creación de las vistas, aunque falta todavía un buen editor visual de interfaces que lo potencie.

Para cubrir requerimientos de notificaciones en el dispositivo, Titanium provee el módulo PushNotifications para plataformas Android e iOS.

La Figura 24 (a) muestra la experimentación con Titanium.

4.1.4.7 Aplicación Generada por Compilación Cruzada de WebUNLP utilizando Xamarin/Visual Studio

Tal como se mencionó en la sección 3.4.1, Xamarin propone un enfoque de desarrollo multiplataforma en el que se comparte la codificación completa de la lógica del negocio. Sin embargo, las interfaces deben ser programadas de manera independiente para cada una de las plataformas destino (ver Figura 23). Así, la reutilización de código, según estudios estadísticos de la compañía Xamarin, es cercana al 85%.

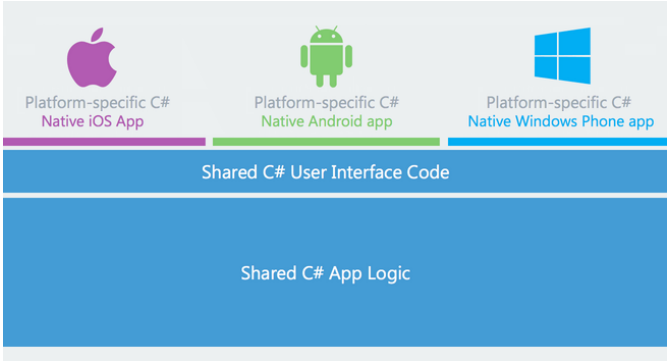


Figura 23: Enfoque único de desarrollo Xamarin

Para el estudio de las aplicaciones generadas por compilación cruzada, se desarrolló la aplicación móvil de WebUNLP utilizando Xamarin integrado con Microsoft Visual Studio. Siguiendo la estrategia más eficiente para trabajar en este entorno se ha creado una solución única conteniendo tres proyectos distintos. Uno de ellos se utilizó para generar la aplicación Android, otro para la aplicación Windwos Phone 8 y el tercero para la implementación de una biblioteca de clases portable (PCL por sus siglas en inglés) con todo el código compartido. Los tres proyectos fueron desarrollándose de manera conjunta y concurrente.

Aunque una de las ventajas más importantes de Xamarin es la reutilización de código, ésta se ve fuertemente afectada por el tipo de aplicación que se desarrolla. La relación existente entre la complejidad de la interfaz y la lógica del negocio impacta directamente sobre la posibilidad de mantener la mayor cantidad de código compartido

en la PCL. En el desarrollo de WebUNLP, la reutilización de código ha sido cercana al 50%. Sin embargo, no debe despreciarse la ventaja de utilizar el mismo lenguaje, entorno y conjunto de herramientas comunes en el desarrollo de aplicaciones para distintas plataformas móviles.

La Figura 24 (b) presenta la interfaz desarrollada para Windows Phone 8.

4.1.4.8 Aplicación Generada por Compilación Cruzada de WebUNLP utilizando Delphi 10 Seattle

Para el profundizar el estudio de las aplicaciones generadas por compilación cruzada, además se desarrolló la aplicación móvil de WebUNLP utilizando Delphi 10 Seattle. En particular, fueron de gran utilidad tanto los componentes provistos por Delphi para acceder a servicios RESTful como el entorno visual de desarrollo de Delphi 10 Seattle.

Para la recepción de notificaciones en el dispositivo de las novedades de WebUNLP se utilizó el componente TPushEvent conectado al servicio Kinvey [58].

La Figura 24 (c) presenta la interfaz de la aplicación desarrollada.

Figura 24: Aplicación desarrollada con: a) Titanium; b) Xamarin/Visual Studio y c) Delphi XE

4.1.5 Conclusiones del experimento WebUNLP

Seleccionar un dominio para desarrollar un caso experimental tiene sus particularidades. La novedad de disponer una aplicación móvil no es motivo suficiente que justifique tal desarrollo. Es necesario satisfacer un conjunto de requerimientos claramente planteados.

WebUNLP es un entorno virtual de enseñanza y aprendizaje utilizado por diversos cursos de grado y postgrado de la Universidad Nacional de La Plata. Por ende, la cantidad de usuarios beneficiados por acceder desde cualquier lugar a este entorno es importante.

De esta manera, se optó por elegir dicho espacio virtual para ser replicado en una versión móvil, que no sólo permita acercarlo a sus usuarios, sino que además amplíe sus funciones, en este caso puntualmente para la cartelera.

Para el desarrollo de esta experimentación se desarrolló la misma aplicación para los cinco tipos de enfoques de desarrollo móvil (web, nativa, híbrida, interpretada y generada por compilación cruzada).

Se destaca la gran simpleza de generar la versión web, dado que se utilizan las mismas herramientas tecnológicas que las usadas para el desarrollo de cualquier aplicación web tradicional. La principal diferencia en este sentido radica en la limitación de espacio en la pantalla del dispositivo. No obstante, el mayor contraste está dado en que no es posible acceder a todas las capacidades de hardware del dispositivo, lo que impidió implementar uno de los requerimientos, probablemente, el más interesante: la notificación de novedades de la cartelera de WebUNLP al usuario.

Por otra parte, las versiones nativas han cumplido todos los requerimientos. Aunque la mayor desventaja consiste en la no portabilidad, lo que implicó un desarrollo específico para la plataforma que se desee cubrir. En este trabajo se han presentado los desarrollos para Android e iOS, sistemas operativos más usados en Argentina, con un costo de desarrollo y mantenimiento inherente mayor.

Para el caso del enfoque híbrido, tanto la versión de WebUNLP desarrollada con PhoneGap y Jquery Mobile, como la versión desarrollada con Sencha Touch, han logrado conjugar la simpleza del desarrollo web con el uso de todas las capacidades del dispositivo. Este tipo de enfoque pretende suplir las desventajas de los dos enfoques previos, hecho que lo posiciona como una elección realmente interesante, siempre condicionada por los requerimientos específicos a cumplir.

Por el lado del modo de desarrollo interpretado, la versión de WebUNLP desarrollada con Titanium logró resolver todos los requerimientos previamente planteados. Al igual que en el enfoque híbrido, es un punto a destacar que el desarrollo de la aplicación con Titanium es relativamente simple y es posible reutilizar todo el

código entre plataformas. Lo que lo hace la elección prima facie es el hecho de obtener interfaces totalmente nativas, algo que no se lograba mediante el enfoque híbrido.

Por último, las versiones de WebUNLP generadas con el enfoque de compilación cruzada con Xamarin y Delphi, lograron cumplir los requerimientos planteados. Se destaca el hecho de que se genera interfaces totalmente nativas, pero considero que el costo es muy alto: se debe diseñar las interfaces para cada plataforma de manera independiente, lo que hace que no se pueda reutilizar todo el código fuente de la aplicación.

4.2 Enfoques de desarrollo: Análisis comparativo de rendimiento

4.2.1 Descripción del Problema

Hace apenas unas décadas atrás, el uso de los sistemas de software estaba restringido a un grupo reducido de usuarios especializados. Aquella época contrasta fuertemente con la actualidad en la que los smartphones, pequeñas computadoras móviles de propósito general, se han vuelto cotidianos y omnipresentes. Estos dispositivos pueden utilizarse para llevar a cabo tareas complejas y críticas, requiriendo la mejora continua de las capacidades de cómputo, la disponibilidad, el rendimiento eficiente, entre otras necesidades. La evolución vertiginosa de esta tecnología ejerce presión en la adaptación de la Ingeniería de Software.

Respecto del desarrollo para dispositivos móviles se debe considerar una serie de características propias de esta actividad que no estaban presentes en el desarrollo tradicional de software [59]. Es necesario tener en cuenta sobre qué tipo de dispositivos se debe ejecutar la aplicación a construir. La diversidad de plataformas, lenguajes de programación, herramientas de desarrollo, estándares, protocolos y tecnologías de red; algunas limitaciones de ciertos dispositivos y las exigencias de tiempo del mercado, entre otros, constituyen algunos de los problemas a tratar.

En la mayoría de los casos, el éxito de un producto de software para dispositivos móviles estará condicionado por el nivel de popularidad que logre alcanzar. Para maximizar su presencia en el mercado es necesario que la aplicación pueda ejecutarse sobre la mayor cantidad de plataformas móviles existentes, especialmente sobre las dos más populares: Android e iOS [60]. Para lograr este objetivo existen dos alternativas:

i) Desarrollar aplicaciones específicas para cada plataforma, impulsando en paralelo varios proyectos de desarrollo, con las herramientas y lenguajes propios de cada una de ellas. Estas aplicaciones, analizadas previamente en este trabajo, se denominan aplicaciones nativas.

ii) Generar aplicaciones que puedan ejecutarse en forma directa sobre más de una plataforma de sistema operativo a partir de un único proyecto de desarrollo. Estas aplicaciones, como ya se ha presentado en este trabajo, se denominan aplicaciones multiplataforma.

En los últimos años se ha incrementado el interés de la comunidad de Ingeniería de Software por el estudio del desarrollo de aplicaciones multiplataforma para dispositivos móviles.

En [5] se presenta un análisis comparativo de enfoques de desarrollo de aplicaciones multiplataforma para dispositivos móviles, proponiendo la siguiente taxonomía: aplicaciones web móviles, aplicaciones híbridas, aplicaciones interpretadas y aplicaciones generadas por compilación cruzada.

En [7] y [8] se analizan aspectos generales de frameworks de desarrollo multiplataforma para dispositivos móviles.

En [61] se comparan aspectos no funcionales para los diferentes enfoques de desarrollo de aplicaciones multiplataforma para dispositivos móviles.

En [9] se analizaron ventajas y desventajas de los enfoques de desarrollo multiplataforma mencionados, desde el punto de vista del Ingeniero de Software.

La elección del modo de desarrollo de aplicaciones para dispositivos móviles depende de varios factores. Uno de ellos, en muchos casos esencial, es el tiempo de ejecución. El interés por optimizar el tiempo de ejecución de una aplicación de software es intrínseco a la disciplina informática. Esto se evidencia, entre otras cosas, por la evolución del poder de cómputo de los procesadores. Además, el rendimiento eficiente es uno de los atributos que las aplicaciones de software deben satisfacer según diversos estándares de calidad, entre otros ISO/IEC 9126 e ISO/IEC 25010 [62].

En lo que respecta al desarrollo de aplicaciones para dispositivos móviles, es oportuno considerar el tiempo de ejecución de las aplicaciones, por diversos motivos.

El tiempo de ejecución de una aplicación está fuertemente ligado al consumo de

energía [63], el cual está limitado por la capacidad de la batería de los dispositivos.

Asimismo, el rendimiento de las aplicaciones se ve reflejado, principalmente, mediante las calificaciones de los usuarios en las tiendas de aplicaciones en línea. Aplicaciones con bajo rendimiento pueden implicar usuarios disconformes, produciendo publicidad negativa [60]. Andre Charland y Brian Leroux identifican al tiempo de ejecución como uno de los problemas principales a resolver en el desarrollo de aplicaciones multiplataforma, y afirman que a los usuarios finales les preocupa la calidad del software y la experiencia de uso [64].

Corral, Sillitti y Succi realizaron un análisis de rendimiento comparativo entre aplicaciones nativas y aplicaciones híbridas utilizando el framework Phonegap, para una versión del sistema operativo Android [65].

Es importante destacar que no se han encontrado estudios que evalúen y comparen el rendimiento entre los diversos modos de desarrollo multiplataforma, de acuerdo a la taxonomía propuesta en [5]: aplicaciones web móviles, aplicaciones híbridas, aplicaciones interpretadas y aplicaciones generadas por compilación cruzada. Esta taxonomía es la que se utiliza de referencia en este trabajo.

El presente apartado realiza un análisis comparativo de rendimiento de aplicaciones para dispositivos móviles generadas por el enfoque de desarrollo nativo y los distintos enfoques de desarrollo multiplataforma según [5].

En la sección siguiente se describe el experimento utilizado para evaluar tiempos de ejecución en aplicaciones nativas y en los diversos modos de desarrollo de aplicaciones móviles multiplataforma, y en las secciones posteriores se muestran y analizan los resultados obtenidos, se presentan las conclusiones y el trabajo futuro.

4.2.2 Desarrollo

4.2.2.1 Diseño de las pruebas

Todas las pruebas cuyos resultados se presentan en este trabajo fueron realizadas sobre dispositivos móviles con sistemas operativos Android e iOS, dado que en la actualidad abarcan gran parte del mercado mundial, de acuerdo a lo indicado en [60].

Para llevar a cabo la experimentación se seleccionaron los frameworks de desarrollo multiplataforma según las alternativas presentadas en [10]. Se diseñaron pruebas para evaluar el rendimiento de Apache Cordova [28], Appcelerator Titanium [35] y Xamarin

[41] que se corresponden con los modos de desarrollo híbrido, interpretado y compilación cruzada respectivamente. También se experimentó con otros dos frameworks de desarrollo multiplataforma que recientemente han ganado popularidad: NativeScript (desarrollo interpretado) y Corona (compilación cruzada). Finalmente se incluyeron al conjunto de pruebas las aplicaciones web y el enfoque nativo de desarrollo para Android e iOS. De este modo se logró una muestra razonablemente representativa de las diversas opciones existentes en la actualidad.

Para la realización de las pruebas se utilizaron seis dispositivos móviles distintos (ver Tabla 2), tres de ellos con sistema operativo Android identificados como D_{A1}, D_{A2} y D_{A3} (dos smartphone y una tablet) y tres con sistema operativo iOS identificados como D_{I1}, D_{I2} y D_{I3} (dos smartphone y una tablet).

ID	Sistema Operativo	Características
D_{A1}	Android 4.4.	Tipo: Smartphone Marca: Motorola Modelo: Moto-G2 Procesador: Quad-core 1.2 GHz Cortex-A7, RAM 1GB Snapdragon 400
D_{A2}	Android 5.0.2.	Tipo: Smartphone Marca: Samsung Modelo: S6 Procesador: Octa-core (4x2.1 GHz Cortex-A57 & 4x1.5 GHz Cortex-A53) RAM 3GB Exynos 7420 Octa
D_{A3}	Android 4.2.2	Tipo: Tablet Marca: Samsung Modelo: Tab 2 Procesador: Dual-core 1.0 GHz RAM 1GB TI OMAP 4430
D_{I1}	iOS 9.2	Tipo: Smartphone Marca: Apple Modelo: 5S Procesador: Dual-core 1.3 GHz Cyclone (ARM v8-based) RAM 1GB Apple A7
D_{I2}	iOS 9.1	Tipo: Smartphone Marca: Apple Modelo: 6 plus Procesador: Dual-core 1.4 GHz Typhoon (ARM v8-based) RAM 1GB Apple A8
D_{I3}	iOS 9.1	Tipo: Tablet Marca: Apple Modelo: Ipad Air

		Procesador: Dual-core 1.3 GHz Cyclone (ARM v8-based) RAM 1GB Apple A7

Tabla 2: Dispositivos móviles utilizados en el experimento

Se definieron siete escenarios de análisis distintos, uno por cada estrategia de desarrollo utilizada:

1. Nativo para Android y nativo para iOS
2. Aplicaciones web (multiplataforma)
3. Apache Cordova (multiplataforma hibrido)
4. Appcelerator Titanium (multiplataforma interpretado)
5. NativeScript (multiplataforma interpretado)
6. Xamarin (multiplataforma compilación cruzada)
7. Corona (multiplataforma compilación cruzada)

En los seis dispositivos se llevaron a cabo pruebas para cada uno de los siete escenarios mencionados, definiendo así un total de 42 casos de prueba.

Con el fin de evaluar la velocidad de procesamiento, se planteó un cálculo simple que incluyó varias iteraciones, funciones matemáticas y aritmética de punto flotante que se resume en la siguiente serie:

$$serie = \sum_{j=1}^{5} \sum_{k=1}^{100000} (\log_2(k) + \frac{3k}{2j} + \sqrt{k} + k^{j-1})$$

A modo de ejemplo en la Figura 25 se muestra el código multiplataforma desarrollado en Apache Cordova para el cálculo de la serie definida.

El experimento planteado permite medir con precisión la variable analizada, en este caso, el tiempo de ejecución requerido para realizar un cálculo matemático intensivo.

Este tipo de cálculo matemático es frecuente en diversas aplicaciones que se ejecutan en dispositivos móviles, por ejemplo juegos, aplicaciones con realidad aumentada, aplicaciones para tratamiento de imágenes, entre otras, en las cuales no siempre es posible utilizar el poder de la Unidad de Procesamiento Grafico (GPU) para el cálculo.

El código fuente de los experimentos llevados a cabo se encuentra disponible en [66].

En las siguientes secciones se describe la experimentación y se analizan los resultados obtenidos.

4.2.2.2 Recolección de datos

Para cada uno de los 42 casos de prueba definidos, se realizaron 30 ejecuciones independientes del experimento diseñado obteniendo en cada caso una muestra $T = T_1, T_2, \ldots T_{30}$, con T_i = tiempo requerido para el cálculo de la serie en la *i*-ésima ejecución del experimento planteado. El tiempo T_i se expresa en milisegundos.

Para caracterizar cada una de las muestras obtenidas, se han calculado los estadísticos $\bar{T}$ y S que se corresponden con la media (o promedio muestral) y la desviación estándar muestral (ver Tabla 3).

```
var startTime = new Date().getTime();
var serie = 0;
for ( var j=1; j <= 5; j++ )
{
   for ( var k=1; k <= 100000; k++ )
   {
      series = series + (Math.log(k)/Math.LN2) + (3*k/2*j) +
               Math.sqrt(k) + Math.pow(k, j-1);
   }
}
var finalTime = new Date().getTime();
var duration = finalTime - startTime;
document.getElementById('result').innerHTML = duration + ' -> ' + series;
```

Figura 25: Cálculo de serie, código multiplataforma desarrollado en Apache Cordova

Dada la muestra $T=T_1, T_2, \ldots, T_n$	
Media o promedio muestral	$\bar{T} = \left(\frac{1}{n}\right)\sum_{i=1}^{n} T_i$
Desviación estándar muestral	$S = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(T_i - \bar{T})^2}$

Tabla 3: Estadísticos utilizados en el análisis de los datos

4.2.3 Resultados Obtenidos

En la Tabla 4 se presenta una síntesis de los resultados obtenidos conformada por los valores $\bar{T}$ y S calculados para cada caso de prueba planteado. Estos valores permiten comparar el rendimiento de las aplicaciones generadas a partir de los distintos enfoques de desarrollo, evaluadas sobre cada uno de los seis dispositivos utilizados.

Los valores presentados en la Tabla 4 y graficados con diagramas de barras en la Figura 26 sugieren un análisis por separado de los casos con sistema operativo Android y los casos con sistema operativo iOS. Claramente las siluetas delineadas en los diagramas de barras de la Figura 26 se repiten de forma similar en los escenarios con el mismo sistema operativo, pero se diferencian notablemente entre distintos sistemas operativos.

		Nativo	**WebApp**	**Apache Cordova**	**Titanium**	**NativeScript**	**Xamarin**	**Corona**
D_{A1}	$\bar{T}$	532.93	186.27	230.33	211.67	187.30	395.17	1401.73
	S	16.14	6.32	14.22	24.95	9.39	8.95	12.60
D_{A2}	$\bar{T}$	211.80	90.67	85.77	95.63	89.67	211.00	600.53
	S	19.97	12.48	8.83	7.64	9.16	6.69	5.95
D_{A3}	$\bar{T}$	763.80	172.73	190.60	192.70	183.50	379.33	1344.30
	S	28.98	15.51	9.36	16.80	3.04	8.31	23.39
D_{I1}	$\bar{T}$	4.13	57.10	323.73	299.77	252.03	125.43	39.63
	S	0.78	16.55	16.62	4.01	7.28	11.03	0.85
D_{I2}	$\bar{T}$	4.13	41.90	263.97	241.13	223.43	103.03	98.63
	S	0.73	5.38	15.44	4.95	8.61	4.91	6.13
D_{I3}	$\bar{T}$	2.53	41.67	292.23	272.67	225.97	110.53	109.67
	S	0.57	4.44	10.82	5.50	2.77	3.90	2.75

Tabla 4: Síntesis de resultados obtenidos

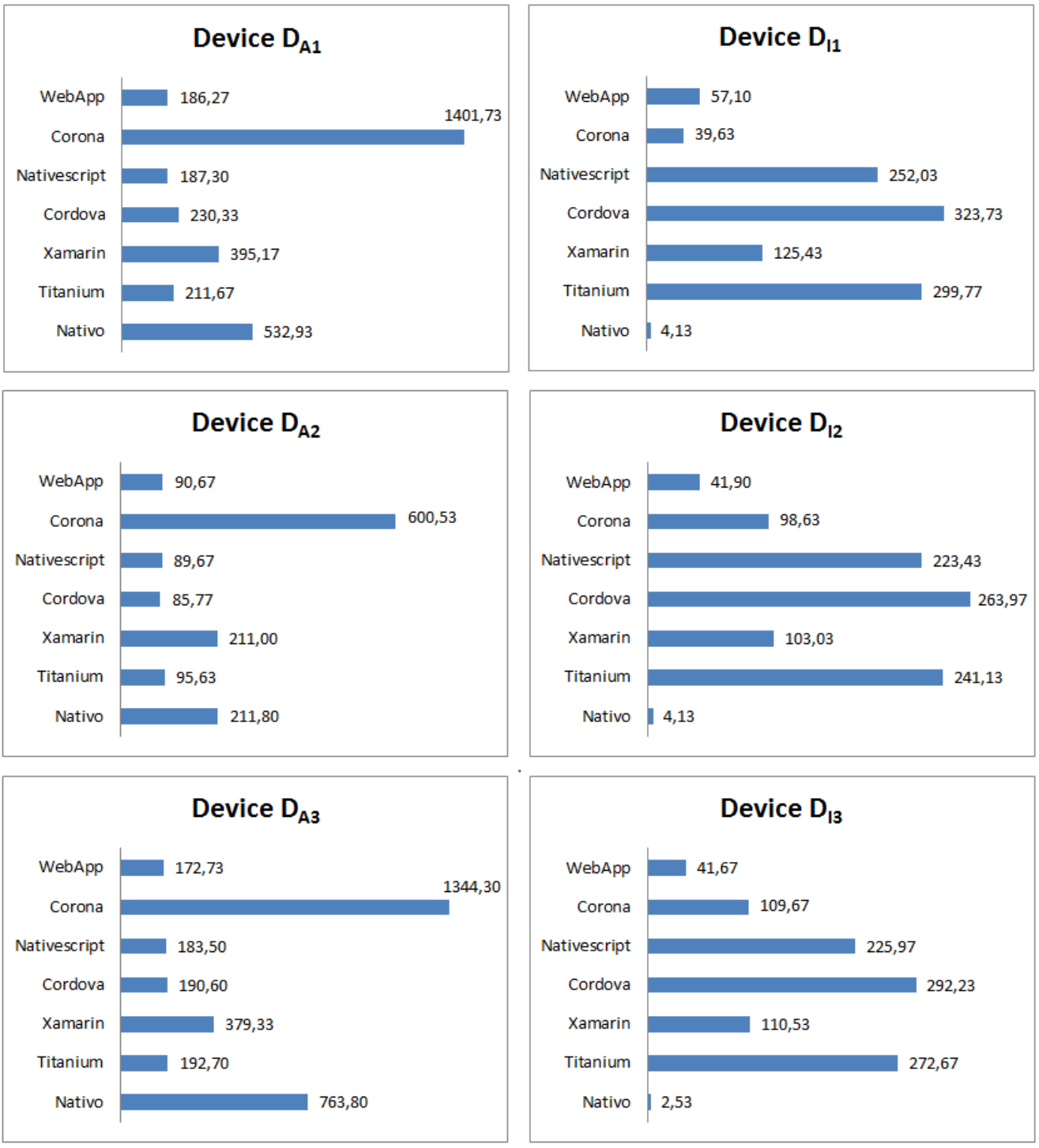


Figura 26: Diagrama de barras con las muestras recolectadas (las barras muestran el tiempo promedio expresado en milisegundos)

4.2.4 Análisis de Resultados

Debido a las diferencias de hardware de los dispositivos utilizados no es prudente comparar el rendimiento de las aplicaciones nativas en Android, con respecto a las de iOS. No obstante, de los resultados obtenidos es posible inferir que el enfoque nativo en iOS resulta mucho más eficiente que el enfoque nativo en Android. Existen diversos factores que presumen justificar este hecho, entre ellos, la diferencia inherente de ejecutar código Objective C en iOS, comparado con ejecutar código Java en Android, el cual necesita del Android Runtime (ART) para su funcionamiento, ralentizándolo.

Respecto de las aplicaciones web multiplataforma, los resultados de rendimiento resaltaron positivamente en relación al resto, tanto en Android como en iOS. Así, el enfoque de desarrollo multiplataforma web sería una opción conveniente para alcanzar un buen rendimiento en todos los dispositivos móviles, independientemente de su sistema operativo. Sin embargo, esta elección podría verse afectada por las limitaciones de estas aplicaciones para acceder en forma completa a las capacidades específicas del dispositivo.

Respecto de los enfoques híbrido e interpretado -analizados a través de las tecnologías Córdova, Titanium y NativeScript- es importante destacar que, si bien estos enfoques trabajan de modo distinto entre sí, tienen algo en común: la ejecución de código JavaScript. En este sentido, el motor de JavaScript -encargado de convertir el código JavaScript en código optimizado para que luego lo interprete un WebView- cumple un rol determinante. Las pruebas de estos enfoques en Android -que utiliza el motor de JavaScript V8- tuvieron un comportamiento similar al enfoque web, y comportamiento superior que el enfoque nativo y el enfoque de compilación cruzada. Por el contrario, en los dispositivos móviles con iOS -que utiliza el motor JavaScriptCore- los resultados de estas pruebas fueron peores que en los enfoques nativos, web y de compilación cruzada.

En los dispositivos móviles con iOS, los casos de compilación cruzada -analizados a través de las tecnologías Xamarin y Corona- obtuvieron mejores resultados que los casos híbridos e interpretados, pero mostraron peor rendimiento que el enfoque nativo y que el enfoque web. En cambio, en los dispositivos móviles con Android los resultados de las pruebas con las aplicaciones construidas mediante el enfoque de compilación cruzada fueron los peores. La necesidad de ejecución de código Java a través del Android Runtime, entre otros factores, justifican este resultado.

4.2.5 Conclusiones

Se ha presentado un estudio comparativo sobre el tiempo de procesamiento de aplicaciones de software para dispositivos móviles, generadas con distintos enfoques de desarrollo.

Los escenarios de prueba diseñados incluyeron los dos sistemas operativos para dispositivos móviles con mayor presencia en el mercado, Android e iOS, ejecutándose cada uno de ellos sobre dos smartphones (considerados de gama media y gama alta al momento en que este trabajo fue escrito) y una Tablet.

Sobre estos seis dispositivos se realizó un estudio de rendimiento para las aplicaciones construidas según los enfoques de desarrollo nativos y multiplataforma, mediante un conjunto de aplicaciones que fueron desarrolladas para tal fin:

1. Aplicación nativa para Android
2. Aplicación nativa para iOS.
3. Aplicación web (multiplataforma).
4. Aplicación Apache Cordova (multiplataforma hibrido).
5. Aplicación Appcelerator Titanium
6. (multiplataforma interpretado).
7. Aplicación NativeScript (multiplataforma interpretada).
8. Aplicación Xamarin (multiplataforma compilación cruzada).
9. Aplicación Corona (multiplataforma compilación cruzada).

En todos los dispositivos con sistema operativo Android el peor rendimiento fue evidenciado por Corona, seguido por el enfoque nativo y Xamarin. Esta situación es totalmente opuesta a los resultados obtenidos sobre el sistema operativo iOS en dónde las tecnologías de desarrollo mencionadas se mostraron siempre entre las cuatro mejores. En particular es destacable la supremacía del enfoque nativo frente a todos los demás para el caso de los dispositivos con iOS.

Respecto al mejor rendimiento en los dispositivos Android no es posible identificar claramente cuál es el enfoque más conveniente. Tanto las aplicaciones web como Córdova, NativeScript y Titanium alcanzaron buenos resultados en relación al resto. Nuevamente esta situación se contrapone con los datos obtenidos sobre los dispositivos iOS en los que Córdova, Titanium y NativeScript obtuvieron las peores mediciones entre las tecnologías utilizadas. No ocurrió lo mismo con las aplicaciones web en iOS, que también produjeron buenos resultados en relación al resto.

En líneas generales, independientemente del sistema operativo, las aplicaciones web han mostrado un buen rendimiento si se consideran todos los casos analizados.

Actualmente al desarrollar un sistema de software, existe la posibilidad de generar su versión para dispositivos móviles. Esta tarea muchas veces no es simple y una de las decisiones más importantes a tomar, es elegir el enfoque de desarrollo.

A partir de este trabajo, se dispone de un indicador de rendimiento que puede ser útil para el Ingeniero de Software que debe seleccionar un enfoque de desarrollo de software para dispositivos móviles.

Por otra parte, no se han encontrado trabajos de otros autores donde se haya analizado el rendimiento en todos los enfoques de desarrollo de software para dispositivos móviles. Este tema, tal como se indicó en la introducción, es importante para la comunidad de Ingeniería de Software, y los trabajos analizados hasta el momento, se han concentrado solo en los enfoques nativos y/o híbridos.

4.3 Análisis comparativo de enfoques de Desarrollo de Aplicaciones Móviles

Como se estudió en las secciones anteriores, existen diferentes enfoques de desarrollo de aplicaciones móviles.

Una opción es desarrollar aplicaciones móviles nativas, las cuales hacen uso de todas las capacidades del dispositivo móvil y permiten una alta experiencia de usuario, debido a que las interfaces de usuario se componen por componentes nativos, resultando semejantes al resto de las interfaces del sistema operativo. No obstante, el desarrollo no puede ser reutilizado para dar soporte a otras familias de sistemas operativos, implicando mayores costos de desarrollo y mantenimiento.

En contraposición al desarrollo nativo, surgió el desarrollo multiplataforma, dando la posibilidad de desarrollar un único producto y ejecutarlo en diferentes plataformas. En este sentido, se estudiaron diferentes enfoques: desarrollo web móvil, híbrido, interpretado y generado por compilación cruzada. Cada uno de estos enfoques cuenta con ciertas características que, naturalmente, los hacen diferenciar, con ventajas y desventajas.

Se identificaron diversos factores que pueden ser utilizados para analizar las

aplicaciones que se generan mediante los enfoques de desarrollo nativo y multiplataforma, los cuales se detallan a continuación:

- **Experiencia de usuario:** conjunto de factores y elementos que hacen referencia al nivel de satisfacción total de los usuarios cuando utilizan un producto o sistema. El resultado es la generación de una percepción positiva o negativa de dicho servicio, producto o dispositivo.
- **Plataformas alcanzadas**: se ha visto en capítulos anteriores que existen distintas familias de sistemas operativos. Según el modo de desarrollo elegido, se puede desarrollar aplicaciones para una plataforma específica o por varias.
- **Costo del desarrollo**: el desarrollo puede ser más o menos costoso según si requiere codificar las soluciones de forma específica para cada sistema operativo o si es posible la reutilización de código.
- **Costo del mantenimiento**: la corrección de errores o agregados de nuevas funcionalidades puede requerir codificar de forma específica para cada sistema operativo. Además, se debe tener en cuenta si deben convivir diferentes versiones del mismo producto
- **Entorno de desarrollo integrado:** Software que asiste al programador en la construcción de aplicaciones.
- **Interfaces de usuario**: factor que analiza los tipos de componentes utilizados para construir las interfaces de usuario. Como se mencionó al analizar los distintos enfoques de desarrollo, la elección de la herramienta a utilizar incide en si las interfaces estarán compuestas mediante componentes nativos o componentes web, los cuales pueden ser decorados para simular componentes nativos.
- **Acceso total al dispositivo**: posibilidad de acceder mediante las herramientas de desarrollo a todas las capacidades del dispositivo, como la cámara, sensores, entre otros.
- **Modo de instalación**: se refiere a si la aplicación puede ser instalada desde las tiendas de aplicaciones o es accesible sin instalación desde un navegador de Internet.
- **Rendimiento**: factor que analiza la ejecución de tareas y los tiempos asociados

en su resolución.

- **Uso offline**: capacidad que tiene la aplicación de funcionar sin estar conectada a internet.
- **Distribución a través de las tiendas**: posibilidad de descargar la aplicación desde las tiendas de aplicaciones.
- **Categorías de aplicaciones a desarrollar**: tipo de aplicación que se desea desarrollar.
- **Porcentaje de código a reutilizar**: Cantidad de líneas de código con respecto al total, que pueden ser reutilizadas para en la generación de aplicaciones multiplataforma.

Con el objetivo de estudiar cómo se comportan e inciden estos factores en los distintos enfoques de desarrollo, se realizó un estudio comparativo de los diferentes criterios a tener en cuenta. Dicho estudio se sintetiza en la Tabla 5.

Factor a analizar	Desarrollo Nativo	Desarrollo Web	Desarrollo Híbrido	Desarrollo Interpretado	Desarrollo Generado por Compilación Cruzada
Experiencia de Usuario	Alta	Muy Baja	Baja	Alta	Alta
Plataformas alcanzadas	Se desarrolla exclusivamente para una plataforma destino	Cualquier plataforma. Se necesita solo un navegador y acceso a internet	Es posible compilar para Android, iOS, Windows Phone	Es posible compilar para Android, iOS, Windows Phone	Es posible compilar para Android, iOS, Windows Phone
Costo del desarrollo	Muy Alto	Muy Bajo	Bajo	Medio	Medio a Alto
Costo del mantenimiento	Muy Alto	Muy Bajo	Medio	Alto	Alto a Muy Alto
Entorno de desarrollo integrado	IDE único por SO	Múltiples opciones	Plugins para IDE genéricos	Algunos frameworks disponen de IDE específico	IDE único por framework
Interfaces de usuario	Nativas	Web	Web	Nativas	Nativas
Acceso total al dispositivo	Si	No	Si	Si	Si
Modo de instalación	Descargar desde la	Sin instalaciones	Descargar desde la	Descargar desde la	Descargar desde la

Factor a analizar	Desarrollo Nativo	Desarrollo Web	Desarrollo Híbrido	Desarrollo Interpretado	Desarrollo Generado por Compilación Cruzada
	tienda de aplicaciones e instalar		tienda de aplicaciones e instalar	tienda de aplicaciones e instalar	tienda de aplicaciones e instalar
Rendimiento	Muy Alto	Muy Bajo	Bajo a Medio	Alto	Medio a Alto
Uso offline	Si	No	Si	Si	Si
Distribución a través de las tiendas	Si	No	Si	Si	Si
Categoría de aplicaciones a desarrollar	Aplicaciones que demandan muchos recursos o que requiere una alta experiencia de usuario	Sitios web existentes	Sitios web encapsulados como aplicación, para ser distribuidas a través de tiendas	Aplicaciones simples a complejas	Aplicaciones simples a complejas
Porcentaje de Código a reutilizar	0%	100%	Cerca del 100%	Cerca del 90%	Entre el 60 y 85%

Tabla 5: Análisis comparativo de enfoques de desarrollo de aplicaciones móviles

Capítulo 5. Conclusiones

Inicialmente los dispositivos móviles fueron pensados y diseñados con un propósito particular. A través de los años, el crecimiento tecnológico ha permitido incorporar funcionalidades adicionales, lo que ha posibilitado expandir el marco de uso.

Actualmente, el poder de cómputo subyacente en una gran variedad de dispositivos móviles ha generado nuevas posibilidades, lo que constituye un reto para los ingenieros de software.

Asimismo, y tal como se mencionó en capítulos anteriores, existen diversos sistemas operativos para dispositivos móviles, siendo los más populares Android, iOS y Windows Phone. Debido a que el éxito de un producto de software está vinculado, en gran parte, a su uso masivo, surge la necesidad de desarrollar aplicaciones móviles teniendo en cuenta cada una de estas plataformas.

El desarrollo de software para dispositivos móviles tiene características propias que difieren del desarrollo tradicional. Esta actividad reciente, impensada décadas atrás, marca un antes y un después en la evolución de la Ingeniería de Software. Es de esperar que dicha evolución continúe, y por consiguiente, que los enfoques de desarrollo para dispositivos móviles cambien o sean reemplazados por otros. Es claro que, por el momento, existen cinco posibilidades de desarrollo de una misma aplicación móvil:

- Aplicaciones nativas
- Aplicaciones web
- Aplicaciones híbridas
- Aplicaciones interpretadas
- Aplicaciones generadas por compilación cruzada.

Para un completo análisis de cada enfoque de desarrollo es que se diseñaron una serie de casos de estudio. El primero de ellos consistió en desarrollar una aplicación móvil para cada enfoque de desarrollo presentado. Teniendo este primer objetivo definido, se decidió extender el entorno virtual de enseñanza y aprendizaje WebUNLP, a través de una aplicación móvil enfocada en la cartelera de novedades de los cursos. La aplicación

móvil debía permitir el ingreso de alumnos y docentes con las mismas credenciales que las utilizadas en la versión web, y debía mostrar información sincronizada con la web, sobre las novedades publicadas en las carteleras de los cursos. Asimismo, cuando los docentes publiquen novedades, aquellos alumnos pertenecientes al curso deben recibir una notificación en su dispositivo móvil.

Luego de desarrollar las aplicaciones móviles para cada enfoque de desarrollo se pueden elaborar algunas conclusiones.

Se destaca la simpleza a la hora de generar la versión web móvil pero se observan limitaciones al no poder acceder a todas las capacidades de hardware del dispositivo.

El desarrollo de las aplicaciones nativas resultó óptimo si se lo compara con los requerimientos oportunamente planteados, con la desventaja de que las aplicaciones no son portables, implicando desarrollos específicos para cada plataforma que se desee cubrir. Por otro lado, el desarrollo de las aplicaciones hibridas –utilizando PhoneGap y Sencha- lograron conjugar la simpleza del desarrollo web con el uso de todas las capacidades del dispositivo, posicionándolo como una elección realmente interesante. De igual manera, la aplicación interpretada desarrollada mediante Titanium, logró resolver todos los requerimientos previamente planteados; se destaca que el desarrollo fue relativamente simple y es posible reutilizar todo el código entre plataformas. El hecho de obtener interfaces totalmente nativas la posiciona como la alternativa ideal. Por último, las aplicaciones generadas con el enfoque de compilación cruzada (Delphi y Xamarin) lograron cumplir los requerimientos planteados; la obtención de interfaces totalmente nativas es algo a destacar, pero conlleva un alto costo, ya que se debe diseñar las interfaces para cada plataforma de manera independiente.

El segundo experimento del presente trabajo, consistió en analizar el rendimiento de las aplicaciones generadas mediante cada enfoque de desarrollo. En total se desarrollaron, mediante distintas tecnologías, nueve aplicaciones. Las pruebas se realizaron en los dos sistemas operativos de mayor relevancia, Android e iOS, a través de seis dispositivos (cuatro smartphones y dos tablets).

En todos los dispositivos con sistema operativo Android el peor rendimiento se encontró en la aplicación generada por Corona, seguido por el enfoque nativo y Xamarin. En cambio, estas tecnologías se mostraron siempre entre las cuatro mejores sobre el sistema operativo iOS.

Respecto al mejor rendimiento en los dispositivos Android no es posible identificar claramente cuál es el enfoque más conveniente. Tanto las aplicaciones web como Córdova, NativeScript y Titanium obtuvieron buenos resultados de rendimiento en relación al resto. Nuevamente esta situación se contrapone con los datos obtenidos sobre los dispositivos iOS en los que Córdova, Titanium y NativeScript obtuvieron las peores mediciones entre las tecnologías utilizadas. No ocurrió lo mismo con las aplicaciones web en iOS, que también produjeron buenos resultados en relación al resto.

En líneas generales, independientemente del sistema operativo, las aplicaciones web han mostrado un buen rendimiento si se consideran todos los casos analizados.

Por último, se elaboró un análisis comparativo de las características inherentes a cada enfoque de desarrollo. Se identificaron ciertos factores de interés que la comunidad de ingeniería de software considera al elegir qué tecnología utilizar, y se analizó su impacto en cada enfoque de desarrollo.

Capítulo 6. Trabajo Futuro

Si bien se estudiaron en el presente trabajo diversas tecnologías para el desarrollo de aplicaciones móviles, se planea profundizar el estudio analizando otras herramientas multiplataforma disponibles en el mercado, como Ionic [32], NativeScript [38], Applause [40], React Native [67], entre otras.

Además, se plantea como línea de trabajo futuro ampliar el análisis comparativo de rendimiento, incluyendo otros aspectos del desarrollo para dispositivos móviles; como por ejemplo el acceso a disco, el consumo de batería y el tiempo de renderización de imágenes.

Por último, es deseable estudiar la experiencia de usuario en las aplicaciones generadas por cada enfoque de desarrollo de aplicaciones para dispositivos móviles

Bibliografía

[1] A. Talukder, H. Ahmed, and R. Yavagal, *Mobile Computing, Technology, Applications, and Service Creation*, Tata McGraw-Hill, Ed., 2010.

[2] Abrahamsson, P. et al., "Mobile-D: An Agile Approach for Mobile Application Development.," in *In Companion To the 19th Annual ACM SIGPLAN Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA '04)*, Vancouver, Canada, 2004, pp. 174-175.

[3] P. Abrahamsson, "Mobile software development - the business opportunity of today," in *Proceedings of the International Conference on Software Development*, Reykjavik, 2005.

[4] I. S. Hayes, *Just Enough Wireless Computing*.: Prentice Hall Professional Technical, 2002.

[5] Stelios Xinogalos Spyros Xanthopoulos, "A Comparative Analysis of Cross-platform Development Approaches for Mobile Applications," in *Proceedings of the 6th Balkan Conference in Informatics*, Thessaloniki, Greece, 2013, pp. 213-220.

[6] Tolety S.B Raj R., "A study on approaches to build cross-platform mobile applications and criteria to select appropriate approach," in *India Conference (INDICON), 2012 Annual IEEE*, India, 2012.

[7] Yonathan Aklilu Redda, "Cross platform Mobile Applications Development," in *Master in Information Systems, Norwegian University of Science and Technology*, Norway, 2012.

[8] Datta S.K., Bonnet C. Nikaein N. Dalmasso I., "Survey, comparison and evaluation of cross platform mobile application development tools," in *Wireless Communications and Mobile Computing Conference (IWCMC)*, Cagliari, Sardinia, Italia, 2013.

[9] Lisandro Delía, Nicolás Galdámez, Pablo Thomas, and Patricia Pesado, "An Experimental Analysis of Application Types for Mobile Devices," in *Computer Science & Technology Series - XIX Argentine Congress of Computer Science - Selected Papers*.: Editorial de la Universidad de La Plata, 2014, pp. 173 -183.

[10] Lisandro Delía, Nicolás Galdamez, Pablo Thomas, Leonardo Corbalán, and Patricia Pesado, "Multi-Platform Mobile Application Development Analysis," in *IEEE Ninth International Conference on Research Challenges in Information Science - IEEE RCIS*, Atenas, Grecia, May 2015.

[11] Telam. [Online]. http://www.telam.com.ar/notas/201504/102073-telefonia-movil-lineas-activas.html

[12] RTVE. [Online]. http://www.rtve.es/noticias/20110212/evolucion-del-telefono-movil-del-zapatofono-smartphones/404523.shtml

[13] Vodafone. [Online]. http://blog.vodafone.co.uk/2013/08/29/vodafone-uks-ceo-guy-laurence-4g-finally-has-a-purpose/

[14] Muy Canal. [Online]. http://www.muycanal.com/2014/01/31/futuro-del-telefono-movil

[15] Statcounter. [Online]. http://gs.statcounter.com/

[16] Anthony I. Wasserman, "Software Engineering Issues for Mobile Application Development," 2010.

[17] Statista. [Online]. http://www.statista.com/

[18] Lisandro Delía, Nicolás Galdamez, Pablo Thomas, Leonardo Corbalán, and Patricia Pesado, "Análisis Experimental de desarrollo de Aplicaciones Móviles Multiplataforma," in *XX Congreso Argentino de Ciencias de la Computación CACIC 2014*, San Justo, Argentina, octubre 2014.

[19] Lisandro Delía, Nicolás Galdamez, Pablo Thomas, and Patricia Pesado, "Un Análisis Experimental de Tipo de Aplicaciones para Dispositivos Móviles," in *XIX Congreso Argentino de Ciencias de la Computación CACIC 2013*, Mar del Plata, Argentina, 2013.

[20] Abhinay Puvvala, Amitava Dutta, Rahul Roy, and Priya Seetharaman, "Mobile Application Developers' Platform Choice Model," in *2016 49th Hawaii International Conference on System Sciences (HICSS)*, Koloa, Hawaii, 2016.

[21] Android Dashboard. [Online]. https://developer.android.com/about/dashboards/index.html

[22] Xcode. [Online]. https://developer.apple.com/xcode/

[23] K.W. Tracy, "Mobile Application Development Experiences on Apple's iOS and Android OS," *Potentials, IEEE*, vol. 31, no. 4, July-Aug 2012.

[24] Google. Recommendations for building smartphone-optimized websites. [Online]. http://googlewebmastercentral.blogspot.com.ar/

[25] Bootstrap. [Online]. http://getbootstrap.com/

[26] Foundation. [Online]. http://foundation.zurb.com/

[27] PhoneGap. [Online]. http://phonegap.com/

[28] Apache Cordova. [Online]. https://cordova.apache.org/

[29] Adobe PhoneGap Build. [Online]. https://build.phonegap.com/

[30] CocoonJS. [Online]. https://cocoon.io/

[31] Ludei. [Online]. https://www.ludei.com/

[32] Ionic. [Online]. http://ionicframework.com/

[33] Sencha. [Online]. https://www.sencha.com/products/touch/#overview

[34] A Kosmaczewski, *Sencha Touch 2 Up and Running*.: O'Reilly, 2013.

[35] Appcelerator Titanium. [Online]. http://www.appcelerator.com/product/

[36] Backbone. [Online]. http://backbonejs.org/

[37] Underscore.js. [Online]. http://underscorejs.org/

[38] NativeScript. [Online]. https://www.nativescript.org/

[39] TypeScript. [Online]. https://www.typescriptlang.org/

[40] applause. [Online]. http://www.applause.com/

[41] Xamarin. [Online]. www.xamarin.com/

[42] Delphi. [Online]. https://www.embarcadero.com/products/delphi

[43] RubyMotion. [Online]. http://www.rubymotion.com/

[44] Xtext. [Online]. http://www.eclipse.org/Xtext/

[45] Mono. [Online]. http://www.mono-project.com/

[46] [Online]. https://webunlp.ead.unlp.edu.ar/

[47] I. Salmre, *Writing Mobile Code Essential Software Engineering for Building Mobile Applications*.: Addison Wesley Professional, 2005.

[48] Mapbox. [Online]. https://www.mapbox.com/labs/twitter-gnip/brands/

[49] Ruby S. Richardson L., *RESTful Web Services*.: O'Reilly Media, 2007.

[50] Matthew J. Hamm, *Wireframing Essentials*.: Packt Publishing, 2014.

[51] GCM. [Online]. https://developers.google.com/cloud-messaging/

[52] iOS Human Interface Guidelines. [Online]. https://developer.apple.com/ios/human-interface-guidelines/overview/design-principles/

[53] APN. [Online]. http://developer.apple.com/library/mac/#documentation/NetworkingInternet/Conceptual

/RemoteNotificationsPG/Chapters/ApplePushService.html#//apple_ref/doc/uid/TP40008194-CH100-SW9

[54] Mark Norm Norman Francis Christian Heilmann, *Web Development Solutions: Ajax, APIs, Libraries, and Hosted Services Made Easy*.: Apress, 2007.

[55] JqueryMobile. [Online]. https://jquerymobile.com/

[56] Pushwoosh. [Online]. https://github.com/Pushwoosh/pushwoosh-phonegap-plugin

[57] J. Clark, *Creating Mobile Apps with Sencha Touch 2*.: Packt Publishing, 2013.

[58] Kinvey. [Online]. http://www.kinvey.com/

[59] Ali Mesbah, Philippe Kruchten Mona Erfani Joorabchi, "Real Challenges in Mobile App Development," in *ACM / IEEE International Symposium on Empirical Software Engineering and Measurement*, Baltimore, Maryland, US, 2013.

[60] André Nitze, Andreas Schmietendorf Florian Rösler, "Towards a Mobile Application Performance Benchmark," in *ICIW 2014: The Ninth International Conference on Internet and Web Applications and Services*, Paris, France., 2014.

[61] Sebastian Hanschke and Tim A. Majchrzak Henning Heitkötter, "Comparing Cross Platform Development approaches For Mobile Applications," in *8th International Conference on Web Information Systems and Technologies (WEBIST)*, Porto, Portugal, 2012.

[62] H.W, Kim, S.G., Chung, C.S. Jung, "Measuring Software Quality: A Survey of ISO/IEC 9126. IEEE Software," vol. 21, no. 5, pp. 88 – 92, September/October 2004.

[63] Luis Corral, Anton B. Georgiev, Alberto Sillitti, and Giancarlo Succi, "Can execution time describe accurately the energy consumption of mobile apps? An experiment in Android.," *GREENS 2014 Proceedings of the 3rd International Workshop on Green and Sustainable Software*, pp. 31-37, June 2014.

[64] Brian Leroux Andre Charland, "Mobile application development: web vs. native.," *Magazine Communications of the ACM*, vol. 54, no. 5, pp. 49-53, May 2011.

[65] Luis Corral, Alberto Sillitti, and Giancarlo Succi, "Mobile multiplatform development: An experiment for performance analysis," in *The 9th International Conference on Mobile Web Information Systems (MobiWIS)*, Ontario, Canada, 2012.

[66] Repositorio Git III-LIDI. [Online]. https://gitlab.com/iii-lidi/performance-assessment-multiplatform-mobile-applications/tree/master

[67] React Native. [Online]. https://facebook.github.io/react-native/

[68] Lisandro, Galdámez, Nicolás, Corbalán, Leonardo, Thomas, Pablo, Pesado, Patricia Delía, "Un análisis comparativo de rendimiento en aplicaciones móviles

multiplataforma," in *XXI Congreso Argentino de Ciencias de la Computación*, Junín, 2015.