UNIVERSIDAD POLITECNICA DE VALENCIA

DSIC
DEPARTAMENTO DE SISTEMAS INFORMÁTICOS Y COMPUTACIÓN

MASTER DE INGENIERIA DE SOFTWARE, METODOS FORMALES Y SISTEMAS DE INFORMACIÓN

Entornos de desarrollo multiplataforma para clientes móviles de aprendizaje ubicuo

Javier Naranjo Sanhermelando

Dirigida por:

Natividad Prieto Sáez
Juan Carlos Ruiz García
David de Andrés Martínez

Septiembre 2011

Índice de contenidos

Índice de Figuras

Índice de Tablas

Capítulo 1

1. Introducción

1.1 Contexto y motivación

El mundo en el que vivimos ha cambiado debido a los recientes progresos en las tecnologías móviles y sus servicios asociados. Debido al gran avance tecnológico de los últimos años, los teléfonos móviles se han convertido en ordenadores en miniatura. Todo esto ha propiciado la aparición de nuevas formas de negocio basadas en los servicios móviles, los cuales tienen un enorme potencial económico para la competitividad, la innovación y el crecimiento. Este sector ofrece un gran potencial para la innovación, ya que implica el uso de nuevas tecnologías, nuevos proceso y nuevas formas de interactuar con el cliente.

En este contexto, el proyecto MOBIP va dirigido a facilitar el acceso a inversores, explotar el potencial de la cooperación empresarial bilateral, así como el acceso a las grandes empresas del sector como fuente de capital y de acuerdos societarios [25]. El objetivo de MOBIP es dar soporte y promover la competitividad de empresas con gran crecimiento en el sector de los servicios móviles, y reforzar sus oportunidades de expansión y acceso al mercado tanto en Europa como en otros mercados internacionales.

En este ámbito, el Instituto Universitario de Aplicaciones de las TIC Avanzadas (ITACA), subcontratado por el Instituto de la Mediana y Pequeña Industria Valenciana (IMPIVA), juega dos papeles principales. En primer lugar, actúa como animador tecnológico, desarrollando un repositorio de tecnologías, oportunidades y ámbitos de negocio, que pondrá a disposición de las pequeñas y medianas empresas (PyME) para facilitar y promover la innovación en sus productos en el sector de los servicios móviles. En segundo lugar, y en conjunción con la PyME Samoo S.L., empresa que ofrece servicios tecnológicos y asesoramiento en soluciones *e-learning*, especialmente Sakai, se pretende desarrollar un portal de soporte a la formación en nuevas tecnologías para las PyME. Por su experiencia y ámbito de negocio, es la empresa Samoo S.L. es la que desarrollara la plataforma de *e-learning* de MOBIP basada en Sakai, mientras que es ITACA la que realizará el desarrollo de los contenidos que deba proporcionar este portal.

De esta manera, y debido a la necesidad de dotar a MOBIP de una herramienta que de soporte a la formación de sus usuarios, Sakai será la solución *e-learning* elegida

para cubrir las necesidades de formación *on-line*. Sakai es un *Learning Management System* (LMS) avalado por las mejores universidades del mundo (MIT, Stanford, Yale, Oxford, Cambridge, etc.) y respaldado por una amplia comunidad de expertos. Está realizado con tecnología Java lo que le proporciona una flexibilidad que permite su utilización en organizaciones con todo tipo de requisitos, desde pequeñas y medianas empresas a grandes universidades y administraciones [28]. El software Sakai posee múltiples funcionalidades de comunicación entre profesores y alumnos, como la publicación de anuncios, distribución de material docente, aviso sobre determinados eventos, gestión de trabajos, etc.

Así pues, dentro de este marco de trabajo, y dada la gran importancia de facilitar el acceso a los contenidos formativos desplegados en la plataforma de *e-learning* en cualquier momento y en cualquier lugar, se plantea la necesidad de proporcionar acceso a través de dispositivos móviles. En la actualidad, el acceso a Sakai se realiza mediante el navegador Web del dispositivo móvil, de la misma manera que lo haríamos desde un ordenador de sobremesa. Esto presenta el inconveniente de que la información que se muestra no está adaptada a las características especificas de cada dispositivo móvil. Por todo ello y para dotar de mayor accesibilidad y difusión a los resultados esperados de este proyecto, Samoo S.L. determina la necesidad de desarrollar clientes adaptados a las características de los dispositivos móviles que accedan a los recursos formativos disponibles y que faciliten así el consumo de los mismos.

Por otro lado, la proliferación de teléfonos de nueva generación, llamados comúnmente *smartphones*, ha provocado la explosión de un nuevo mercado de aplicaciones móviles, convirtiéndose en unos de los mercados más dinámicos y lucrativos dentro del desarrollo software. La fragmentación del mercado actual dada la gran variedad de dispositivos móviles (teléfonos, tabletas, etc) y sistemas operativos (iOS, Android, BlackBerry, etc) existentes, hace inviable el desarrollo de aplicaciones adaptadas a cada una de estas plataformas. Entre los distintos elementos que complican en gran medida el desarrollo podemos citar los siguientes: distintos lenguajes de programación, variable resolución de las pantallas, interacción del usuario mediante teclado o de manera táctil, diferentes tiendas de aplicaciones, etc.

Para hacer frente a la fragmentación, en los últimos años, han aparecido multitud de *Frameworks* multiplataforma de desarrollo de aplicaciones móviles. Dentro de las distintas plataformas existentes hay una gran diversidad: plataformas de pago, plataformas que soportan muchos dispositivos, plataformas que generan aplicaciones totalmente nativas, plataformas basadas en tecnologías Web, etc. Cada uno de estos entornos multiplataforma está basado en distintos fundamentos y tiene sus puntos fuertes y sus puntos débiles. En problemáticas como la del desarrollo multiplataforma no existen soluciones perfectas. La idoneidad de una plataforma dependerá no solo de sus propiedades intrínsecas, sino del tipo de aplicación que se quiera desarrollar. Las características a estudiar para cada plataforma son diversas; cómo es la aplicación que se genera, qué tipo de interfaz presenta, qué herramientas posee para realizar el desarrollo, la calidad de la documentación, etc.

El desafío del trabajo presentado en esta Tesina de Máster consiste en afrontar la especificación e implementación de clientes adaptados a las características de los dispositivos móviles que accedan a los recursos formativos disponibles, a través de

entornos de desarrollo multiplataforma. De esta manera, los beneficios esperados del uso de estas plataformas se centran en la reducción del tiempo y del tiempo y coste de desarrollo, ya que se realiza una única implementación del producto, que posteriormente distribuye a cada una de las plataformas desarrolladas por el entorno.

1.2 Objetivos

Como hemos comentado en el apartado anterior, nos encontramos con la problemática de realizar el desarrollo de una aplicación que permita a los usuarios de MOBIP poder acceder a los distintos contenidos en nuevas tecnologías que estén disponibles. Esta aplicación debe estar disponible en las plataformas móviles más importantes, utilizando para ello alguno de los múltiples entornos de desarrollo móvil multiplataforma.

Los objetivos que se nos plantean, tanto de carácter de investigador como tecnológico, son los siguientes:

- Realizar un estudio comparativo entre los entornos multiplataforma existentes, analizando los puntos fuertes y débiles de cada uno. A partir de este estudio seleccionaremos el entorno multiplataforma adecuado para el desarrollo.

- Desarrollar clientes adaptados a las características de los distintos dispositivos móviles y que accedan a los recursos formativos de Sakai.

- Realizar un análisis del desarrollo, detallando como ha sido todo el proceso, describiendo los problemas encontrados y especificando si ha habido diferencias comparando el desarrollo para los distintos dispositivos. En base a este trabajo, crear un conjunto de recomendaciones o *best practises* que faciliten la toma de decisiones a las que se enfrentará un desarrollador novel en a la hora de enfrentarse a un desarrollo.

- Validar el desarrollo multiplataforma obtenido, para de esta manera comprobar que el entorno de desarrollo hace lo que se supone que debe hacer. Para ello, la aplicación generada debe de cumplir una serie de requisitos:

    - La funcionalidad obtenida es la especificada en los requisitos.
    - Adaptación a los distintos entornos-dispositivos: generación de interfaz nativa de cada plataforma, visualización de los componentes gráficos independientemente de la resolución del dispositivo, etc.
    - La usabilidad de la aplicación es buena en los distintos dispositivos.

1.3 Descripción del documento

Este documento se organiza de la siguiente manera. En este primer capítulo, se detallan los motivos que nos han llevado a la realización de esta Tesina y los objetivos

que se persiguen. En el capítulo 2 se describe el contexto, centrándose en el proyecto europeo MOBIP. En este capítulo también presentamos el sistema de *e-learning* Sakai y el estado del arte del acceso móvil a este tipo de sistemas. En el capítulo 3 se describe el estado del arte del desarrollo de aplicaciones móviles multiplataforma. En el capítulo 4 describimos la plataforma elegida para el desarrollo multiplataforma: Titanium. El capítulo 5 está centrado en la aplicación desarrollada: en qué consiste, su especificación y desarrollo. El enfoque del capítulo estará orientado a la problemática multiplataforma, comparando el desarrollo iOS-Android y comentando las limitaciones que se han podido encontrar usando Titanium. En el capítulo 6 presentamos los distintos aspectos analizados para validar la aplicación, especificando los criterios de evaluación y su importancia. Finalmente, en el capítulo 7 se muestran las conclusiones y trabajos futuros.

Capítulo 2

2. Contexto

2.1 Introducción

Definimos TIC (Tecnologías de la información y la comunicación) móviles como el subconjunto de las TIC que dan soporte a la movilidad. Hay numerosas definiciones del concepto de movilidad orientadas al marco de negocio y de sistemas de información. Hasta no hace mucho tiempo atrás, este concepto se reducía a la informática portátil (agendas personales, ordenadores portátiles y teléfonos móviles fundamentalmente). El concepto de movilidad en el ámbito de las TIC, hace referencia al conjunto de tecnologías que permiten el acceso a la información y a servicios desde dispositivos inalámbricos, en cualquier momento, y desde cualquier lugar [24].

La movilidad está basada en la combinación de una serie de tecnologías y en el desarrollo de unas pautas de comportamiento. Actualmente, se está constituyendo en uno de los temas cruciales que afectan cada vez más a sectores de todo tipo, pudiendo citar como ejemplos:

- Servicio y Soporte al cliente.
- Fuerza de Ventas.
- Aplicaciones en Marketing.
- Logística e Inventario.
- Automatización de la Cadena de Distribución.
- Automatización de servicios de campo.
- Las compras.
- Oficina móvil y herramientas corporativas.
- Aplicaciones para sectores industriales.
- Administración electrónica.

El mundo en el que vivimos ha cambiado debido a los recientes progresos en las tecnologías móviles y en sus servicios asociados, y estos cambios van a ir a más con el paso del tiempo. Los servicios móviles tienen un enorme potencial económico de cara a la competitividad, la innovación y el crecimiento.

En este capítulo profundizaremos en el proyecto europeo MOBIP, analizando su importancia respecto a los servicios móviles y el enorme potencial que poseen. También veremos en que consiste Sakai y su relación con el proyecto MOBIP.

2.2 MOBIP

El proyecto MOBIP [25] va dirigido a facilitar el acceso a inversores, explotar el potencial de la cooperación empresarial bilateral, así como el acceso a las grandes empresas del sector como fuente de capital y de acuerdos societarios. El proyecto proporcionará a los negocios basados en servicios móviles oportunidades para obtener contactos y nuevas oportunidades de negocio, además de acceso a fuentes financiación y nuevos mercados.

2.2.1 Actividades

Dentro de las distintas actividades llevadas a cabo dentro de MOBIP, podemos destacar las siguientes:

- En proyectos de gran crecimiento, evaluar las necesidades de investigación, formación, habilidades y financiación.
- Facilitar la transferencia de conocimiento entre los distintos miembros del sector mediante el desarrollo y testeo de herramientas.
- Apoyo a la innovación dentro de las iniciativas existentes.
- Promoción del uso de estándares.
- Organización de foros.
- Mejora de la disponibilidad de inversiones en las pequeñas y medianas empresas proporcionando acceso a financión y a los distintos mercados.
- Evaluación a posteriori de los nuevos servicios y herramientas.
- Divulgación de las actividades y resultados de los proyectos.
- Desarrollo de una estrategia de salida adecuada.
- Establecimiento de un acelerador europeo.

2.2.2 Objetivos

A modo de resumen, podemos decir que los objetivos del proyecto europeo MOBIP son:

- Apoyo a la competitividad de proyectos conjuntos de crecimiento exponencial en el sector de los servicios móviles y reforzar sus oportunidades de crecimiento y acceso al mercado tanto en Europa como en mercados internacionales más amplios.
- Fomentar la innovación en los servicios móviles.
- Desarrollar nuevas y mejores herramientas para el apoyo de la innovación mediante la creación de iniciativas fructíferas.

2.2.3 PYMEs

Uno de los objetivos fundamentales de MOBIP es la ayuda a las pequeñas y medianas empresas, proporcionando una gran cantidad de oportunidades a este tipo de empresas, ya sean oportunidades de financiación, oportunidades tecnológicas, oportunidades de negocio, etc.

Un aspecto fundamental para cualquier PYME es el acceso a fuentes de financiación para su negocio. Desde MOBIP se proporciona consejo y apoyo para conseguir este objetivo. El origen de estas fuentes de financiación puede ser diverso, pudiendo ser tanto nacional como internacional; fondos privados, como inversores ángeles o fondos de capital riesgo; fondos públicos, como subvenciones en I+D y subvenciones en innovación.

Otro aspecto importante que proporciona MOBIP a las PYMEs es el acceso a grandes eventos internacionales. La asistencia a estos eventos es una oportunidad de mostrar sus negocios a inversores de toda Europa, incluyendo inversores con intereses específicos en los servicios móviles. Esta es una de las oportunidades fundamentales que ofrece MOBIP, la posibilidad de que la empresa sea identificada como una compañía de servicios móviles a tener en cuenta.

Debido al carácter internacional de MOBIP, las empresas tienen acceso a nuevos mercados internacionales, lo que puede significar nuevas oportunidades de negocio. Además, el acceso a socios corporativos del sector móvil puede suponer oportunidades de inversión y nuevas posibilidades de colaboración.

La industria de los servicios móviles es una industria que evoluciona constantemente. MOBIP da la oportunidad a las empresas de entrar en contacto con actores claves a nivel global dentro de los servicios móviles; modelos a seguir y ponentes que están a la vanguardia de las últimas tendencias en la industria, así como los más recientes avances.

2.2.4 Instituto ITACA

ITACA [26] es un instituto de Investigación y desarrollo de la Universidad Politécnica de Valencia dentro del parque de institutos de la Ciudad Politécnica de la Innovación (CPI). El instituto ITACA tiene como misión la investigación aplicada en el campo de las Tecnologías de la Información y Comunicaciones TIC en un contexto nacional e internacional, con una utilidad directa de sus desarrollos hacia las empresas públicas y privadas. Con el desarrollo de innovaciones tecnológicas, el instituto ITACA pretende reforzar la competitividad y el desarrollo económico de la sociedad industrial, tanto local como europea, que establezca un beneficio para la sociedad en general.

Uno de los proyectos en los que participa el Instituto ITACA es MOBIP. Dentro de este proyecto europeo desarrolla dos funciones:

- **Animador tecnológico:** Debido a su condición de expertos, su función consistirá en asesorar a las empresas que lo soliciten sobre que tecnologías utilizar.
- **Formación en nuevas tecnologías:** Se encargarán de proveer contenidos en las distintas disciplinas de las tecnologías móviles. Para llevar a cabo esta tarea, Sakai es la solución *e-learning* elegida para cubrir las necesidades de formación *on-line* de MOBIP.

Para la implantación de Sakai como solución *e-learning* se cuenta con la colaboración de la empresa Samoo. Esta empresa se encarga de ofrecer servicios tecnológicos y asesoramiento a cualquier compañía u organización de cualquier sector en soluciones *e-learning* estando especializados en Sakai. Samoo es *commercial affiliate* de Sakai, siendo el único en España y uno de los dos que existen en países de habla hispana.

2.3 Proyecto Sakai

El Proyecto Sakai [22] desarrolla software educativo de código abierto que tiene su origen en la Universidad de Míchigan y en la Universidad de Indiana, a las que se unieron el MIT y la Universidad de Stanford, junto a la Iniciativa de Conocimiento Abierto (OKI) y el consorcio uPortal [21]. El Proyecto se consolidó con generosa ayuda de la Fundación Mellon.

Históricamente, las facultades y universidades han tenido que desarrollar sus propias aplicaciones software o confiar en un software propietario carente de interoperabilidad con otros sistemas y de flexibilidad para cubrir las necesidades únicas de cada institución. El proyecto Sakai busca desarrollar una colección integrada de Entornos de Aprendizaje Colaborativos (*Collaboration and Learning Environment*) de código abierto para la comunidad universitaria, incluyendo la gestión de cursos, valoración de herramientas y un sistema de colaboración de soporte a la investigación.

Según el proyecto Sakai, dos barreras técnicas importantes han frustrado los esfuerzos por reunir las inversiones en software educativo que podrían haber impulsado las economías de escala:

- Las arquitecturas técnicas locales han evitado la migración de software de unas instituciones a otras incluso cuando el software era gratuito y de código abierto.
- La falta de coordinación entre instituciones ha impedido reunir requisitos y recursos de manera sinérgica, desperdiciando los esfuerzos invertidos.

El proyecto Sakai tiene como objetivo crear un entorno de colaboración y aprendizaje para la educación superior y superar las barreras expuestas anteriormente. Para superar estas barreras, se establece un Perfil de Portabilidad Tecnológica (*Technology Portability Profile*) estándar que resume los elementos esenciales para la migración de código.

Sin embargo, el coste de poner operativo un sistema tan grande y conectarlo a otros sistemas heredados desarrollados por cada institución, puede ser un impedimento. Para gestionar el Proyecto se ha creado la Fundación Sakai, a la que pertenecen más de 100 Universidades. Destacan algunas de ellas por el número de cursos y usuarios:

- Indiana University.
- University of Michigan.
- Yale University.
- Stanford Univeristy.

- Oxford University.
- Cambridge University.

Sakai está realizado con tecnología Java lo que le proporciona una flexibilidad que permite su utilización en organizaciones con todo tipo de requisitos, desde pequeñas y medianas empresas a grandes universidades y administraciones.

El software Sakai se caracteriza por ofrecer una extensa lista de herramientas que pueden ser usadas como apoyo a la formación presencial, semipresencial y on-line pura, así como para crear espacios de trabajo colaborativo.

Sakai es también una comunidad para mejorar la enseñanza, el aprendizaje y la investigación. La comunidad trabaja en conjunto para definir las necesidades académicas de los usuarios, creando herramientas de software, compartiendo las mejores prácticas, conocimientos y recursos en apoyo de este objetivo. Cada día, los miembros de la comunidad comparten miles de interacciones, construyendo y mejorando el software, solicitando ayuda y colaborando en proyectos que surgen de este trabajo.

2.3.1.1 Sakai en Español

Cuando nació Sakai, en un primer momento se expandió por universidades americanas y de países anglosajones. En aquella época, la Universidad de Lleida decidió entrar en el proyecto, pero Sakai no estaba internacionalizado. Gracias al enorme esfuerzo de esta universidad y su continuación en la Universidad Politécnica de Valencia y algunas universidades Japonesas, Sakai pasó a ser un producto totalmente multilingüe lo que unido a otras características como su gran abanico de funcionalidades, su robusta arquitectura basada en Java, la capacidad de integración y comunicación con otros sistemas, el exhaustivo trabajo de control de calidad realizado en los últimos años y su activa comunidad, ha ampliado su ámbito de expansión a un nivel global.

En el área iberoamericana tenemos claros referentes que utilizan Sakai, como son las siguientes universidades:

- Universidad de Lleida.
- Universidad Politécnica de Valencia.
- Universidad Pública de Navarra.
- Universidad Complutense de Madrid.
- Universidad Virtual de Guadalajara.
- Universidad Nacional Autónoma de México

2.3.2 Acceso móvil actual

La proliferación de teléfonos de nueva generación, llamados comúnmente *smartphones*, ha revolucionado la manera en que los usuarios usan sus dispositivos. El uso de aplicaciones Web ha sido sustituido por el uso de aplicaciones nativas. Actualmente Sakai no dispone de aplicación específica en las plataformas móviles. El

acceso al sistema se realiza mediante el navegador Web, de la misma manera que lo haríamos desde un ordenador. Una vez nos hemos validado, debido al uso de vistas y distintos *templates*, el aspecto está optimizado para ser visualizado desde una pantalla de móvil.

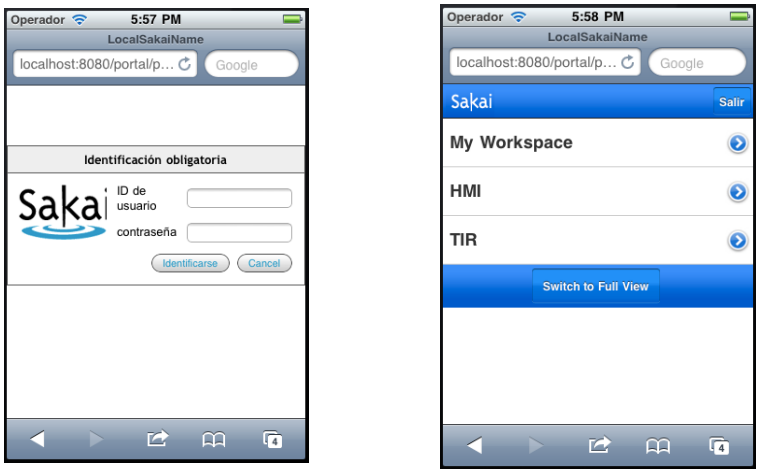


Figura 2.1: Acceso móvil Sakai

Conforme vamos navegando, la información se va adaptando al tamaño de la pantalla. En ocasiones la información no se muestra completamente y es necesario hacer Scholl horizontal. Esto hace incomodo y poco eficiente la consulta de información por parte de los usuarios móviles.

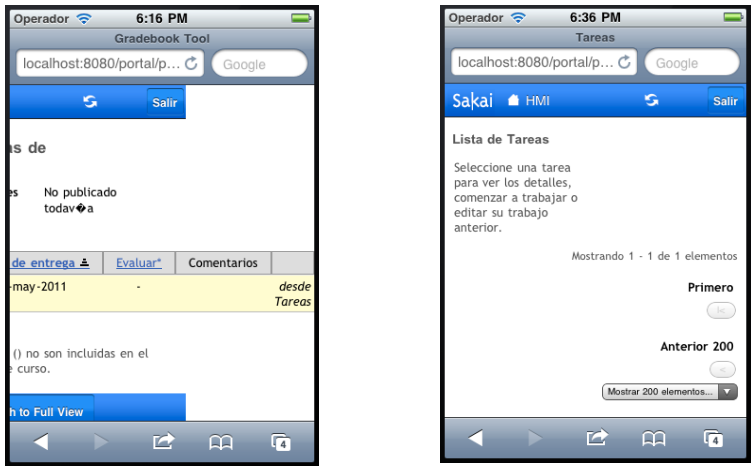


Figura 2.2: Información en acceso móvil Sakai

Debido a todo esto, el desarrollo de una aplicación móvil puede resultar muy útil para realizar determinadas acciones, como la consulta de anuncios, eventos de calendario, etc. Además del hecho que debido al auge de las aplicaciones móviles, el usuario es el que demanda este tipo de interacción.

2.3.3 Sakai CLE Mobile

La comunidad Sakai es una comunidad compuesta por una gran cantidad de desarrolladores que realizan determinados desarrollados que luego comparten con el

resto. Como parte del proyecto *Sakai Google Summer of Code*, miembros de la comunidad están desarrollando un proyecto para crear una aplicación multiplataforma para Sakai [23]. A continuación vamos a detallar algunas de las características de este proyecto, centrándonos en la plataforma utilizada, el modo en que obtiene la información de la instancia de Sakai y los prototipos de interfaz usados.

2.3.3.1 Plataforma

Para llevar a cabo el desarrollo han elegido PhoneGap [3] como plataforma. PhoneGap es una solución *open-source* para el desarrollo multiplataforma de aplicaciones móviles. Hace uso de HTML, CSS y JavaScript junto con librerías propias para el desarrollo multiplataforma de aplicaciones nativas.

2.3.3.2 Obtención de información

Un aspecto básico en una aplicación de este tipo es de qué manera se va a obtener la información. En este caso, los desarrolladores del proyecto han decidido hacer uso de la interfaz EntityBroker de Sakai y de sus fuentes de datos de tipo JSON. La recuperación de determinada información puede realizarse mediante las fuentes de datos ya existentes, pero es necesario el desarrollo de otros para que la aplicación pueda funcionar. Este trabajo se hará paralelamente al desarrollo de la aplicación

2.3.3.3 Prototipos de interfaz

Conforme va avanzando el desarrollo, distintos prototipos de interfaz se van desarrollando. En un primer momento, los desarrolladores mostraron bocetos de interfaz hechos en papel. Además de estos, también hicieron uso de algunas maquetas de interfaz, como las mostradas en la Figura 2.3.

Figura 2.3: Primeros prototipos Sakai CLE Mobile

2.4 Nuestro camino

Previamente a que la comunidad Sakai pusiera en marcha el proyecto Sakai CLE Mobile, pensamos que la realización de una aplicación multiplataforma específica para Sakai sería una buena manera de poner en práctica los conceptos tratados en esta Tesina. Nos encontramos con la problemática de realizar el desarrollo de una aplicación que permita a los usuarios de MOBIP podrán acceder a los distintos contenidos en nuevas tecnologías que estén disponibles. Esta aplicación debe estar disponible en las plataformas móviles más importantes, utilizando para ello alguno de los múltiples entornos de desarrollo móvil multiplataforma. En el próximo capítulo se realizará un estudio comparativo entre los distintos entornos multiplataforma para de esta manera encontrar el idóneo para el desarrollo.

2.5 Conclusiones

En este capítulo hemos visto el contexto en el que se va a desarrollar esta Tesina.

En primer lugar hemos hablado del concepto de movilidad y como este es un aspecto fundamental dentro del proyecto europeo MOBIP. Hemos destacado como uno de los objetivos fundamentales de MOBIP es la ayuda a las pequeñas y medianas empresas. Además hemos visto los principales objetivos y tareas que se llevan a cabo dentro de MOBIP.

Para dar soporte a la parte de formación de MOBIP, se utiliza la solución *e-learning* Sakai. Hemos analizado brevemente en qué consiste Sakai, resaltando cuáles son sus principales características y cuáles son los objetivos que persigue. Sakai permite su utilización en organizaciones con todo tipo de requisitos, desde pequeñas y medianas empresas a grandes universidades y administraciones.

A continuación hemos descrito el modo en que un usuario puede tener un acceso móvil en Sakai, mostrando los inconvenientes que representa y apuntando a la idoneidad del desarrollo de una aplicación nativa. Una vez visto el acceso móvil actual hemos repasado el proyecto de aplicación móvil desarrollado por un grupo de desarrolladores de la comunidad Sakai, especificando el tipo de entorno multiplataforma elegido además de mostrar algunos de los primero avances que se produjeron en el desarrollo.

Capítulo 3

3. Desarrollo móvil multiplataforma

3.1 Introducción

En los últimos tiempos el mercado de aplicaciones móviles ha explotado, convirtiéndose en unos de los mercados más dinámicos y lucrativos dentro del desarrollo software. Esto es consecuencia de la proliferación de los llamados *smartphones*, los teléfonos de nueva generación que han revolucionado la manera en que los usuarios se comunican entre sí, debido en gran medida a su conexión permanente a Internet. Además del cambio que se ha producido desde el punto de vista del usuario, también se ha producido una revolución desde el punto de vista del desarrollador. Esto es debido a la progresiva sustitución de desarrollo de aplicaciones Web por el desarrollo de aplicaciones nativas. Este cambio se justifica por las características intrínsecas de una aplicación nativa, como son su facilidad de uso y su mejor rendimiento sobre dispositivos móviles. Pero esta revolución no se habría producido sin un elemento esencial; la aparición de las tiendas de aplicaciones, que son las que permiten al usuario final el fácil acceso a éstas. La compañía que vio cual era el camino a seguir fue Apple, que con su AppStore consiguió un éxito sin precedentes que hizo que las demás plataformas sacaran su tienda propia. Actualmente, cada una de las distintas plataformas dispone de su propio sistema operativo y de su tienda de aplicaciones.

Una vez se ha extendido el uso de los llamados *smartphones*, es importante saber cuál es la situación actual del mercado. A este respecto, podemos decir que es un mercado bastante fragmentado, en el que no hay una plataforma que disfrute de una posición claramente dominante, a diferencia de lo que sucede en el mercado PC, monopolizado por Windows. Además, si atendemos a los datos proporcionados por las distintas consultoras del sector, esta fragmentación es distinta según la situación geográfica. Si nos centramos en el mercado estadounidense, según los datos recogidos por la consultora Nielsen en marzo del año 2011 [15], tres plataformas congregaban en torno al 86% del total de usuarios: Android, iOS y BlackBerry. Android con un 37% es la plataforma con mayor número de usuarios debido a la multitud de dispositivos de distintas marcas que lo llevan preinstalado. En segunda posición con un 27% se sitúa iOS, el sistema operativo de Apple, con un porcentaje muy elevado teniendo en cuenta que fabrican sus propios dispositivos y que solo éstos llevan su sistema operativo. El tercer integrante del trío dominador es BlackBerry con un 22%, plataforma que basa su fortaleza en los usuarios del mercado profesional. Las tres son las grandes dominadoras

del mercado norteamericano actual, siendo Android y iOS las que previsiblemente irán aumentando su porcentaje hasta dominar el mercado. El resto de plataformas tienen un porcentaje más residual.

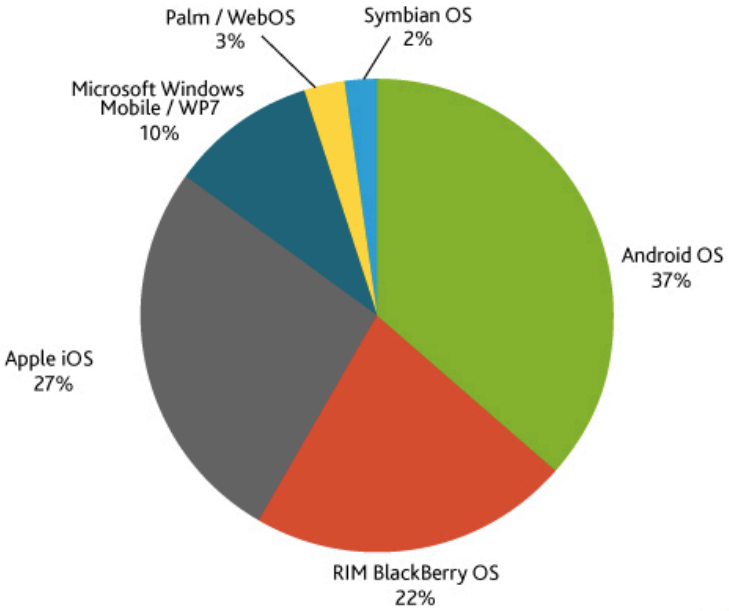


Figura 3.1: Mercado USA de *smartphones* (Nielsen, Marzo 2011)

Si en lugar de centrarnos en el mercado de Estados Unidos, nos fijamos en los datos a nivel mundial nos encontramos con algunas diferencias. Según los datos recogidos por la consultora Canalys en el último cuarto del año 2010 [16], cuatro plataformas congregaban el 94% de los usuarios de *smartphones*. El principal aspecto a destacar es que además de las tres plataformas dominadoras del mercado americano, existe una cuarta con un gran porcentaje de usuarios: Symbian. La información de cómo está distribuido el mercado es muy útil, ya que para el desarrollador puede ser un factor decisorio importante de cara al desarrollo de aplicaciones, ya que el objetivo siempre es llegar al mayor número potencial de usuarios.

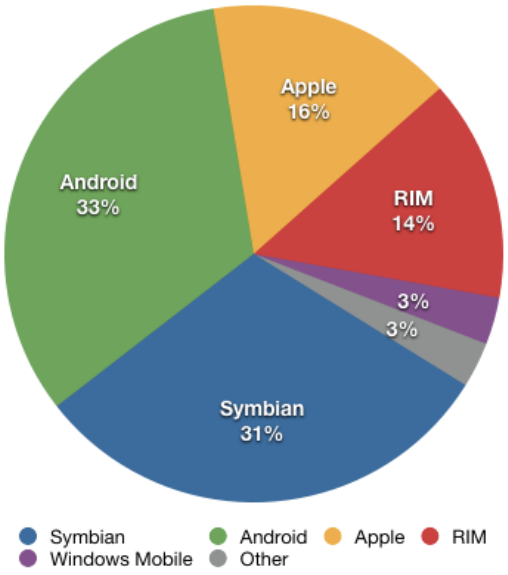


Figura 3.2: Mercado mundial de *smartphones* (Canalys, Q4 2010)

Como consecuencia de la aparición y proliferación de los distintos sistemas operativos y sus respectivas tiendas de aplicaciones, el desarrollador se enfrenta al problema de la fragmentación. Podemos definir la fragmentación [1] como la imposibilidad de poder desarrollar una aplicación y que esta se pueda ejecutar en distintos dispositivos (*write once and run anywhere*). Esto provoca que los desarrolladores de aplicaciones móviles se tengan que enfrentar a multitud de desafíos, que complican en gran medida el desarrollo. Para ilustrar esto, podemos comentar el hecho de que cada plataforma tiene sus herramientas y APIs especificas, o que cada plataforma hace uso de distintos lenguajes de programación. Esto provoca que el esfuerzo requerido para desarrollar una aplicación en una determinada plataforma sea muy difícil de medir hasta que no se ha desarrollado una. El desarrollador tiene que hacer frente al arduo proceso de hacerse desarrollador de una plataforma, teniendo en cuenta que cada una tiene sus procesos y requerimientos distintos. Además, la documentación existente no es muy clara, lo que complica aún más el proceso. Todo esto, unido a que cada plataforma tiene además su propia tienda para distribuir las aplicaciones, hace que el desarrollador se plantee buscar alternativas que faciliten su labor, y así es como surge la necesidad del desarrollo multiplataforma de aplicaciones móviles.

Los entornos de desarrollo multiplataforma han surgido como una necesidad. A grandes rasgos, podemos decir que existen dos grandes categorías dentro de estos entornos multiplataforma. Por un lado tenemos aquellos que mediante el uso de determinadas APIs, permiten crear una aplicación nativa para cada uno de los dispositivos definidos en el entorno. Por otro lado tenemos aquellos que permiten que una aplicación Web sea empaquetada como una aplicación nativa, la cual se ejecutará sobre el motor del navegador Web del dispositivo móvil.

En este capítulo vamos a tratar la fragmentación dentro del desarrollo de aplicaciones móviles. Además, realizaremos un estudio comparativo de los distintos entornos multiplataforma existentes.

3.2 Fragmentación

Como hemos comentado, la fragmentación es la imposibilidad de “write once and run anywhere”. Su incidencia en el desarrollo de aplicaciones móviles ha hecho que se convierta en un fuerte tema de debate. La imposibilidad de poder desarrollar una aplicación, y que esta se pueda ejecutar en distintos dispositivos del mercado, es un problema al que se tienen que enfrentar los desarrolladores.

3.2.1 Causas de la fragmentación

La fragmentación dentro del desarrollo de aplicaciones para dispositivos móviles puede venir causada por una serie aspectos. A continuación vamos a ver ejemplos que describen cada uno de estos aspectos.

3.2.1.1 Software

Una de las principales causas de la fragmentación viene determinada por el *software*. Como hemos comentado en la introducción, existen distintos sistemas operativos para cada una de las plataformas que conforman el mercado; iOS Android, BlackBerry, Windows Mobile, etc. Esto implica también el uso de distintos lenguajes de programación, así como el uso de APIs para acceder a los elementos hardware de los dispositivos. Además, el soporte multimedia de cada dispositivo también puede ser diverso, debido al uso de distintos codecs. Otro aspecto diferente puede ser el acceso al almacenamiento interno del dispositivo. Todo esto hace que la interacción entre el usuario y la aplicación sea muy diversa.

3.2.1.2 Hardware

Otro aspecto fundamental son las diferencias de hardware que existen entre los distintos dispositivos. Existen multitud de configuraciones distintas de hardware que dificultan el desarrollo de una aplicación para múltiples plataformas. Como ejemplo, podemos citar la pantalla del dispositivo. Las pantallas tienen una serie de características que determinan la interacción con el usuario, como es la resolución de pantalla, si ésta es táctil o no, si es panorámica, etc. Además de la pantalla, hay otros elementos hardware que propician la fragmentación, como el procesador del que dispone el dispositivo, la cantidad de memoria del dispositivo, el tamaño del almacenamiento interno, la existencia de cámara, etc.

3.2.1.3 Versionado

La estrategia comercial del desarrollador puede provocar que la hora de desarrollar y distribuir una aplicación, no haya una sola versión de ésta. Muchas aplicaciones se distribuyen en distintas versiones. Por ejemplo, una versión Lite con determinadas funciones y una versión PRO con toda la funcionalidad. Esto es un elemento más que provoca fragmentación.

3.2.1.4 Entorno

De la misma manera que existen diferentes sistemas operativos en las distintas plataformas, también existen sus correspondientes tiendas de aplicaciones. La importancia de estas tiendas es fundamental, ya que son las encargadas de hacer llegar al usuario final la aplicación desarrollada. Las distintas plataformas tienen sus propias tiendas de aplicaciones, con sus propias normas y procedimientos, lo que complica enormemente el proceso de desarrollo.

3.2.1.5 Tipo de usuario

El tipo de usuario final es un aspecto que también provoca fragmentación. Cuando desarrollamos una aplicación debemos tener en cuenta tanto el idioma del usuario, como a qué tipo de mercado se va a implantar. Otro aspecto importante es el hecho de que la aplicación pueda ser usada por personas con algún tipo de discapacidad.

En este caso hay una serie de estándares que hay que cumplir para que nuestra aplicación pueda ser usada por este tipo de usuario.

3.2.2 Importancia de la fragmentación

Como hemos visto en el capitulo anterior, la fragmentación viene provocada por multitud de causas. De manera un poco más formal, podemos decir que la fragmentación es la imposibilidad de desarrollar una aplicación sobre un determinado contexto operativo (CO) y conseguir que la aplicación también funcione en el resto de COs en los que podría funcionar. Podemos definir el CO como todo lo que rodea e influye en el funcionamiento de la aplicación.

La fragmentación afecta a todos los actores relacionados con las aplicaciones móviles, ya sean los usuarios finales que van a utilizar la aplicación, los desarrolladores de la aplicación, los proveedores de contenido, las operadores móviles, etc. Pero sobretodo, la fragmentación tiene una gran importancia en el desarrollo de una aplicación móvil, ya que afecta a cada una de sus etapas. A continuación vamos a ver en que medida dificulta cada una de las fases de desarrollo de una aplicación.

3.2.2.1 Modelado de negocio

La fase de modelado es donde los analistas de negocio deben preguntarse cuál es el conjunto de COs apropiados para la aplicación. Este punto es crucial, porque determinará el esfuerzo necesario para el desarrollo. Por ejemplo, se debe decidir si la aplicación a desarrollar es adecuada para el dispositivo X, o si vale la pena adecuar la aplicación para que también esté disponible en el dispositivo Y, etc. Para decidir es importante saber, entre otros aspectos, el tamaño de los mercados de cada dispositivo, para poder desechar aquellos que no sean potencialmente rentables.

3.2.2.2 Gestión de requisitos

En la fase de gestión de requisitos es donde se especificaran los casos de uso de la aplicación a desarrollar. Esta tarea se verá dificultada en gran medida por la fragmentación, ya que la manera en que el usuario interacciona con la aplicación difiere en función del CO. Esto provocará diferentes comportamientos que tendrán que ser especificados en los casos de usos, por lo que será necesario un gran conocimiento de todos los COs por parte de los analistas.

3.2.2.3 Análisis y diseño

A la hora de realizar el análisis y diseño de la aplicación, la existencia de distintos COs en los que la aplicación debe funcionar complica en gran medida esta fase. La creación de la estructura del sistema y la especificación del diseño, deben hacerse de manera que el desarrollo para todos estos COs sea viable. Además, se deben prever futuros COs a los que aplicación estará expuesta durante su existencia.

3.2.2.4 Implementación

La fase de implementación también se ve dificultada por la fragmentación. Por un lado, los programadores tienen que determinar la manera en que se desarrolla una aplicación para un determinado CO: el lenguaje de programación, las herramientas necesarias, la documentación, etc. Además, es importante determinar las diferencias entre los distintos CO, actuando en consecuencia para cada uno de ellos. Debido a todo esto, un aspecto que puede disminuir la problemática es determinar si con una versión podemos llegar a distintos COs, o es necesario un desarrollo por cada CO.

3.2.2.5 Testeo

La fase de testeo vendrá determinada por el número total de COs a los que queramos llegar. Un número elevado puede hacer larga y costosa esta fase dentro del desarrollo de nuestra aplicación, ya que es necesario hacer intensivas pruebas en cada una de ellos. Las pruebas se deberán realizar en primer lugar en los emuladores de cada dispositivo. Esto no será suficiente, y las pruebas en dispositivos reales serán fundamentales, ya que el comportamiento de las aplicaciones es muy distinto al ejecutarse en un simulador o en un dispositivo real.

3.2.2.6 Gestión de proyecto

Desde el punto de vista de la gestión de un proyecto, los problemas derivados de la fragmentación crearán más de un contratiempo. Las personas encargadas de la gestión del proyecto determinan los tiempos de entrega, la cantidad de recursos necesarios para el desarrollo, etc. Todo esto puede verse afectado por la aparición de un nuevo CO en el que la aplicación debe funcionar, o por cambios en un CO ya previsto.

3.2.2.7 Gestión de configuraciones y cambios

La fragmentación puede afectar el mantenimiento del código y la gestión de cambios de éste. El hecho de que la aparición de nuevos dispositivos sea continua y de que el software de cada plataforma evolucione constantemente provocaran cambios continuos en el código.

3.2.2.8 Entorno

Afrontar los distintos problemas derivados de la fragmentación no es fácil. En muchos casos las personas implicadas en el desarrollo intentarán resolver estos problemas con las herramientas que tengan disponibles en ese momento. Esto muchas veces no será suficiente, y será necesaria la utilización de nuevas herramientas que nos ayuden a resolver los distintos problemas que nos vayamos encontrando.

3.3 Frameworks multiplataforma

Como hemos comentado en la introducción de este apartado, para hacer frente a la fragmentación han surgido en los últimos años multitud de Frameworks multiplataforma de desarrollo de aplicaciones móviles. Como muestra de este esfuerzo, a continuación se listan algunas de estas plataformas:

- **PhoneGap** [3]: Uso de HTML, CSS y JavaScript junto con librerías propias para el desarrollo multiplataforma de aplicaciones nativas. Soporta las siguientes plataformas: iOS, Android, BlackBerry, Symbian, Windows Phone 7.
- **Rhomobile** [4]: Hace uso de Ruby para la lógica de negocio y utiliza HTML, CSS y JavaScript para la interfaz. Uso de Rhosync para la sincronización de datos cliente-servidor. Soporta las siguientes plataformas: iOS, Android, BlackBerry, Symbian, Windows Phone 7.
- **Titanium** [5]: Uso de JavaScript junto a librerías propias para el diseño de aplicaciones nativas. Es un Framework de código abierto. Soporta las siguientes plataformas: iOS, Android, BlackBerry (fase Beta).
- **Mosync** [6]: Hace uso de código C/C++ para el desarrollo multiplataforma de aplicaciones móviles. Es un Framework propietario. Soporta las siguientes plataformas: iOS, Android, Windows Mobile, Symbian, JavaME y Moblin.
- **Bedrock** [7]: Compilador multiplataforma que convierte el código fuente J2ME a código C++, que genera simultáneamente la aplicación a distintas plataformas móviles. Las plataformas soportadas son: iOS (iPhone), Android, BREW, Windows Mobile, BlackBerry, Bada.
- **Alchemo** [8]: Plataforma que automatiza la conversión de aplicaciones J2ME a distintas plataformas móviles: iOS (iPhone), Android, BREW, Windows Mobile.
- **JMango** [9]: Hace uso de su propio lenguaje: JMango Script. Mediante este lenguaje accedemos a las funcionalidades del teléfono. Soporta las siguientes plataformas: Java ME, Android, Bada, BlackBerry, iPhone, Windows Mobile 6, Windows Phone 7.
- **Marmalade** [10]: Hace uso de código C/C++ para el desarrollo multiplataforma de aplicaciones móviles. Permite el uso de librerías C++ ya existentes. Utilizado para la conversión de videojuegos a plataformas móviles. Soporta las siguientes plataformas: iOS, Android, Bada, Symbian, WebOS.
- **QuickConnectFamily** [11]: Uso de HTML, CSS y JavaScript para generar aplicaciones nativas para iOS, Android, BlackBerry, WebOS. Hace uso de plantillas para acceder a comportamientos nativos.

Debido al elevado número de entornos multiplataforma, se hacía necesario elegir un número reducido de ellos para poder profundizar más en cada uno. Los elegidos fueron:

- PhoneGap.
- Rhomobile.
- Titanium.

Las razones para elección de estas tres plataformas son diversas. Un aspecto fundamental es el hecho de que aparentemente son las plataformas con mayor número

de desarrolladores. Dentro del conjunto de entornos multiplataforma que han aparecido en el mercado, estas tres plataformas son las más comentadas y mejor analizadas en las distintas referencias existentes, además de que poseen una gran número de aplicaciones en la App Store y el Android Market. Como ejemplo de la importancia de estas tres plataformas podemos citar el libro *Pro Smartphone Cross-Platform Development* [2], el cual se centra en PhoneGap, Rhomobile y Titanium.

Por otro lado, estas tres plataformas también han recibido distintos premios. Titanium recibió el premio *2010 Jolt Productivity Award de Dr Dobbs*, encargado de premiar a productos que han sacudido el mercado y que se caracterizan por hacer de la creación de software algo más rápido, sencillo y eficiente. Rhomobile obtuvo el premio *Best Startup Company* en la edición *2009 Best of Interop Awards*, por ser considerada la mejor entre las nuevas compañias que se presentaron a la convención. Por su parte, PhoneGap ganó en el *Web 2.0 Expo LaunchPad* en Abril de 2009, que también premia a la mejor entre las nuevas compañias.

3.3.1 Aspectos a analizar

Para cada una de las plataformas, nos vamos a centrar en una serie de aspectos generales de cada una de ellas.

- **Aplicación:** En este apartado se explica que tipo de aplicación genera la plataforma, las tecnologías en las que se basa y el tipo de aplicación a desarrollar más indicada.
- **Interfaz y APIs:** En este apartado se especifica si la interfaz generada por la plataforma es nativa o no. Este aspecto es muy importante de cara al usuario final, ya que un aspecto no nativo puede producir rechazo. Además de esto, se describen las distintas APIs soportadas, haciendo hincapié en las capacidades del teléfono que son accesibles por cada una de las plataformas (geolocalización, cámara, etc.).
- **Desarrollo:** Las herramientas de las que dispone el usuario para realizar el desarrollo de las aplicaciones es un factor fundamental para el desarrollador. En este apartado se describirán las existentes para cada plataforma, especificando si disponen de un IDE propio o hacen uso de otros ya existentes.
- **Documentación y comunidad:** Aquí se describen dos aspectos claves para el éxito de una plataforma, como son la documentación existente y la comunidad de desarrolladores que la usa. En los primeros pasos como desarrollador, un aspecto muy importante es la existencia de buenas guías y tutoriales, y también de ejemplos disponibles de aplicaciones desarrolladas.
- **Generación:** En este apartado se describe la manera en que se realiza la generación de la aplicación final, nombrando las herramientas necesarias y el proceso a seguir.

3.3.2 PhoneGap

PhoneGap es una solución open-source para el desarrollo multiplataforma de aplicaciones móviles. Como hemos comentado en la introducción del capítulo, existen dos grandes grupos de Frameworks multiplataforma: los que generan aplicaciones que

se ejecutaran sobre el motor del navegador Web, y los que generan aplicaciones nativas. Phonegap pertenece al primer grupo. Está basado en HTML5, y principalmente hace uso de las tecnologías Web más usadas por los desarrolladores: HTML y JavaScript. Soporta las siguientes plataformas: iOS, Android, BlackBerry, Symbian, Windows Phone 7.

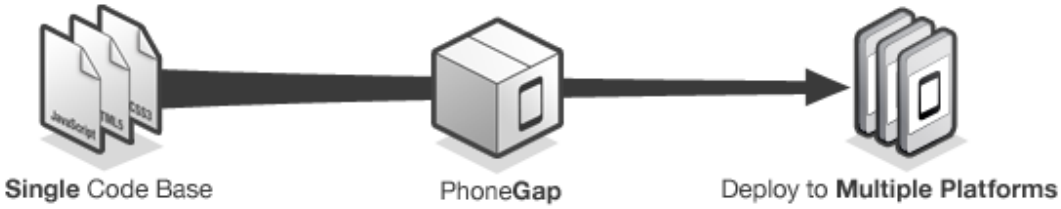


Figura 3.3 Plataforma PhoneGap

3.3.2.1 Características generales

En este apartado vamos a comentar los aspectos característicos de PhoneGap como entorno de desarrollo multiplataforma.

Aplicación

Una aplicación PhoneGap es en realidad una aplicación Web. Como tal, todo el código está escrito en HTML/CSS/JavaScript. PhoneGap no exige que la aplicación tenga una cierta estructura, ni propone una guía de desarrollo. De hecho, se puede convertir una aplicación Web en una aplicación PhoneGap. Para realizar la conversión a aplicación instalable en un dispositivo, PhoneGap dispone de un conjunto de APIs.

En cuanto al tipo de aplicaciones a desarrollar, PhoneGap no está pensado para aplicaciones que hagan uso de animaciones 3D, ni que requieran de potentes cálculos matemáticos. PhoneGap tampoco proporciona soporta a base de datos, dejando esto en manos de las APIs de persistencia de HTML5. El tipo de aplicación a desarrollar seria la típica aplicación Web.

Interfaz y APIs

El diseño de la interfaz dentro de PhoneGap se realiza mediante HTML/CSS/JS. Debido a que una aplicación en PhoneGap es básicamente una aplicación Web corriendo sobre el navegador del móvil, cualquier código Web, ya sea HTML o JavaScript, puede ser integrado en la aplicación. Esto es una gran ventaja para los desarrolladores Web que quieren utilizar los conocimientos ya adquiridos.

Para poder acceder a las funcionalidades propias del dispositivo, PhoneGap dispone de una conjunta de APIs desarrolladas por ellos. Estas funcionalidades son el acceso a contactos, geolocalización, acelerómetro, cámara, etc. Debido a que PhoneGap soporta varios dispositivos, puede que alguna de estas funcionalidades no esté disponible para alguno de ellos.

Desarrollo

El entorno de desarrollo está abierto a la elección del desarrollador. La parte positiva de esto es que los desarrolladores Web pueden usar las herramientas a las que ya están acostumbrados, pero es un problema para aquellos que carecen de esa experiencia y quieren dar los primeros pasos en la plataforma. Según el proyecto, el desarrollador tiene que decidir que librerías necesita, que IDE es el más apropiado, o que Framework táctil elegir. También se puede utilizar la librería XUIJS, desarrollada por los creadores de PhoneGap.

Documentación y comunidad

Respecto a la documentación existente, PhoneGap dispone de una gran cantidad de guías y tutoriales. Dentro de la documentación que proporciona PhoneGap en su Web, destaca la sección *Get started*, cuya importancia es fundamental debido a que es la puerta de entrada de los programadores en la plataforma. Incluye los pasos a realizar para configurar el ordenador para cada una de las plataformas, además de los requerimientos necesarios, herramientas disponibles para el desarrollo o las instrucciones a seguir. También hay documentación específica de las distintas APIs propias de PhoneGap.

El contacto con la comunidad también es un aspecto importante en PhoneGap. Entre otros elementos, existe un grupo de Google, además de canales en IRC y una extensa Wiki.

Generación

Una vez desarrollada la aplicación, podremos generar nuestro ejecutable especifico para cada plataforma. Para ello se hace uso de la herramienta PhoneGap Build. El primer paso es subir el proyecto PhoneGap al servicio PhoneGap Build. Una vez hecho esto, se configuran para cada plataforma sus respectivos certificados, claves, etc. Por último, generamos la aplicación, que ya estará preparada para ser distribuida. Es importante resaltar que para generar el código para la plataforma iOS es necesario disponer un Mac con las herramientas de desarrollo instaladas.

3.3.2.2 Puntos fuertes/débiles

A modo de resumen, a continuación se muestran los puntos fuertes y los puntos débiles de PhoneGap como Framework de desarrollo multiplataforma de aplicaciones móviles:

Puntos fuertes

- Libre (Open-source)
- A partir de un determinado código genera una aplicación en múltiples plataformas.

- Soporta las principales plataformas móviles (iOS, Android, BlackBerry, Symbian, Windows Phone 7).
- Existencia de librerías que acceden al hardware del teléfono (cámara, GPS, acelerómetro…).
- Uso de las principales tecnologías Web (gran ventaja para los desarrolladores Web).
- Generación de aplicaciones desde el navegador con *PhoneGap Build.*

Puntos débiles

- Aspecto no "nativo".
- La aplicación se ejecuta sobre el navegador del dispositivo, lo que repercute negativamente en el rendimiento de la aplicación.
- No hay soporte específico para bases de datos, se usan las APIs de HTML5.

3.3.3 Rhomobile

Rhomobile es otra solución para el desarrollo multiplataforma de aplicaciones móviles. A diferencia de PhoneGap, Rhomobile pertenece al grupo de Frameworks cuya finalidad es crear una aplicación nativa. Consta de un conjunto de productos, siendo Rhodes el producto principal. De manera general podemos decir que Rhodes es un Framework open-source que genera código nativo para la mayoría de plataformas: iOS, Android, BlackBerry, Nokia y Windows Mobile. En la siguiente figura se muestra la estructura de Rhomobile:

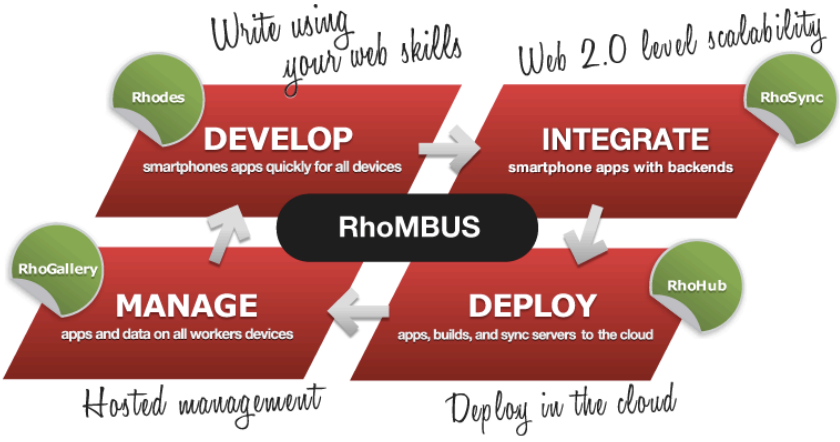


Figura 3.4 Plataforma Rhomobile

3.3.3.1 Características generales

Rhomobile consta de un conjunto de productos mediante los cuales gestionar todo lo relacionado con el desarrollo de aplicaciones (Rhodes, RhoHub, RhoSync, RhoGallery…). Rhodes es el Framework MVC (Model-View-Controller) mediante el cual se desarrollan y generan las aplicaciones. A continuación vamos a comentar los aspectos característicos de Rhomobile como entorno de desarrollo multiplataforma.

Aplicación

A pesar de que las aplicaciones desarrolladas mediante Rhodes se instalan y ejecutan como aplicaciones nativas, el proceso es muy similar al seguido para el desarrollo Web, siguiendo el estándar MVC (Model-View-Controller):

- *View*: Para la definición de las vistas se hará uso de HTML y CSS, pudiendo utilizar JavaScript de la misma manera que haríamos en el desarrollo Web. Este código Web se entremezcla con código Ruby dentro de ficheros ERB (Embedded Ruby), para de esta manera poder añadir la lógica de negocio a las vistas. En un fichero ERB se entremezclan partes en código Ruby con partes HTML de una manera similar a como funciona ASP/PHP.
- *Controller*: Para la definición de controladores se hace uso de scripts en código Ruby, mediante ficheros "controller.rb".
- *Model:* Para la definición del modelo se hace uso de scripts en código Ruby, mediante ficheros "model.rb".

Si la aplicación a desarrollar necesita sincronizarse con determinados servidores, se puede hacer uso de uno de los productos que conforman Rhomobile: RhoSync. De manera genérica se puede definir RhoSync como un servidor que actúa como un nivel intermedio entre una aplicación móvil y el servicio Web que accede para recuperar información. Está escrito en Ruby. El problema de RhoSync es que se requiere suscripción de pago para su uso.

Respecto al acceso local de datos, Rhodes está diseñado para funcionar con una base de datos local de tipo SQLite o HSQLDB, lo que permite trabajar con datos sincronizados.

Interfaz y APIs

La interfaz gráfica de una aplicación Rhomobile no tiene un aspecto enteramente nativo. En muchos casos la apariencia es similar, pero las diferencias son apreciables. El acceso a las distintas funcionalidades nativas del teléfono se realiza mediante una serie de APIs desarrolladas para Rhodes. De esta manera accedemos a la agenda del teléfono, al GPS, a la cámara, etc. Dependerá de la plataforma de destino de la aplicación el acceso a las determinadas funcionalidades. El código que hay que escribir en nuestra aplicación para hacer uso de estas APIs es independiente de la plataforma destino de la aplicación.

Desarrollo

Para el desarrollo de aplicaciones, Rhodes no incluye su propio IDE de desarrollo. El código de la aplicación se puede escribir con cualquier editor que soporte HTML y Ruby. RhoStudio es un plugin de Rhodes para Eclipse, que se encuentra en fase Beta.

Mientras se desarrolla la aplicación, ésta se puede probar tanto en el emulador como en el dispositivo correspondiente. Para ello debemos de tener instalados los emuladores, siguiendo las instrucciones de instalación de cada plataforma.

Documentación y comunidad

En la página oficial existen una gran cantidad de tutoriales y guías que permiten conocer los principales aspectos necesarios para poder desarrollar aplicaciones. Se echa en falta un mayor número de ejemplos, lo que nos permitiría ver de una manera más clara de lo que es capaz el Framework y cuál es la manera de realizarlo.

Respecto a la comunidad de desarrolladores, existe un grupo en Google Groups, donde los desarrolladores pueden interactuar con otros miembros para resolver dudas, compartir experiencias, etc. En este aspecto, Rhomobile no tiene detrás una comunidad tan potente como otras plataformas.

Generación

Una vez desarrollada la aplicación, a partir del código Ruby se genera el código nativo de la plataforma que haya sido seleccionada. Esto se hace mediante RhoHub, que es la parte de Rhomobile encargada de la generación de las aplicaciones. Desde RhoHub se gestionan los siguientes aspectos:

- Las distintas versiones de nuestra aplicación.
- El repositorio Git con el código de nuestra aplicación.
- Los ejecutables a generar, especificando el dispositivo y su respectiva versión de firmware.
- Colaboradores en el desarrollo de la aplicación.

Todo el código generado no es nativo. La aplicación es interpretada en parte por el *RubyVM interpreter*, lo que puede afectar al rendimiento de la aplicación. Es importante resaltar que para generar el código para la plataforma iOS es necesario disponer un Mac con las herramientas de desarrollo instaladas.

3.3.3.2 Puntos fuertes/débiles

A modo de resumen, a continuación se muestran los puntos fuertes y los puntos débiles de Rhomobile como Framework de desarrollo multiplataforma de aplicaciones móviles:

Puntos fuertes

- Libre (*Open-source*)
- A partir de un determinado código genera una aplicación en múltiples plataformas.
- Compila una aplicación nativa.

- Soporta las principales plataformas móviles (iOS, Android, BlackBerry, Symbian, Windows Phone 7).
- Permite trabajar con emuladores y con el teléfono.
- Librerías que acceden al hardware del teléfono (cámara, GPS, acelerómetro…).
- Uso Ruby como lenguaje principal (punto fuerte si sabes Ruby).
- Generación de aplicaciones desde el navegador con RhoHub.
- Fácil sincronización de datos entre dispositivos y en la nube, mediante RhoSync.

Puntos débiles

- Todo el código que genera no es nativo. La aplicación es interpretada en parte por el *RubyVM interpreter*.
- Pocos ejemplos en comparación con Titanium.
- Aspecto no tan "nativo".
- Uso Ruby como lenguaje principal (punto débil si no sabes Ruby).
- Hay que pagar para determinados usos de RhoSync.

3.3.4 Titanium

La tercera alternativa a exponer es Titanium. Al igual que Rhomobile, Titanium también pertenece al grupo de Frameworks cuya finalidad es crear una aplicación nativa. Titanium es una plataforma open-source para el desarrollo multiplataforma de aplicaciones móviles basado en tecnologías Web. Permite crear, ejecutar y empaquetar aplicaciones nativas para iOS, Android y BlackBerry (en fase Beta), mediante el uso de APIs propias.

3.3.4.1 Características generales

En este apartado vamos a comentar los aspectos fundamentales de Titanium como Framework de desarrollo multiplataforma de aplicaciones móviles.

Aplicación

De forma genérica, se puede decir que una aplicación de Titanium consta de 4 partes, como se muestra en la Figura 3.5:

1. El código HTML/CSS/JavaScript que define la lógica y la interfaz de la aplicación.
2. Las APIs que acceden a la funcional nativa del dispositivo.
3. El puente que se encarga de convertir el código Web en código nativo de la plataforma especificada.
4. El conjunto de instrucciones en tiempo de ejecución que empaquetan la aplicación para ser distribuida.

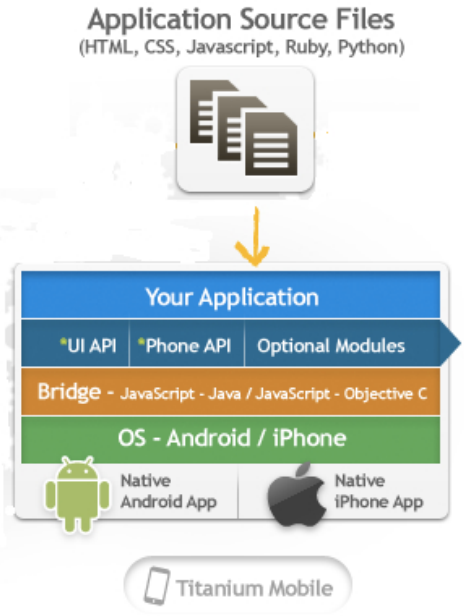


Figura 3.5 Plataforma Titanium

El lenguaje fundamental dentro de una aplicación Titanium es JavaScript. Las aplicaciones hechas con Titanium se ejecutan sobre un motor JavaScript que invoca a las APIs nativas.

Interfaz y APIs

Para diseñar la interfaz de una aplicación, Titanium dispone de una API basada en JavaScript que da acceso a componentes nativos de interfaz. Gracias a este API, el aspecto de una aplicación Titanium es igual que el de una aplicación diseñada con un procedimiento estándar para una determinada plataforma. Este es uno de los aspectos fundamentales del Framework, y uno de sus principales puntos fuertes. También se pueden desarrollar interfaces mediante HTML y CSS, pero no es lo habitual.

Respecto a acceso a las distintas funcionalidades del teléfono, Titanium dispone de una serie de APIs que dan acceso al GPS, el sistema de ficheros, la cámara, reproducción de video, etc. De la misma manera que para la interfaz, el acceso a estas APIs se hace mediante código JavaScript.

Desarrollo

Titanium consta de un potente IDE para el desarrollo de aplicaciones, llamado Titanium Studio. Este entorno de desarrollo surge de la compra de Aptana por parte de Titanium en Enero de 2011. Aptana es uno de los entornos de desarrollo Web más usados en el mundo. Hasta la aparición de Titanium Studio, el desarrollo se realizaba sobre cualquier editor con soporte para JavaScript, y después mediante Titanium Developer se generaba el código final de nuestra aplicación. Titanium Developer es una herramienta que permite crear, ejecutar y generar aplicaciones Titanium. De la fusión entre Aptana y Titanium Developer surge Titanium Studio.

Para la depuración de código, Titanium Studio incorpora un *debugger,* pero esta opción solo está disponible para los usuarios de pago. Para el resto de usuarios el método de depuración es mediante la escritura de trazas en la consola del IDE.

Documentación y comunidad

Como hemos comentado con anterioridad, uno de los aspectos fundamentales para una Framework de desarrollo es la documentación existente. Titanium pone a disposición del desarrollador multitud de guías y tutoriales que cubren todo lo necesario para conocer a fondo la plataforma. También hay una gran cantidad de ejemplos, cuyo mayor exponente es el llamado *Kitchen Sink*. Esta aplicación de ejemplo hace uso de todas las APIs disponibles de Titanium. De esta manera, podemos ver cuál es el modo de usar cada una de ellas. Un aspecto negativo de la gran cantidad de documentación existente, es que a veces es complicado encontrar exactamente lo que estás buscando.

Otro aspecto a destacar es la numerosa comunidad desarrolladores existentes usando Titanium. Dentro de la parte dedicada a los desarrolladores en la página oficial, la sección Q&A (*Questions & Answers*), permite poner en común dudas que serán contestadas por miembros de Titanium o por otros miembros de la comunidad.

Generación

Una vez desarrollada, la generación de la aplicación se realiza a través de Titanium Studio. Además de entorno de desarrollo, Titanium Studio incorpora las herramientas de generación incluidas en su predecesora Titanium Developer, además de la posibilidad de ejecutar el simulador.

Es importante resaltar que para generar el código para la plataforma iOS es necesario disponer un Mac con las herramientas de desarrollo instaladas, además de seguir los pasos necesarios: darse de alta como desarrollador, obtener licencia, obtener certificados, firmar fuentes, etc.

3.3.4.2 Puntos fuertes/débiles

A modo de resumen, a continuación se muestran los puntos fuertes y los puntos débiles de Titanium como Framework de desarrollo multiplataforma de aplicaciones móviles:

Puntos fuertes

- Libre (*Open-source*)
- A partir de un determinado código generar una aplicación en múltiples plataforma.
- Compila una aplicación nativa.
- Aspecto “nativo” de las aplicaciones.
- Potente IDE para el desarrollo de aplicaciones (integra edición y generación).
- Permite trabajar con emuladores y con el teléfono.

- Librerías JavaScript que acceden al hardware del teléfono (cámara, GPS, acelerómetro).
- Gran cantidad de ejemplos y tutoriales.
- Comunidad en expansión y muy activa.
- Genera código nativo para cada plataforma.
- Posibilidad de añadir plugins con código nativo de casa plataforma (Java/Android, Objective C/iPhone).
- Uso JavaScript/JSON como lenguaje principal (punto fuerte si sabes JavaScript).
- Generación y testeo de aplicaciones desde el Framework.

Puntos débiles

- Inclusión de *debugger* como opción de pago dentro de las opciones de Titanium Studio.
- Soporta solo Android y iOS (BlackBerry en fase Beta).
- Uso JavaScript/JSON como lenguaje principal (punto débil si no se tiene conocimientos de JavaScript).

3.4 Comparativa y elección de plataforma

Una vez vistos los principales aspecto de cada plataforma, el siguiente paso es analizar estos resultados, centrándonos en los aspectos a resaltar de cada una. En primer lugar nos vamos a centrar en PhoneGap. Como hemos visto en el apartado 3.3.2, PhoneGap es una plataforma fuertemente basada en tecnologías Web, basándose casi por entero en estas. Este es un punto muy importante para ellos, y su principal argumento para atraer desarrolladores, junto con su otro punto fuerte, que es el soporte de la mayoría de plataformas móviles. Pero todo tiene un coste, y el principal problema de PhoneGap es que la aplicación generada no es nativa, sino una aplicación que se ejecuta sobre el navegador del dispositivo. La aplicación generada tendrá un consumo de recursos elevado, lo que repercutirá en una disminución del rendimiento. La interfaz no nativa de las aplicaciones de PhoneGap es otro aspecto negativo. Por todo esto, podemos concluir que PhoneGap es una plataforma idónea para desarrolladores Web que quieren hacer uso de sus conocimientos, que además quieren llegar al máximo número de dispositivos y que no le dan importancia al aspecto nativo y a la perdida de rendimiento.

Respecto a Titanium, una de las características que más importantes y su principal atractivo es el aspecto "nativo" de sus aplicaciones. Las aplicaciones desarrolladas con Titanium tienen un aspecto prácticamente idéntico a las nativas. Actualmente soporta iOS y Android, además de BlackBerry en fase Beta. Otro aspecto a destacar es el crecimiento espectacular que ha tenido Titanium, siendo la última plataforma en aparecer de las tres analizadas. La principal consecuencia es la existencia de una gran comunidad de desarrolladores detrás, algo básico para el éxito de una plataforma de este estilo. Un hecho que demuestra la viabilidad futura de la plataforma es la compra que hicieron de la compañía Aptana por más de 60 millones de dólares. Aptana es la empresa desarrolladora de un popular IDE de desarrollo Web, usado por 1,5 millones de desarrolladores. Como consecuencia de esta compra, Titanium integró

en su herramienta de desarrollo todas las facilidades para el desarrollador que proporciona Aptana, junto con las herramientas propias de generación de aplicaciones que proporciona Titanium. Una vez dicho todo esto, podemos concluir que Titanium es una plataforma idónea para aquellos que quieren una aplicación nativa para iOS/Android, que consideran fundamental el tener una interfaz con elementos nativos y que buscan una plataforma con futuro.

	Titanium	Rhodes	Phonegap	
Android	✓	✓	✓	
iOS	✓	✓	✓	
BlackBerry	✎	✓	✓	El soporte de Titanium a BlackBerry está en fase Beta.
Symbian	✗	✓	✓	
Windows Phone 7	✗	✓	✓	
Interfaz nativa	✓	✎	✎	PhoneGap hace uso de componentes Web. Rhomobile soporta unos pocos componentes nativos.
Código nativo	✓	✓	✓	
Generación en local	✓	✓	✎	PhoneGap tiene librerías de terceros.
Generación sin Mac	✗	✗	✗	Generar iOS apps necesita un Mac.
IDE propio	✓	✎	✗	RhoStudio está en fase Beta
Comunidad	✓	✓	✓	
Lenguajes	JS	Ruby, HTML, CSS	HTML, CSS, JS	
Soporte Pro ($/año)	499-2388	5.000-10.000	249-19.999	

Tabla 3.1 Comparativa de funcionalidad general

En la Tabla 3.1 cada fila hace referencia a una de las características generales de cada una de las plataformas. El símbolo ✓ significa que la plataforma soporta esa característica, el símbolo ✎ representa que esa característica está soportada en fase Beta o de manera parcial. Por último, el símbolo ✗ significa que esa característica no está soportada por la plataforma.

En la Tabla 3.2 se muestra una comparativa del soporte que da cada una de las plataformas analizadas respecto a las funcionalidades que son accesibles del teléfono.

	Titanium	Rhomobile	Phonegap	
Acelerómetro	✓	✓	✓	
Vibración	✓	✓	✓	
Contactos	✓	✓	✓	
Cámara	✓	✓	✓	
Multitouch	✓	✓	✓	
Geolocalización	✓	✓	✓	
SMS	✓	✓	✓	
API del teléfono	✓	✓	✓	
Acceso sist. de ficheros	✓	✓	✓	
Copy/Paste	✓	✓	✓	
Sonidos	✓	✎	✎	PhoneGap y Rhomobile no puede grabar sonidos.
Captura de video	✓	✗	✗	
Bluetooth	✗	✗	✗	

Tabla 3.2 Comparativa APIs soportadas

Respecto a Rhomobile, al igual que sucede con PhoneGap, soporta las principales plataformas móviles del mercado. Rhomobile está basado en Ruby, un lenguaje utilizado en el desarrollo Web. Uno de los aspectos en el que Rhomobile es claramente mejor que las otras dos propuestas, es en la sincronización de datos con otros sistemas. Esto es algo muy útil si por ejemplo nuestra aplicación está enfocada al entorno empresarial. Pero en este punto aparece también uno de los inconvenientes de Rhomobile, que es el pago de suscripciones. Un aspecto que puede ser clave en el desarrollo de una aplicación y que puede hacer a un desarrollador decidirse por Rhomobile, se convierte en una razón para elegir otra. Comparándola con Titanium, la viabilidad futura de Rhomobile puede ser más incierta, debido sobre todo a su menor adopción por parte de los desarrolladores. La vida de una plataforma viene determinada por sus usuarios, y si no se produce un despegue, la plataforma corre el riesgo de desaparecer a medio/largo plazo. Como comentario final, podemos concluir que Rhomobile es una plataforma idónea para desarrolladores en Ruby, que quieren una aplicación cuasi nativa para la mayoría de plataformas y que no tienen inconveniente en pagar si quieren un soporte para la sincronización de datos con otros sistemas.

3.4.1 Elección de plataforma

Con el objetivo de encontrar la plataforma más acorde a nuestras necesidades y objetivos, en este capítulo hemos realizado un estudio sobre los tres Frameworks más importantes para el desarrollo multiplataforma de aplicaciones móviles. Como premisa teníamos claro que todas las plataformas tienen sus puntos fuertes y sus debilidades, y de que no hay una plataforma perfecta. Además, la elección se tenía que realizar sin tener un conocimiento profundo de ninguna de las tres, ya que uno no sabe el verdadero potencial de una plataforma hasta que hace un autentico desarrollo de una aplicación y se enfrenta a los problemas reales que ello acarrea.

Después del estudio realizado, el siguiente paso es elegir una plataforma para el desarrollo de nuestra aplicación. El primer paso es descartar uno de los Frameworks, y éste fue PhoneGap. El hecho de que la aplicación generada no sea nativa, sino una aplicación que se ejecuta sobre el navegador del dispositivo, fue un factor determinante. La aplicación generada tendrá un consumo de recursos elevado, lo que repercutirá en una disminución del rendimiento. A esto hay que unir el hecho del aspecto no nativo de la aplicación, algo muy importante de cara al usuario final.

Una vez desechada PhoneGap, la elección se centraba en dos candidatas, Titanium y Rhomobile. Debido a una serie de factores que se detallan a continuación, la plataforma elegida ha sido Titanium. Tanto Rhomobile como Titanium tienen sus respectivos puntos a favor y en contra, pero los aspectos positivos de Titanium han prevalecido.

El por qué de la elección de Titanium viene determinado principalmente por tres factores:

- Aspecto "nativo" de las aplicaciones. Titanium hace uso de componentes de interfaz nativos, otorgando a las aplicaciones un aspecto idéntico al de una aplicación desarrollada de una manera tradicional.

- Crecimiento espectacular de la plataforma. Este hecho viene refrendado por la gran cantidad de desarrollares que forman parte de la comunidad.

- Viabilidad futura de la plataforma. La compra de la compañía Aptana por más de 60 millones de dólares demuestra la ambición y las ganas de crecer de Titanium.

3.5 Conclusiones

En este capítulo hemos visto el estado del arte del desarrollo móvil multiplataforma.

En primer lugar hemos visto la situación actual del mercado de los *smartphones* y como la aparición de las aplicaciones móviles en las distintas plataformas ha producido un problema de fragmentación al que tienen que hacer frente los desarrolladores.

Después hemos profundizado un poco más en el problema de la fragmentación. Para ello hemos descrito las causas que la provocan, especificando cada una de ellas. Una vez hecho esto, también era importante destacar la importancia de la fragmentación, haciendo hincapié en su influencia negativa en cada una de las etapas de desarrollo de una aplicación.

Una vez situados en el mercado actual de *smartphones* y visto la problemática asociada a este, el siguiente paso es ver las alternativas existentes para solventarla. Actualmente existen en el mercado multitud de entornos multiplataforma para el desarrollo de aplicaciones móviles. De entre todas ellas, en este capítulo se han seleccionado tres para poder realizar un estudio y poder determinar en qué consiste cada una de ellas. En el estudio nos hemos fijado en las principales características de cada una de ellas, como son el tipo de aplicación generada, la interfaz y las APIs de la plataforma, las herramientas usadas para el desarrollo y la generación de la aplicación.

Por último, hemos realizado un análisis conjunto de las tres plataformas, centrándonos en los aspectos a resaltar de cada una, además de mostrar tablas comparativas de las principales características y funcionalidades soportadas. Una vez hecho esto, hemos elegido Titanium como plataforma para nuestro desarrollo.

Capítulo 4

4. Titanium

4.1 Introducción

Una vez elegida Titanium como plataforma para nuestro desarrollo, el siguiente paso es conocerla en profundidad. Como hemos visto en el apartado 3.3.4, Titanium es una plataforma *open-source* para el desarrollo multiplataforma de aplicaciones móviles. Titanium nos permite crear, ejecutar y empaquetar aplicaciones nativas para iOS, Android y BlackBerry (en fase Beta).

Para usar una plataforma de desarrollo de este tipo es importante conocerla en profundidad. La arquitectura de Titanium es la que define su comportamiento. Su proceso de funcionamiento, en el que partiendo de un código JavaScript se generan aplicaciones para distintas plataformas, consta de varias fases. A grandes rasgos, estas fases se pueden describir de la siguiente manera:

- El desarrollador escribe el código HTML/CSS/JavaScript que define la lógica y la interfaz de la aplicación, haciendo uso de los distintos APIs que acceden a la funcionalidad del teléfono.
- El compilador de Titanium se encarga de convertir el código Web en código nativo de la plataforma especificada.
- Se ejecutan un conjunto de instrucciones en tiempo de ejecución que empaquetan la aplicación para ser distribuida.

Dentro de la arquitectura de una plataforma, destacamos los fundamentos del diseño. Titanium sigue el estándar MVC (*model-view-controller*), muy extendido en la aplicaciones Web.

Para entender mejor como es el desarrollo de una aplicación, es básico conocer como es la estructura de un proyecto Titanium. Al generar un proyecto nuevo, se genera una estructura de forma automática, a partir de la cual iremos añadiendo funcionalidad. El desarrollador basa todo su trabajo en el uso de las distintas APIs desarrolladas por Titanium. Mediante estas APIs, el desarrollador diseña la lógica y la interfaz de la aplicación, además de acceder a las distintas funcionalidades del teléfono. Una de las características principales de Titanium es que la interfaz es completamente nativa.

4.2 Arquitectura Titanium

En primer lugar vamos a describir en qué consiste la arquitectura propia de Titanium [12]. Por un lado, repasaremos como es el funcionamiento interno de la plataforma, explicando las distintas partes del proceso de generación de la aplicación nativa. Por otro lado, explicaremos brevemente cuales son los conceptos de diseño en los que se basa una aplicación Titanium, haciendo hincapié en la estructura de ventanas y vistas.

4.2.1 Funcionamiento

Como hemos comentado en el capítulo 3.3.4.12, Titanium genera una aplicación nativa a partir de un desarrollo en JavaScript. A grandes rasgos, el funcionamiento de Titanium consiste en traducir el código JavaScript a código nativo y después invocar a las herramientas de la plataforma en cuestión para generar el paquete final. Este proceso se puede dividir en tres bloques:

- Precompilador.
- Compilador *Front-End*.
- Compilador y empaquetador de la plataforma.

Precompilador

La función del precompilador se divide en dos tareas. Cuando un desarrollador escribe código, se generan multitud de saltos de línea, espacios en blanco, etc. El primer paso realizado por el precompilador consiste en optimizar el código JavaScript escrito por el desarrollador. En segundo lugar, el precompilador crea una dependencia jerárquica de todas la APIs utilizadas en nuestra aplicación, la cual será utilizada en los siguientes pasos del proceso seguido por Titanium.

Compilador *Front-End*

El objetivo de Titanium es generar una aplicación nativa, equivalente a una aplicación generada por el entorno de desarrollo de cada plataforma móvil. Para ello necesitamos que nuestra aplicación esté en código nativo. La función del compilador *Front-End* es generar el código nativo que pueda ser compilado por las herramientas propias de una plataforma. Para ello, genera el código nativo específico de la plataforma, generando un proyecto nativo si es necesario.

Compilador y empaquetador de la plataforma

Una vez tenemos el código generado por el compilador *Front-End*, se compilará la aplicación nativa final mediante las herramientas propias de la plataforma. En el caso de Android mediante el Android SDK, y en caso de Apple mediante Xcode. Una vez obtenido este paquete final, podemos ejecutar la aplicación sobre el simulador, sobre el dispositivo o preparar el paquete para la distribución.

4.2.2 Diseño

Para la construcción de la aplicación, Titanium sigue el estándar MVC (*model-view-controller*), muy extendido en la aplicaciones Web. Una de las características principales de Titanium es que la interfaz es completamente nativa. Mediante el conjunto de APIs que conforman *Titanium.UI*, tenemos acceso a los distintos componentes que formarán la interfaz de nuestra aplicación. Los principales componentes de diseño en Titanium son:

- **Ventanas:** componentes que contendrán una o varias vistas.
- **Vistas:** componentes que muestran contenido en la pantalla.
- **Widgets:** son tipos especiales de vistas.

A continuación se describirán las principales características de los componentes básicos en el diseño de una aplicación Titanium.

4.2.2.1 Ventanas

En primer lugar, vamos a describir en qué consisten las ventanas dentro del diseño de una aplicación. Éstas son componentes que pueden contener una o más vistas en su interior, las cuales son también vistas en sí mismas. Se pueden abrir, cerrar, son configurables (pantalla completa, modal, pantalla partida). A la hora de crear una ventana, hay dos posibilidades:

- Crear una ventana y en el mismo fichero añadir su contenido.
- Crear la ventana y llamar a otro fichero que contendrá sus elementos.

En la Figura 4.1 se muestra un ejemplo de una ventana creada y definida en un mismo archivo:

```
var ventana = Titanium.UI.createWindow();
var vista = Ti.UI.createView();
ventana.add(view);
ventana.open();
```

Figura 4.1: Código ventana básica

Para las ventanas cuyo contenido se define en otro fichero, en el fichero actual se crea el objeto ventana, especificando el archivo que contendrá sus elementos y su lógica:

```
var ventana = Titanium.UI.createWindow({
    url:'combo.js',
    title:'Vista'
});
ventana.open();
```

Figura 4.2: Código ventana referenciada en otro fichero

En el fichero *combo.js*, cuyo contenido se muestra en la Figura 4.3, obtenemos la referencia de la ventana y especificamos los elementos que la componen:

```
var ventana = Ti.UI.currentWindow;
var vista = Ti.UI.createView();
ventana.add(vista);
```

Figura 4.3: Código fichero referenciado en la ventana

4.2.2.2 Vistas

Las vistas son los componentes básicos de la interfaz de usuario de una aplicación Titanium. Las vistas son componentes jerárquicos ya que pueden tener otros componentes gráficos que dependan de ellas, que podemos denominar como hijos. *Titanium.UI.View* es el objeto mediante el cual se define una vista básica, pero dentro del API *Titanium.UI* hay varios objetos que son especializaciones de este tipo de vista. Como ejemplo, podemos citar *TableView, ImageView, WebView*, etc.

En la Figura 4.4 se muestra un ejemplo de creación de una vista en base a una imagen:

```
var ventana = Ti.UI.createWindow();
var vista = Ti.UI.createView({backgroundImage:"imagen.png"});
ventana.add(view);
ventana.open();
```

Figura 4.4: Ejemplo básico de una vista

Otro aspecto a destacar de las vistas es que pueden interactuar según determinados eventos. Una vez creada la vista se le añade un *eventlistener*, el cual ejecutará su código asociado cuando se produzca el evento definido, como se muestra en la Figura 4.5:

```
vista.addEventListener('dblclick',function()
{
   vista.animate({width:vista.width*2,height:vista.height*2,duration:2000});
});
```

Figura 4.5: Código listener de una vista

4.2.2.3 Widgets

Los Widgets son tipos específicos de vistas, pero que tienen una funcionalidad especial. Como ejemplos de Widgets podemos citar los botones o los campos de texto. Un aspecto que diferencia los Widgets de las vistas es que no pueden tener hijos. Como ejemplo, en la figura se muestra la creación de un botón, con un evento mediante el cual al pulsarlo se abre otra ventana:

```
var boton = Ti.UI.createButton({
  title:"Entrar",
  width:70,
  height:35
});
button.addEventListener("click",function(){
  var ventana = Titanium.UI.createWindow({
        url:'combo.js',
        title:'Vista'
    });
    ventana.open();
});
```

Figura 4.6: Código creación de un botón

4.3 Estructura aplicación Titanium

A la hora de diseñar una aplicación con Titanium, es importante tener claro la estructura básica de un proyecto [19]. Cuando creamos un nuevo proyecto, se generan automáticamente una serie de ficheros y carpetas que conforman el esqueleto de la aplicación. A partir de esta estructura inicial, podemos ir añadiendo componentes y funcionalidades a nuestra aplicación.

4.3.1 Carpeta del proyecto

La estructura básica de un proyecto consta de un conjunto de ficheros y carpetas, los cuales se detallan a continuación:

- **CHANGELOG.txt:** Fichero encargado de mostrar los cambios de la versión respecto a versiones anteriores.
- **LICENSE:** Fichero que contiene la licencia de Appcelerator. No se distribuye con la aplicación.
- **LICENSE.txt:** Fichero con la licencia para el usuario final.
- **manifest:** Fichero que contiene información básica del proyecto, como el nombre de la aplicación, el desarrollador, el icono, etc.
- **README:** Fichero que describe la aplicación. No se distribuye con la aplicación.
- **tiapp.xml:** Fichero que describe diversos aspectos del proyecto. Es usado en la ejecución de la aplicación y también cuando se procede al empaquetado de ésta.
- **build:** Carpeta que contiene los ficheros específicos de cada plataforma generados por Titanium. Estos ficheros son los ficheros que las herramientas propias de la plataforma (Android SDK o Xcode) usarán para compilar y generar la aplicación final. Estos ficheros se modifican automáticamente conforme vamos haciendo cambios en nuestra aplicación, por lo que es recomendable no modificarlos directamente.
- **Resources:** Carpeta que contiene los ficheros que conforman la aplicación, principalmente ficheros JavaScript con el código desarrollado e imágenes usadas por la aplicación. Es la carpeta más importante, ya que contiene todos los elementos que hemos desarrollado y que dotan de funcionalidad a la aplicación.

- **i18n:** Carpeta opcional, que contendrá los ficheros necesarios para la internacionalización de la aplicación [17]. Esta carpeta no se crea de manera automática.

En la Figura 4.7 se muestran los elementos dentro de la carpeta del proyecto:

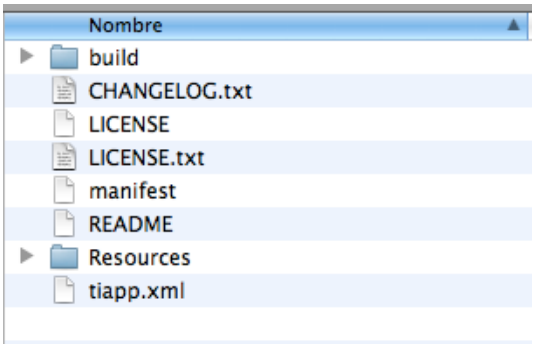


Figura 4.7: Estructura aplicación

4.3.2 Carpeta Resources

Como hemos comentado en el apartado anterior, la carpeta *Resources* es la carpeta principal durante el desarrollo de la aplicación. El elemento central de esta carpeta, y por añadidura de la aplicación, es el fichero *app.js*.

app.js

Este fichero es el punto de entrada en la aplicación. Además, debido a que el proyecto está basado en JavaScript, este fichero actuará de contexto de ejecución raíz del conjunto de la aplicación. Teóricamente, toda la lógica de la aplicación podría incluirse en este fichero, pero lo normal es dividir la aplicación en múltiples ficheros, incluyéndolos en el mismo contexto.

Carpetas especificas por plataforma

Dentro de la carpeta *Resources* existen subcarpetas donde colocar elementos específicos para iOS y Android. Estos subdirectorios son la carpeta *android* y la carpeta *iphone*. En estas carpetas se colocan elementos como el icono de la aplicación para cada una de las plataformas, la imagen de inicio, etc. También podemos colocar ficheros específicos para cada plataforma que se necesite sobrescribir de la carpeta *Resources*.

4.4 Titanium API

Una vez hemos visto como funciona la arquitectura interna de Titanium y como es la estructura de un proyecto, el siguiente paso es ver algunas de las APIs más importantes, mostrando ejemplos de su funcionamiento.

Todos los elementos necesarios para construir nuestra aplicación se hallan en un conjunto de APIs desarrolladas por Titanium. Cada uno de estas APIs se divide en

distintas categorías, guardando siempre la nomenclatura Titanium.*, como por ejemplo, *Titanium.Map* o *Titanium.UI*.

4.4.1 Interfaz

Una de las características principales de Titanium es que la interfaz es completamente nativa. Mediante el conjunto de APIs que conforman Titanium.UI, tenemos acceso a los distintos componentes que formarán la interfaz de nuestra aplicación. La lista de componentes es muy extensa: vistas, botones, tablas, combos, etc.

Como hemos comentado en el apartado 4.2.2, el diseño de la interfaz se basa principalmente en el uso de ventanas y vistas. Para poder ver de una forma concreta como es el proceso de diseño, vamos a ver el funcionamiento de las tablas en un desarrollo Titanium.

4.4.1.1 Tablas

Las tablas son uno de los elementos de interfaz más comunes dentro de cualquier aplicación móvil. En este apartado vamos a describir las características básicas del uso de tablas dentro de Titanium [20].

Para la creación y uso de tablas, Titanium dispone del API *Titanium.UI.TableView*. Gracias al API proporcionado, disminuye la complejidad que comporta el uso de tablas dentro del desarrollo de una aplicación, siendo incluso más sencilla que en las plataformas nativas. De entre la funcionalidad proporcionada podemos destacar lo siguiente:

- Personalizar el aspecto de las tablas y de sus filas.
- Poder añadir y eliminar filas de manera dinámica.
- Poder añadir cabeceras en el encabezado y en el pie de las tablas.
- Posibilidad de añadir una barra de búsqueda a la tabla.

Una vez hemos decidido hacer uso de tablas en nuestra aplicación, debemos decidir el nivel de complejidad que queremos darle, para lo cual elegiremos entre *Simple TableView* o *Standard TableView*.

Simple TableView

De entre las dos opciones disponibles para la creación de tablas, *Simple TableView* es la que menor complejidad comporta. El principal aspecto que la caracteriza es la manera en que se especifican los datos que poblarán la tabla. Para ello, se creará un *array* o matriz de objetos, cada uno de los cuales contendrá la información a mostrar. En cada uno de estos objetos se especificará la información en la propiedad *title*, y si se quiere cambiar el color del texto se especificará en la propiedad *color*.

```
// Array con los datos de la tabla
var dataPaises = [
   {title:'Spain', hasChild:true},
   {title:'France', hasChild:true},
   {title:'Great Britain', hasChild:true},
   {title:'United States', hasChild:true},
];

// Se crea la tabla en base al array
var tabla = Titanium.UI.createTableView({
       data:dataPaises
});
```

Figura 4.8: Código *Simple TableView*

Standard TableView

Cuando nuestra tabla requiera de más complejidad, usaremos la *Standard TableView*. La principal característica de este tipo de tabla es que para diseñar las filas con un aspecto más complejo hace uso de distintas vistas, botones, etc. Esta es la principal diferencia con la *Simple TableView,* la cual hace uso de un *array* o matriz para poblar la tabla. En una *Standard TableView,* haremos uso de objetos *Titanium.UI.TableViewRow*. Mediante estos objetos podemos crear filas como si se tratará de vistas, utilizando distintos componentes de interfaz.

```
// Array con información a utilizar
var dataPaises = [
   {title:'Spain', flag:'images/es.png', trend:'up.png',hasChild:true},
   {title:'France', flag:'images/fr.png',trend:'down.png',hasChild:true},
   {title:'Great Britain', flag:'images/gb.png', trend:'down.png',hasChild:true},
   {title:'United States', flag:'images/us.png', trend:'up.png', hasChild:true},
];
// Creación de la tabla, usando TableViewRow para cada fila
var tabla = Titanium.UI.createTableView();
var data=[];
for (var i = dataPaises.length - 1; i >= 0; i--) {
   var row = Titanium.UI.createTableViewRow();
   var flag =  Titanium.UI.createImageView({
       url:dataPaises[i].flag,
       width:32,height:32,left:4,top:2
   });
   var country = Titanium.UI.createLabel({
       text:dataPaises[i].title,
       font: {fontSize:16,fontWeight:'bold'},
       width:'auto',textAlign:'left',
       top:3,left:40
   });
   var trend =  Titanium.UI.createImageView({
       url:dataPaises[i].trend,
       width:16,height:16,right:30
   });
   row.add(flag);
   row.add(country);
   row.add(trend);
   row.hasChild=dataPaises[i].hasChild;
   row.className = 'coutry_row';
   data[i] = row;
};
tableview.setData(data);
```

Figura 4.9: Código *Standard TableView*

Descripción de la prueba

Para probar el uso de las tablas se diseño una aplicación cuya interfaz estuviera compuesta por una *Standard TableView*. Las filas de la tabla mostrarán información de distintos países, como la bandera y el nombre del país. La composición de estas filas se hará mediante objetos *Titanium.UI.TableViewRow*.

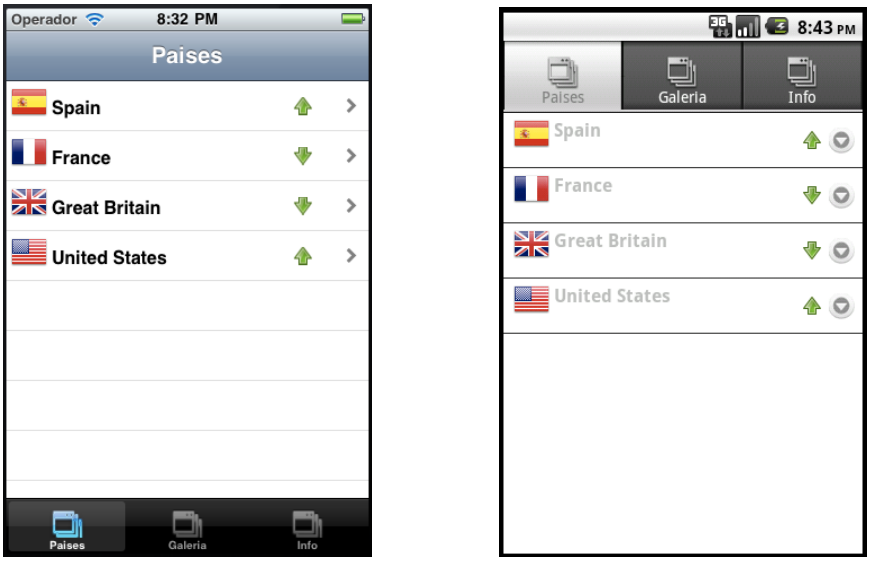


Figura 4.10: Aplicación ejemplo tablas

Resultados de la prueba

La tabla se muestra correctamente en ambas plataformas. Las imágenes y el texto se visualizan tanto en iOS como en Android, aunque su disposición difiere ligeramente entre las dos plataformas.

Conclusiones de la prueba

El desarrollo de una tabla compuesta por filas complejas, que contienen vistas e imágenes además de información, fue satisfactorio tanto para iOS como para Android. De todas formas, hay que tener en cuenta que las propiedades que definen la posición de cada componente de interfaz no se traducen exactamente igual para cada plataforma. Este hecho es importante, y obliga a probar muy bien todos los cambios en ambas plataformas para asegurarnos de la correcta visualización de cada componente.

4.4.1.2 Diferencias entre iOS/Android

Uno de los aspectos importantes a tener en cuenta en el tema de la interfaz, es que ésta difiere en algunos aspectos entre las plataformas iOS y Android. En la mayoría de los casos el desarrollador puede escribir su código teniendo la seguridad de que la interfaz saldrá perfectamente para cada una de las plataformas. Pero si queremos mantener los fundamentos de diseño de interfaz de cada plataforma, tenemos que atender a ciertos detalles para que la experiencia final de usuario sea exacta a la de una aplicación desarrollada directamente con las herramientas nativas. Además, existen

APIs específicas para cada una de las dos plataformas si queremos darle un toque mucho más nativo a la aplicación, teniendo en cuenta que esa API no es válida para el resto de plataformas.

Una forma sencilla de diferenciar las partes de código que son distintas para las dos plataformas es definirse una variable booleana que determina si la aplicación en ejecución es Android o no.

```
var isAndroid = Ti.Platform.osname == 'android';
```

Figura 4.11: Variable ejecución sobre Android

4.4.2 Comunicaciones

Es muy posible que nuestra aplicación necesite comunicarse con otros sistemas o servicios Web. Para ello, Titanium dispone del API *Titanium.Network* [14]. De entre las distintas funciones del API, nos vamos a centrar en la comunicación con servicios Web mediante GET/POST y mediante SOAP.

La manera que tiene el API de Titanium de conectarse con otros servidores es por HTTP, mediante el objeto *Titanium.Network.HTTPClient*. El comportamiento de un objeto de este tipo es equivalente al objeto *XMLHTTPRequest* en un navegador Web. A continuación, mediante distintos ejemplos veremos cómo es el funcionamiento de Titanium para este tipo de comunicaciones.

4.4.2.1 GET

El proceso necesario para comunicarnos con otros sistemas consta de una serie de pasos. En primer lugar se crea la petición mediante las APIs de Titanium. Una vez creado el objeto el objeto que contiene la información de la petición, la lanzamos. Por último obtenemos la respuesta, la cual puede estar en formato XML o en formato JSON, dependiendo del tipo de retorno del servicio Web. Es importante destacar que la posibilidad de realizar peticiones GET está disponible tanto para iOS como para Android.

A continuación vamos a ver algunos aspectos respecto al código necesario para realizar una petición a un servicio Web. Este código tiene como mínimo tres partes:

- **createHTTPClient:** función que crea el objeto HTTPClient, a partir del cual se desarrollará toda la lógica.
- **open:** método que se encarga de abrir la conexión HTTP.
- **send:** método encargado de mandar la petición al servicio Web.

Junto con estos métodos, el API de Titanium proporciona otros métodos muy útiles de cara a la comunicación con el servicio Web. El uso o no de estos métodos dependerá de cuál sea nuestro objetivo:

- **onload:** función que tratará la respuesta del servicio Web, una vez este responde a nuestra petición.
- **onerror:** función que tratará la lógica a seguir cuando se produce un error.

En la Figura 4.12 se muestra el código básico de creación de una petición GET:

```
var xhr = Ti.Network.createHTTPClient();
xhr.onload = function(e) {
    // Aqui va la lógica que recoge la respuesta del servidor
    //y la muestra en pantalla
};
xhr.onerror = function(e) {
            Ti.API.info('IN ERROR ' + e.error);
};
xhr.open('GET', urlWebService);
xhr.send();
```

Figura 4.12: Petición GET

Una vez hemos realizado la petición, el siguiente paso es el tratamiento de la respuesta. Como hemos comentado antes, la respuesta enviada por el servicio Web consultado puede ser en XML o en JSON. Debido a que Titanium está basado en JavaScript, el tratamiento de XML puede llegar a ser problemático. Por esta razón se recomienda el uso de JSON siempre que sea posible, ya que JSON es un formato de intercambio de datos pensado especialmente para JavaScript.

Entre las ventajas de JSON respecto a XML, se pueden citar las siguientes:

- Facilidad para serializar una respuesta JSON en objetos JavaScript.
- Menor consumo de ancho de banda.
- Rapidez.

En la figura se muestra un ejemplo de tratamiento de una respuesta de tipo JSON:

```
xhr.onload = function(e) {
    // Creamos un objeto JSON a partir de la respuesta del servidor
    var objetoJSON = JSON.parse(this.responseText);
    // A continuación iria la lógica que accede a la info almacenada
    // en el objeto JSON
}
```

Figura 4.13: Tratamiento respuesta JSON

Si el servicio Web a consultar solo soporta XML, el proceso de tratamiento de la respuesta seria más laborioso, al tener que ir recorriendo los distintos nodos del XML.

```
xhr.onload = function(e) {
        // Este es edocumento XML recuperado en la respuesta
        var doc = this.responseXML.documentElement;
        // Con el documento XML recuperamos la información a partir de
        // etiquetas de los nodos
        var elems = doc.documentElement.getElementsByTagName('etiqueta');
}
```

Figura 4.14: Tratamiento respuesta XML

Con el fin de comprobar el funcionamiento del API, diseñamos una serie de aplicaciones de prueba a modo de ejemplo.

Descripción de la prueba

Para probar el uso de una petición GET se diseñó una aplicación que recuperaba la previsión del tiempo dentro de un conjunto de ciudades. Además de probar el uso de una petición, el objetivo de la prueba era diseñar una aplicación que hiciera uso de distintos componentes de interfaz para probar su funcionamiento. En la pantalla principal se muestra un combo con las distintas ciudades y un botón para obtener la previsión. Una vez elegida la ciudad y pulsado el botón "Tiempo actual", en una segunda pantalla se muestra el icono del estado actual, junto a diferentes datos como temperatura, humedad etc. En la Figura 4.15 se muestran dos capturas de la aplicación:

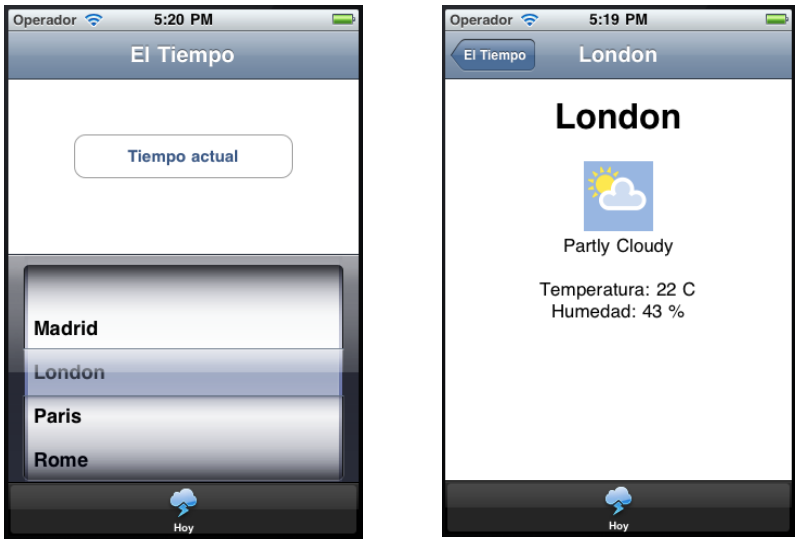


Figura 4.15: Aplicación ejemplo GET

El servicio Web utilizado en la aplicación estaba preparado para enviar la respuesta en formato JSON. En la url de la petición se especifican los parámetros del servicio. La sentencia GET enviada seria:

```
var url = 'http://www.worldweatheronline.com/feed/weather.ashx?q='
          +window.title+'&format=json&num_of_days=2&key=348f47bb43223116110603';
xhr.open('GET', urlWebService);
xhr.send();
```

Figura 4.16: Petición GET ejemplo

Resultados de la prueba

Una vez enviada la petición, el servicio Web la procesa y a continuación envía la respuesta en formato JSON, la cual es tratada y serializada a objetos JavaScript, que usaremos para mostrar la información. En la Figura 4.17 se muestra una parte de la

respuesta en formato JSON del servicio Web. El resultado de la prueba es satisfactorio tanto en iOS como en Android.


```
"current_condition": [ {
   "cloudcover": "75",
   "humidity": "65",
   "observation_time": "10:04 PM",
   "precipMM": "0.0",
   "pressure": "1031",
   "temp_C": "4",
   "temp_F": "39",
   "visibility": "10",
   "weatherCode": "116",
   "weatherDesc": [ {"value": "Partly Cloudy" } ],
   "weatherIconUrl": [
       {"value": "http:\/\/www.worldweatheronline.com\/images\
       /wsymbols01_png_64\/wsymbol_0004_black_low_cloud.png" }
   ],
   "winddir16Point": "E",
   "winddirDegree": "90",
   "windspeedKmph": "20",
   "windspeedMiles": "13"
   } ],
```


Figura 4.17: Parte respuesta JSON ejemplo GET

Conclusiones de la prueba

Gracias a esta prueba hemos conseguido probar tanto el uso de una petición GET a un servicio Web, como el diseño de una sencilla interfaz mediante distintos componentes. El hecho de que el servicio Web devolviera la respuesta mediante JSON facilitó el proceso. Los distintos componentes de interfaz de interfaz se mostraban correctamente en las dos plataformas, sin tener que hacer ninguna distinción en el código desarrollado.

4.4.2.2 POST

Además de realizar peticiones mediante GET, también podemos realizar peticiones a servicios Web mediante POST. El funcionamiento y la creación de este tipo de peticiones es casi idéntico al procedimiento para GET, pero con algunas diferencias. La principal diferencia se centra en que los parámetros se especifican dentro del apartado *send*, y no dentro del apartado *open* junto a la url.

Descripción de la prueba

Para probar el uso de una llamada POST se usó un servicio Web preparado para recibir este tipo de peticiones. En el caso del ejemplo, se utilizó un servicio Web que devolvía un XML con la previsión del tiempo de una ciudad. Como se muestra en la Figura 4.18, los parámetros de la petición, el nombre de la ciudad y el nombre del país, se especifican dentro del método *send* del objeto *HTTPClient*.

```
xhr.open('POST','http://www.webservicex.net/globalweather.asmx/GetWeather');
xhr.setRequestHeader('Content-Type','text/xml;charset=utf-8');
xhr.send({
      CityName:'munich',
      CountryName:'germany'
});
```

Figura 4.18: Código petición POST

Resultados de la prueba

Una vez realizada la petición, el servicio Web devuelve la respuesta XML en la que se recogen los datos referentes la meteorología de Múnich. En la Figura 4.19 se muestra un ejemplo de respuesta de tipo XML. Al surgir problemas en el tratamiento del XML, utilizamos el servicio de *Q&A* de la comunidad de Titanium. Al buscar problemas similares vimos que otros desarrolladores también habían tenido problemas con el tratamiento de XML.

```
<string xmlns="http://www.webserviceX.NET">&lt;?xml version="1.0" encoding="utf-16"?&gt;

&lt;CurrentWeather&gt;

  &lt;Location&gt;Munchen, Germany (EDDM) 48-21N 011-47E&lt;/Location&gt;

  &lt;Time&gt;Aug 13, 2011 - 12:20 PM EDT / 2011.08.13 1620 UTC&lt;/Time&gt;

  &lt;Wind&gt; from the W (270 degrees) at 5 MPH (4 KT) (direction variable):0&lt;/Wind&gt;

  &lt;Visibility&gt; greater than 7 mile(s):0&lt;/Visibility&gt;

  &lt;SkyConditions&gt; partly cloudy&lt;/SkyConditions&gt;

  &lt;Temperature&gt; 73 F (23 C)&lt;/Temperature&gt;

  &lt;DewPoint&gt; 57 F (14 C)&lt;/DewPoint&gt;

  &lt;RelativeHumidity&gt; 56%&lt;/RelativeHumidity&gt;

  &lt;Pressure&gt; 29.88 in. Hg (1012 hPa)&lt;/Pressure&gt;

  &lt;Status&gt;Success&lt;/Status&gt;

&lt;/CurrentWeather&gt;</string>
```

Figura 4.19: Respuesta XML ejemplo POST

Conclusiones de la prueba

Mediante esta prueba hemos conseguido probar el funcionamiento de una petición POST. A diferencia de la prueba para GET, se ha utilizado un servicio cuya respuesta es un XML. El tratamiento de la respuesta en XML ha resultado ser más complejo que en el caso de una respuesta en formato JSON. Constatamos los problemas de tratar XML al consultar el de *Q&A* de la comunidad de Titanium.

4.4.2.3 SOAP

La realización de peticiones a servicios Web tipo SOAP es posible dentro de Titanium, pero sin un soporte especifico. Para realizar la comunicación con este tipo de servicios tenemos dos posibilidades. En primer lugar podemos crear la cabecera SOAP, con su correspondiente estructura y mandarla dentro de una petición GET. Esta opción no es aconsejable debido a lo laborioso del proceso. Si necesitamos utilizar un servicio Web mediante SOAP, la otra opción es usar la librería *suds*, la cual fue creada por un miembro de la comunidad y que facilita la comunicación con este tipo de servicios.

El funcionamiento de *suds* es sencillo. A continuación detallamos los pasos necesarios para su uso:

- Creación de un objeto *SudsClient*, especificando el *endpoint* y el *namespace* del servicio SOAP.
- Creación de una variable con los parámetros de la petición.
- Ejecución de la función *invoke*, dentro de la cual se trata la respuesta del servicio Web.

Descripción de la prueba

Para probar el uso de SOAP se utilizó el ejemplo contenido dentro del *Kitchen Sink*. Este ejemplo se encarga de realizar una petición a un servicio Web que realiza la conversión entre dos monedas.

```
var url = "http://www.webservicex.net/CurrencyConvertor.asmx";
var suds = new SudsClient({
    endpoint: url,
    targetNamespace: 'http://www.webserviceX.NET/'
});
var callparams = {
    FromCurrency: 'EUR',
    ToCurrency: 'USD'
};
try {
    suds.invoke('ConversionRate', callparams, function(xmlDoc) {
        // Aqui se trata la respuesta
    });
} catch(e) {
    Ti.API.error('Error: ' + e);
}
```

Figura 4.20: Petición ejemplo SOAP

Resultados de la prueba

El resultado de la prueba es satisfactorio. Se realiza la petición mediante *suds*, y el servicio Web responde con la cantidad de la moneda correspondiente. La respuesta es de tipo XML.

Conclusiones de la prueba

A pesar de no tener soporte oficial por parte de Titanium, es posible la realización de peticiones a servicios Web SOAP. El uso de *suds* es sencillo y funciona de manera satisfactoria. A pesar de recibir la respuesta en formato XML, el tratamiento ha generado menos problemas que en el ejemplo de petición POST. La posibilidad de realizar peticiones SOAP ha funcionado tanto en iOS como en Android.

4.4.2.4 Resumen de comunicaciones

Una vez probado el API *Titanium.Network*, se pueden extraer varias conclusiones. Podemos decir que gracias a este API podemos conectar con cualquier servicio Web. Dependiendo del tipo de servicio y sobre todo del tipo de respuesta, la facilidad del proceso variará.

Las pruebas con peticiones GET/POST fueron satisfactorias. El tratamiento de respuestas de tipo JSON es mucho más sencillo que el de respuestas XML. Después de probar sobre iOS y Android, el funcionamiento es correcto en ambas plataformas.

Respecto a SOAP, el hecho de que Titanium no lo soporte directamente puede parecer un problema, pero el uso de la librería *suds* facilita el trabajo. Las pruebas sobre iOS y Android fueron satisfactorias.

4.4.3 Datos locales

Un aspecto importante de cara al desarrollo de una aplicación, es la gestión de los datos locales [13], ya que hasta la aplicación más simple tiene la necesidad de almacenar algún tipo de datos. De cara a gestionar el almacenamiento de datos, Titanium dispone de tres tipos de objetos:

- **Titanium.App.Properties:** ideal para almacenar información de configuración de la aplicación.
- **Titanium.Filesystem:** da soporte a la manipulación de ficheros y directorios.
- **Titanium.Database:** permite el uso de bases de datos SQLite3.

4.4.3.1 Properties

El objeto *Titanium.App.Properties* es el objeto encargado de almacenar determinada información de forma sencilla. Se recomienda su uso para determinados casos, aunque esto puede variar en función del proyecto:

- Cuando la información consiste en clave/valor.
- La información está más relacionado con la aplicación que con el usuario.
- La información no requiere de otra información.
- Solo se necesita una versión de la información en un momento dado.

Su funcionamiento consiste en la escritura/lectura de propiedades. Estas propiedades se pueden definir en distintos tipos de datos: *Bool, String, Int, List, Double*.

Otra manera de trabajar con objetos *Properties* es la utilización de objetos JSON. En estos casos el funcionamiento seria pasar a *String* el objeto JSON para almacenar la propiedad, y parsear el *String* almacenado para la lectura. El objeto *Titanium.App.Properties* está disponible tanto para iOS como para Android.

4.4.3.2 Filesystem

El objeto *Titanium.Filesystem* es el encargado de realizar operaciones sobre el sistema de fichero del dispositivo. Su uso está recomendado para determinados casos:

- La información se proporciona en un fichero.
- La información es una imagen.

Dentro del API podemos distinguir por un lado las propiedades y por otro los métodos. Las propiedades son constantes que nos indican cuales son las rutas de almacenamiento de información de nuestra aplicación. Son propiedades de solo lectura:

- **applicationDataDirectory:** Constante que especifica donde se encuentra el directorio con los datos de la aplicación dentro del dispositivo.
- **resourcesDirectory:** Constante que especifica donde se encuentra el directorio que contiene los recursos de la aplicación.
- **tempDirectory:** Constante que especifica donde se colocan los archivos temporales de la aplicación.

Respecto a los métodos, son los que nos permiten realizar las funciones básicas en un sistema de ficheros. A continuación se muestran algunos ejemplos:

- **getFile():** crea un fichero o directorio en la ruta especificada y devuelve un objeto de tipo *Titanium.Filesystem.File.*
- **deleteFile():** borra el fichero sobre el que se ejecuta.
- **exists():** devuelve trae si existe el fichero o el directorio
- **extension():** devuelve la extensión del fichero.
- **move():** mueve un fichero a otra ruta.
- **rename():** renombra el fichero.
- **nativePath():** devuelve la ruta nativa del fichero o directorio.
- **read():** lee el contenido del fichero como *blob*.
- **write():** escribe el contenido de un fichero.
- **writeable():** devuelve si el fichero tiene permiso de escritura.
- **spaceAvalaible():** devuelve si existe espacio en disco en la ruta especificada.

El objeto Titanium.Filesystem está disponible tanto para iOS como para Android.

Descripción de la prueba

Para probar la funcionalidad del API *Titanium.Filesystem*, se hizo una aplicación de prueba que hace uso de sus distintas propiedades y funciones. Esta aplicación consta de una pantalla principal con un conjunto de botones, cada uno de los cuales se encarga

de hacer una función. La prueba se llevará a cabo mediante el simulador, para poder mostrar por la consola de Titanium Studio los resultados de cada acción.

Figura 4.21: Aplicación ejemplo Filesystem

La idea es conseguir crear un directorio para la aplicación, crear un fichero, comprobar que existe el fichero, borrar el fichero, borrar el directorio, escribir información en un fichero, leer la información de un fichero.

Resultado de la prueba

El resultado de las pruebas en iOS fue satisfactorio en todos los casos. La funcionalidad de todos los botones fue la esperada. Estas propiedades sí que son accesibles desde iOS, como se muestra en la Figura 4.22.

```
[INFO] Directorio Application: /Users/Naranjito/Library/Application
Support/iPhone Simulator/4.2/Applications/0544669D-85D0-4336-B439-
F6BB4EE24240/Applications
Directorio Application Data: /Users/Naranjito/Library/Application
Support/iPhone Simulator/4.2/Applications/0544669D-85D0-4336-B439-
F6BB4EE24240/Documents
Directorio Resources:
/Users/Naranjito/Documents/Tesina/Titanium/DbFiles/Resources
Directorio temporal: /var/folders/Th/Thgt+cTJHZOOHvFCDEdQV++++TI/-Tmp-/
```

Figura 4.22: Propiedades Filesystem iOS

En cambio para Android las rutas de almacenamiento de información no se encontraban accesibles, como se observa en la Figura 4.23:

```
[INFO] [4,470792] Directorio Application: null
[INFO] /TiAPI ( 318): Directorio Application Data: appdata-private://
[INFO] /TiAPI ( 318): Directorio Resources: app://
[INFO] /TiAPI ( 318): Directorio temporal: undefined
```

Figura 4.23: Propiedades Filesystem Android

Conclusiones de la prueba

Mediante esta prueba hemos conseguido probar el objeto encargado de realizar operaciones básicas sobre el sistema de ficheros. El uso de los diferentes métodos encargados de crear directorios, ficheros, modificar archivos, borrar archivos, etc., funciona perfectamente en ambas plataformas, si exceptuamos en Android el caso de las propiedades que no muestran las rutas de almacenamiento de información.

4.4.3.3 Database

El objeto *Titanium.Database* es el encargado de gestionar el motor de base de datos soportado por Titanium. Se recomienda su uso para determinados casos:

- Hay muchos elementos de información parecidos.
- Los elementos de información se relacionan entre sí.
- La información se va acumulando en el tiempo.

Mediante el objeto *Titanium.Database* tenemos acceso al motor de base de datos soportado por Titanium; SQLite3. Este motor se caracteriza por su bajo consumo de recursos y su simple configuración, lo que le convierte en idóneo para usar en un dispositivo móvil, donde los recursos son limitados.

También existe la posibilidad de usar otros motores de base de datos, pero se recomienda usar el mecanismo soportado por Titanium a no ser que sea una exigencia de la aplicación. Podemos citar los siguientes:

- Conexión remota.
- Uso de HTTPClient.
- PHP + MySQL

A la hora de usar una base de datos en nuestra aplicación tenemos dos posibilidades. Por un lado, podemos crear una base de datos desde cero. Para ello solo tenemos que escribir el código correspondiente, que se ejecutará al empezar a usar la aplicación. La otra opción es precargar una base de datos con una estructura predefinida y con datos en su interior. Para poder usar una ya precargada el procedimiento a seguir es colocar el fichero de tipo SQLite3 en la carpeta *Resources* de nuestra aplicación. Una vez hecho esto, se copiará el archivo SQLite3 en la primera ejecución de la aplicación dentro de la carpeta propia de ésta en el teléfono. El objeto *Titanium.Database* está disponible tanto para iOS como para Android.

Descripción de la prueba

Para probar el API de bases de datos de Titanium se hizo una aplicación de prueba. El objetivo de esta aplicación era probar los aspectos básicos del uso de una base de datos: crear una base de datos, crear unas tablas y realizar operaciones básicas sobre ellas: inserciones, borrados y modificaciones. Por un lado tenemos la aplicación, que consta de una serie de botones encargados de realizar estas acciones, y por otro lado mostramos en la consola del editor el resultado de estas acciones.

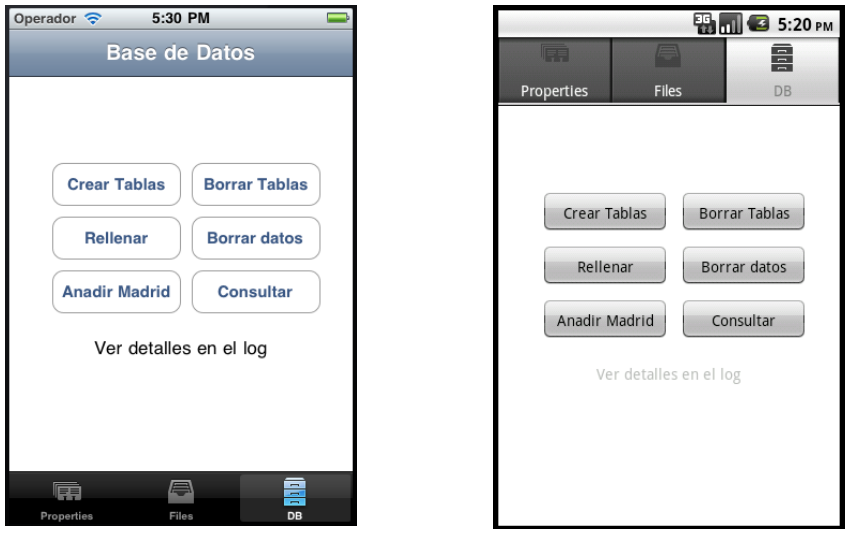


Figura 4.24: Aplicación ejemplo Database

Resultados de la prueba

Los resultados de las pruebas enfocadas a la creación y gestión de una base de datos fueron satisfactorios. La base de datos se creó correctamente, de la misma manera que la creación de tablas y su posterior inserción de datos.

Conclusiones de la prueba

En Titanium, la gestión de bases de datos se realiza de una manera sencilla gracias al objeto *Titanium.Database*. La prueba fue satisfactoria en ambas plataformas, por lo que el uso de bases de datos en Titanium está bien soportado.

4.4.3.1 Resumen datos locales

Una vez hechas distintas pruebas sobre los distintos APIs encargados de la gestión de los datos locales en un dispositivo, se pueden extraer varias conclusiones. Gracias al API *Titanium.App.Properties* podemos almacenar determinados datos como propiedades de nuestra aplicación, ya sean datos simples o datos complejos en formato JSON. Se recomienda su uso para definir información de configuración de la aplicación. Funciona tanto en iOS como en Android.

Respecto al API *Titanium.FileSystem*, es el encargado de realizar operaciones sobre el sistema de fichero del dispositivo. Podemos resaltar su importancia debido a que su uso es necesario en casi cualquier aplicación, ya sea creando un directorio, creando un fichero, comprobando el espacio disponible o la existencia de un fichero, etc. Una vez realizadas las pruebas, cabe resaltar que para Android, las constantes que devuelven las rutas de almacenamiento de información no se encontraban accesibles. El resto de pruebas fueron satisfactorias, tanto en iOS como en Android.

Por último, con el objeto *Titanium.Database* se gestiona el uso de bases de datos en Titanium. Con este API se da soporte a SQLite3, que es el motor de base de datos

que utilizaremos en aplicaciones Titanium. El uso de SQLite3 asegura la rapidez y el escaso consumo de recursos. Existe la posibilidad de usar otros motores de bases de datos pero no es lo recomendado.

4.5 Conclusiones

En este capítulo, hemos profundizado en la plataforma Titanium. En primer lugar, hemos visto su arquitectura, centrándonos en el *background* sobre el que se basa el funcionamiento de la plataforma. Conocer su funcionamiento interno es esencial, ya que nos permite entender de una manera más clara y precisa todo el proceso de desarrollo de una aplicación multiplataforma desde Titanium.

A continuación, hemos visto como es la estructura típica de un proyecto Titanium, haciendo hincapié en los archivos y carpetas generados y viendo como es el esqueleto de una aplicación. Esto nos permitirá tener una idea global de cómo están distribuidos los distintos elementos de un proyecto, y facilitará el desarrollo de una aplicación para un desarrollador inexperto en la plataforma.

Por último, hemos descrito algunas APIs importantes de cara al desarrollo de una aplicación, haciendo uso de ejemplos y analizando los resultados de las pruebas realizadas. En este apartado no se buscaba analizar todas las APIs, sino mostrar el funcionamiento de algunas de las más importantes y así mostrar el camino a seguir para el uso de otras APIs.

El objetivo de este capítulo ha sido obtener una “Guía de desarrollo” que permita a un nuevo desarrollador ver la potencia de Titanium, que puede hacer y que no y cuáles son las diferencias de funcionamiento entre iOS y Android. Este capítulo no pretende contener todo lo necesario para programar una aplicación mediante Titanium, ya que para eso ya está la extensa documentación disponible, sino extraer de ésta las características principales y guiar al programador novato sobre determinadas APIs y su viabilidad según la plataforma.

Capítulo 5

5. Aplicación SakMob

5.1 Introducción

En los capítulos anteriores hemos tratado distintos aspectos relacionados con el desarrollo multiplataforma y el siguiente paso es ponerlos en práctica. En primer lugar hemos descrito en qué consiste MOBIP como proyecto europeo y la utilización de Sakai por parte de este. También hemos visto en qué consiste la plataforma de *e-learning* Sakai, haciendo hincapié en su acceso actual mediante dispositivos móviles y razonando la elección de Titanium como plataforma elegida. Posteriormente hemos analizado los diferentes entornos multiplataforma mediante los cuales desarrollar aplicaciones móviles. Por último hemos entrado en profundidad en la plataforma Titanium, haciendo una guía de desarrollo que sirva de ayuda a nuevos desarrolladores. Llegados a este punto, la idea es crear una aplicación multiplataforma que se conecte a una instancia de Sakai asociada al proyecto MOBIP. Este capítulo se centra en la aplicación desarrollada en el marco de la Tesina.

Como hemos comentado en el apartado 2.3, Sakai es un software educativo de código abierto, que debido a su flexibilidad que permite su utilización en organizaciones con todo tipo de requisitos, desde pequeñas y medianas empresas a grandes universidades y administraciones. El software Sakai posee múltiples funcionalidades de comunicación entre profesores y alumnos que pueden ser implementadas en una aplicación móvil, como la publicación de anuncios, aviso sobre determinados eventos, gestión de trabajos, etc. El objetivo principal de SakMob es permitir a los usuarios acceder a la información más importante para poder consultarla de una forma clara y sencilla. Para ello se realizará un desarrollo mediante Titanium que permita generar una aplicación para iOS y para Android.

En los siguientes apartados veremos en qué consiste la aplicación, cual es su especificación, el desarrollo que se ha seguido y los problemas que se han encontrado, comparando el desarrollo para iOS y Android.

5.2 Especificación

En este apartado vamos a desarrollar algunos aspectos de la especificación de la aplicación. En primer lugar se van a describir los requisitos que debe cumplir la aplicación. Con el fin de describir el funcionamiento de la aplicación, el siguiente paso será especificar los distintos casos de uso. Por último, se describirán los servicios Web de Sakai, indicando su funcionamiento y estructura.

5.2.1 Requisitos del sistema

A continuación se van a describir las principales características que debe cumplir el sistema:

5.2.1.1 Autenticación

El usuario tiene que autenticarse en el sistema Sakai correspondiente. Mediante una pantalla de *login*, el usuario introducirá el id de usuario y el pin. Los roles de los usuarios serán alumno o profesor.

5.2.1.2 Información global

Dentro de la aplicación existirá una sección en la que se mostrará información global sobre distintas herramientas del usuario, las cuales no dependerán de un sitio en concreto. Estas herramientas serán los anuncios y los eventos de calendario del usuario. Se mostrará el número de elementos que hay de cada uno.

5.2.1.3 Lista de sitios

Dentro de la aplicación existirá una sección que listará los sitios en los que un usuario está dado de alta dentro de Sakai. La interfaz consistirá en una tabla, siendo seleccionables cada una de las filas que la componen. El usuario podrá navegar entre los distintos sitios en los que esté dado de alta.

5.2.1.4 Lista de herramientas

Dentro de cada uno de los sitios, el usuario podrá navegar entre distintas herramientas del sitio en cuestión. Las herramientas a mostrar, siempre que éstas estén dadas de alta, serán las siguientes:

- Anuncios.
- Eventos de calendario.
- Recursos.
- Tareas.

5.2.1.5 Visualización de recursos

Dentro de las herramientas de un sitio puede estar la herramienta “Recursos”. La aplicación deberá intentar visualizar los tipos de recursos que sean posibles para cada una de las plataformas.

5.2.1.6 Información del usuario

Dentro de la aplicación existirá una pantalla en la que se muestre información del perfil del usuario que se ha logueado.

5.2.1.7 Configuración IP de Sakai

Existencia de una pantalla de configuración desde la que especificar la IP de acceso a la instancia de Sakai. La correcta configuración de este parámetro es necesaria para el funcionamiento de la aplicación.

5.2.2 Casos de uso del sistema

A continuación se va a mostrar la especificación de los casos de uso de la aplicación. Para cada uno de ellos se va a mostrar el diagrama, la descripción del caso de uso, los actores que participan, el escenario principal y las pruebas de aceptación.

5.2.2.1 Login

Diagrama

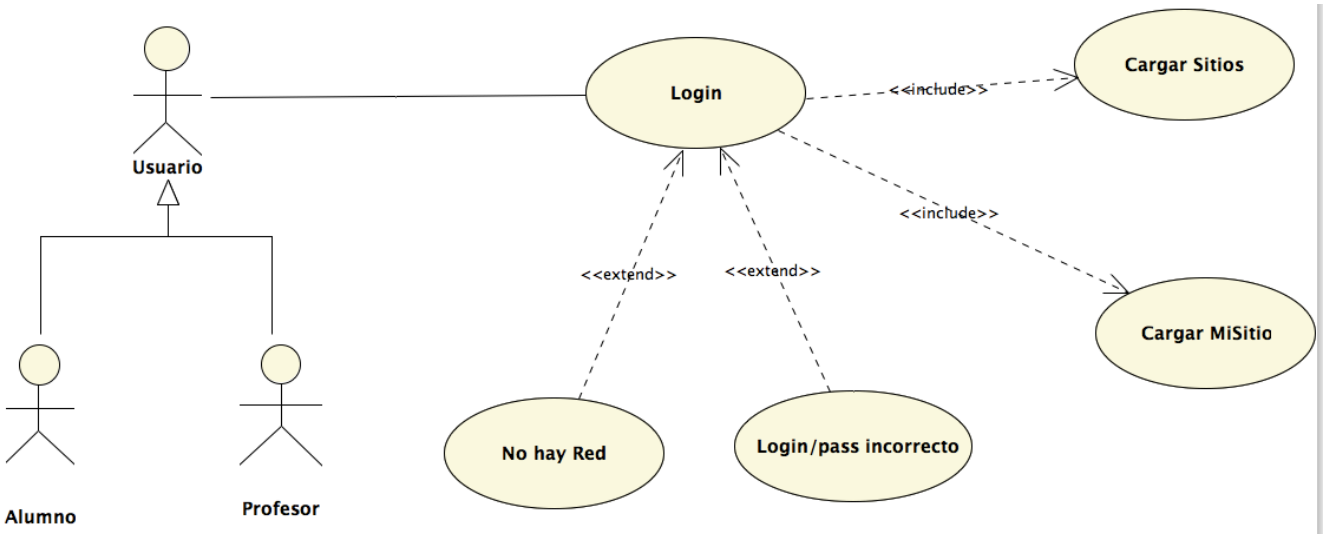


Figura 5.1: Caso de uso *login*

Descripción

Este caso de uso se encarga de la validación del usuario en el sistema.

Actores

- Alumno.
- Profesor.

Escenario principal

- Se rellenan los campos del identificador del usuario y de su pin.
- Se pulsa el botón “Entrar”.
- El usuario se valida en el sistema.

Pruebas de aceptación

- El campo del id no puede estar vacío.
- El campo del pin no puede estar vacío.
- El usuario tiene que estar validado en el sistema Sakai.
- El dispositivo tiene que estar conectado a Internet.

5.2.2.2 Logout

Diagrama

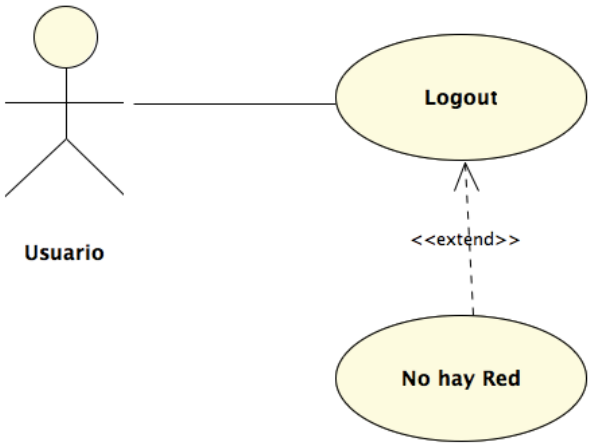


Figura 5.2: Caso de uso *logout*

Descripción

Este caso de uso se encarga de cerrar la sesión del usuario.

Actores

- Alumno.
- Profesor.

Escenario principal

- Se pulsa la opción “Salir”.
- El usuario confirma que desea salir.
- El sistema cierra la sesión.
- El sistema vuelve a la pantalla de *login*.

Pruebas de aceptación

- El dispositivo tiene que estar conectado a Internet.

5.2.2.3 Configuración IP

Diagrama

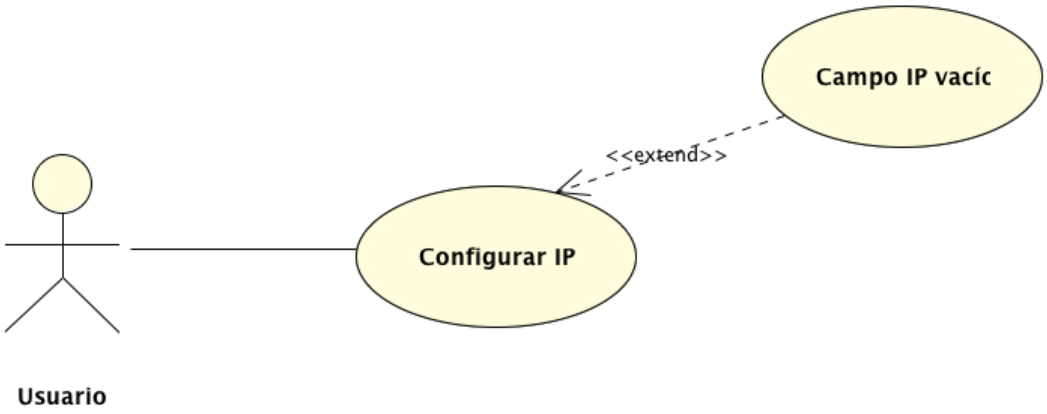


Figura 5.3: Caso de uso configurar IP

Descripción

Este caso de uso se encarga de configurar la IP de la instancia de Sakai.

Actores

- Usuario.

Escenario principal

- Se pulsa la opción "Config." desde la pantalla de *login*.
- El usuario rellena el campo "IP" con la IP de la instancia de Sakai.
- Se pulsa la opción correspondiente.
- Se actualiza el fichero de configuración con la nueva IP.
- El sistema vuelve a la pantalla de *login*.

Pruebas de aceptación

- El campo "IP" no puede estar vacío.
- El campo "IP" se debe inicializar con el valor existente en el fichero de configuración.

5.2.2.4 Carga de "Mi Sitio"

Diagrama

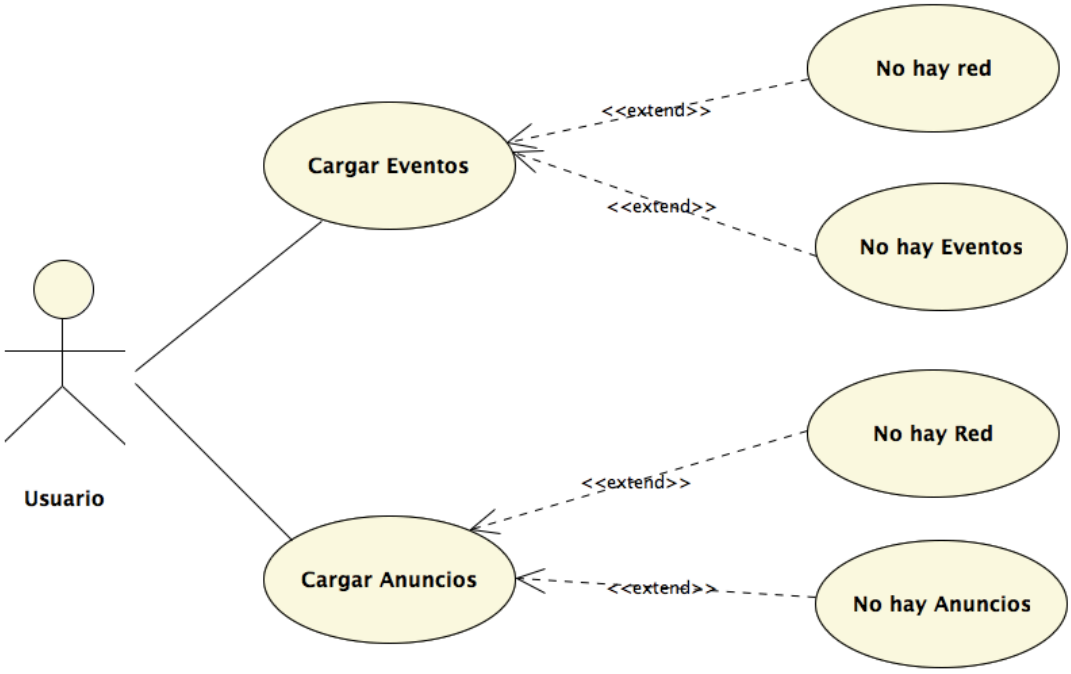


Figura 5.4: Caso de uso MiSitio

Descripción

Este caso de uso se encarga de obtener información global del usuario: los eventos de calendario y los anuncios de todos los sitios en los que está dado de alta.

Actores

- Alumno.
- Profesor.

Escenario principal

- Se comprueba si el usuario tiene sitios.
- Se recuperan el número de eventos y de anuncios.
- Se muestra la información en pantalla.

Pruebas de aceptación

- El dispositivo tiene que estar conectado a Internet.
- El usuario no está dado de alta en ningún sitio.
- El usuario no tiene anuncios.
- El usuario no tiene eventos.

5.2.2.5 Carga de "Sitios"

Diagrama

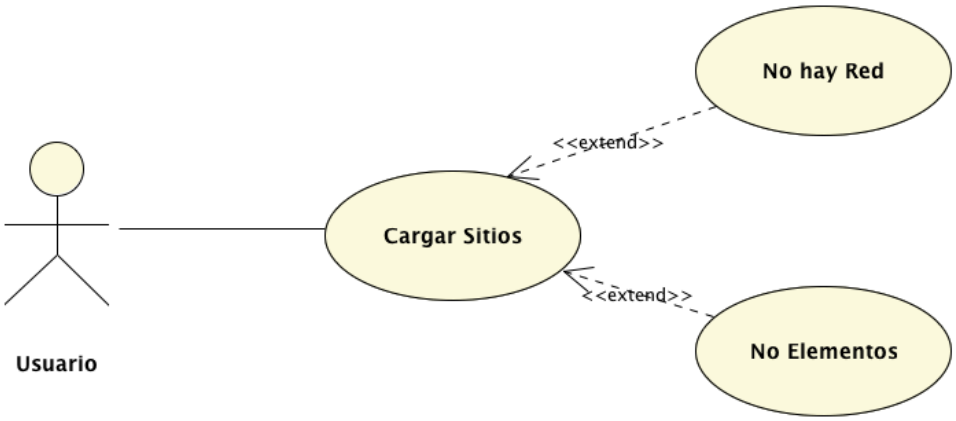


Figura 5.5: Caso de uso Sitios

Descripción

Este caso de uso se encarga de recuperar los sitios en los que el usuario está dado de alta.

Actores

- Alumno.
- Profesor.

Escenario principal

- Se recuperan los sitios en los que el usuario está dado de alta.
- Se muestra la información en pantalla.

Pruebas de aceptación

- El dispositivo tiene que estar conectado a Internet.
- El usuario no está dado de alta en ningún sitio.

5.2.2.6 Carga de un sitio

Diagrama

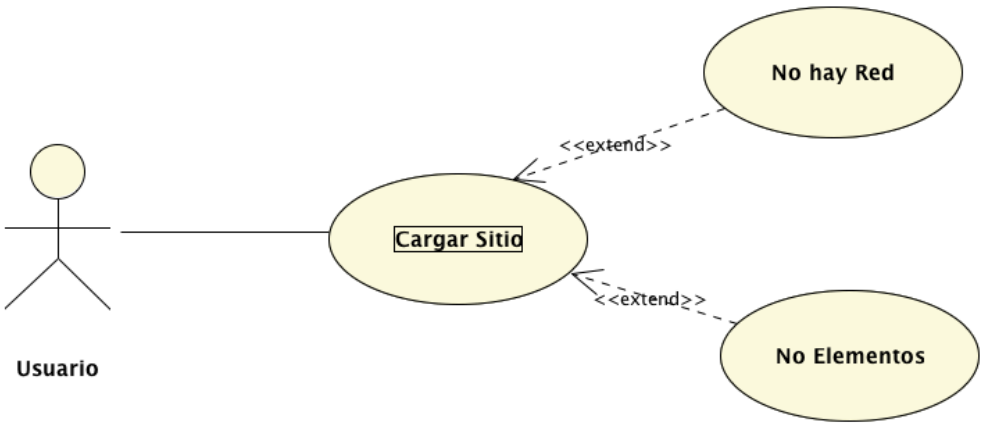


Figura 5.6: Caso de uso sitio

Descripción

Este caso de uso se encarga de obtener las distintas herramientas de las que dispone un usuario para un sitio en concreto.

Actores

- Alumno.
- Profesor.

Escenario principal

- Se recuperan las herramientas disponibles en el sitio especificado.
- Se recuperan el número elementos presentes en cada una de las herramientas.
- Se muestra la información en pantalla.

Pruebas de aceptación

- El dispositivo tiene que estar conectado a Internet.
- El usuario no tiene herramientas asignadas en ese sitio.

5.2.2.7 Carga de una herramienta

Diagrama

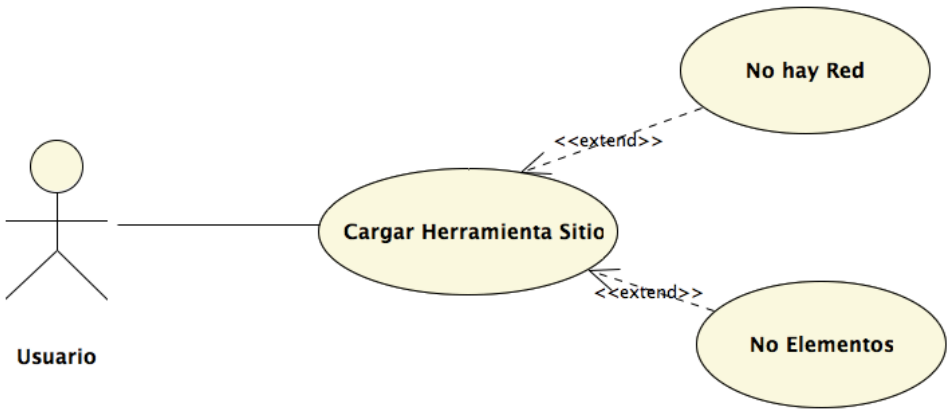


Figura 5.7: Caso de uso herramienta

Descripción

Este caso de uso se encarga de obtener el listado de elementos de una herramienta en concreto, dentro de un sitio.

Actores

- Alumno.
- Profesor.

Escenario principal

- Se recuperan los distintos elementos de una herramienta.
- Se muestra la información en pantalla.

Pruebas de aceptación

- El dispositivo tiene que estar conectado a Internet.
- El usuario no tiene elementos en esa herramienta.

5.2.2.8 Carga de un recurso

Diagrama

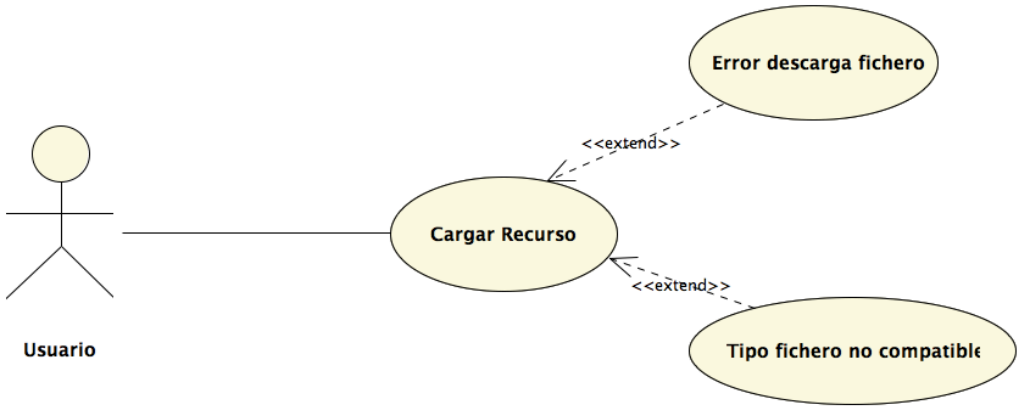


Figura 5.8: Caso de uso recurso

Descripción

Este caso de uso se encarga de la descarga/visualización de un recurso del usuario en un sitio concreto.

Actores

- Alumno.
- Profesor.

Escenario principal

- Se selecciona un recurso de la lista de recursos del sitio.
- Se descarga el recurso en el dispositivo.
- Se visualiza el recurso en pantalla si la plataforma lo permite.

Pruebas de aceptación

- El dispositivo tiene que estar conectado a Internet.
- El tipo de fichero del recurso no es compatible.
- No hay espacio de almacenamiento para el fichero a descargar.

5.2.2.9 Cargar elemento de una herramienta

Diagrama

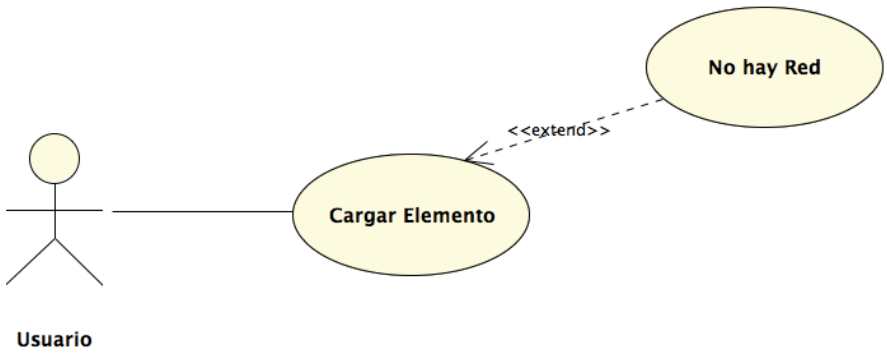


Figura 5.9: Caso de uso elemento de herramienta

Descripción

Este caso de uso se encarga de visualizar un elemento del listado de elementos de una herramienta en concreto, dentro de un sitio.

Actores

- Alumno.
- Profesor.

Escenario principal

- Se selecciona un elemento de la lista de elementos de la herramienta.
- Se muestra la información del elemento en pantalla.

Pruebas de aceptación

- El dispositivo tiene que estar conectado a Internet.

5.2.2.10 Añadir anuncio

Diagrama

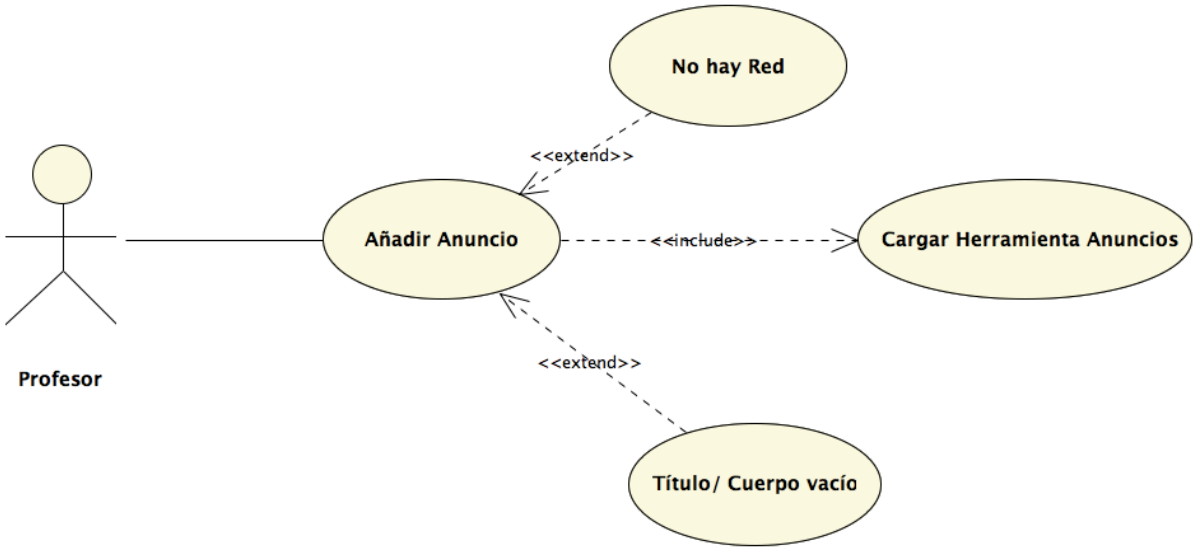


Figura 5.10: Caso de uso añadir anuncio

Descripción

Este caso de uso de añadir un anuncio dentro de un sitio concreto.

Actores

- Profesor.

Escenario principal

- Se escribe el titulo del anuncio.
- Se rellena el contenido del anuncio.
- Se pulsa el botón.
- El sistema crea el nuevo anuncio.
- El sistema vuelve a la pantalla con el listado de anuncios del sitio, mostrando el nuevo anuncio.

Pruebas de aceptación

- El dispositivo tiene que estar conectado a Internet.
- El usuario tiene permisos para poder añadir anuncios.
- El campo título no está vacío.
- El campo cuerpo no está vacío.

5.2.2.11 Cargar perfil usuario

Diagrama

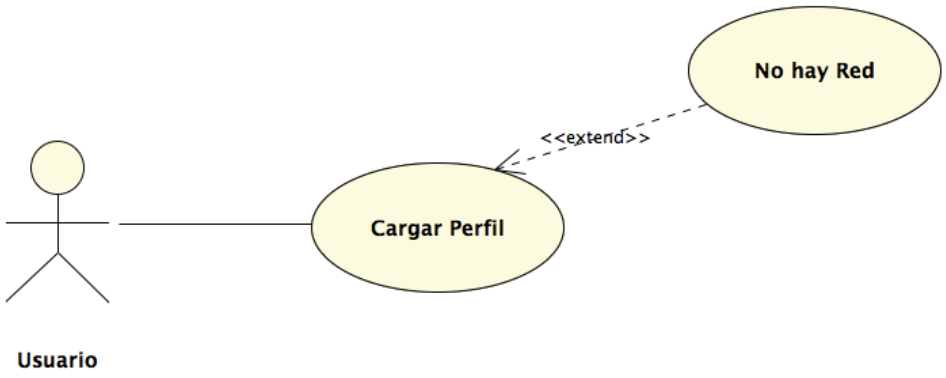


Figura 5.11: Caso de uso perfil de usuario

Descripción

Este caso de uso se encarga de obtener la información de perfil del usuario: el id dentro del sistema, el nombre y el email.

Actores

- Alumno.
- Profesor.

Escenario principal

- Se selecciona la pestaña perfil.

- Se muestra la información del usuario en pantalla.

Pruebas de aceptación

- El dispositivo tiene que estar conectado a Internet.

5.2.3 Servicios Web Sakai

SakMob es una aplicación cliente de una instancia de Sakai, la cual está desplegada en un servidor Tomcat. Como aplicación cliente, SakMob se nutre de información de esa instancia de Sakai, la cual tratará y mostrará al usuario final de la aplicación. Debido a esto, es necesario el uso de servicios Web que sean capaces de recuperar la información necesaria. La ruta en la que se encuentran desplegados es la carpeta "opt/tomcat/webapps/sakai-axis".

El siguiente paso es saber que estructura tienen estos servicios Web. Los servicios Web de Sakai son ficheros ".jws" que obtienen la información a partir de los métodos y clases del API de Sakai. Los servicios Web ya disponibles en la instancia de Sakai contienen métodos para realizar algunas de las acciones básicas (*login*, *logout*, etc.). Obviamente, los métodos ya existentes no son suficientes para recuperar toda la información necesaria para nuestra aplicación. Este hecho hace necesario el desarrollo de nuevos servicios Web que hagan esta función. Una vez creado un fichero ".jws" con los métodos correspondientes en su interior, éste se despliega en la carpeta "sakai-axis" del Tomcat. A partir de ese momento, ese servicio Web ya está disponible para ser llamado desde la aplicación.

A continuación se detallan los servicios servicios Web desarrollados para el funcionamiento de la aplicación.

5.2.3.1 getSitesAndRoleForUser

Servicio Web cuya finalidad es recuperar los sitios en los que está dado de alta un determinado usuario. Para cada uno de estos sitios se recuperan los permisos del usuario para poder añadir un anuncio. Se recupera la siguiente información:

- Id del sitio.
- Título del sitio
- Permisos para añadir anuncios.

Parámetros

sessionid, userid.

Respuesta

XML con la información de cada uno de los sitios.

```
<getSitesAndRoleForUserReturn xsi:type="xsd:string">
 <?xml version="1.0" encoding="UTF-8"?>
   <list>
       <item>
           <siteId>6f59679d-0f39-46b2-944e-dce63aa5a31d</siteId>
           <siteTitle>HMI</siteTitle>
           <addAnnPerm>false</addAnnPerm>
           </item>
       <item>
           <siteId>05bcf0e4-d01a-4631-94b5-c939c05fc5d6</siteId>
           <siteTitle>TIR</siteTitle>
           <addAnnPerm>false</addAnnPerm>
       </item>
   </list>
</getSitesAndRoleForUserReturn>
```

Figura 5.12: Respuesta *getSitesAndRoleForUser*

5.2.3.2 getUserInfo

Servicio Web cuya finalidad es recuperar la información básica del usuario de la sesión actual.

- Id interno del usuario.
- Id del usuario.
- Nombre del usuario.
- Email del usuario.

Parámetros

sessionid.

Respuesta

XML con la información de cada uno de los eventos de calendario del sitio.

```
<getUserInfoReturn xsi:type="xsd:string">
 <?xml version="1.0" encoding="UTF-8"?>
   <list>
       <item>
           <userId>bb3f74ab-7449-42ab-a566-0a8b55c8194f</userId>
           <displayId>jnaranjo</displayId>
           <displayName>Javier Naranjo</displayName>
           <userEmail>javinaranjo@test.com</userEmail>
       </item>
   </list>
</getUserInfoReturn>
```

Figura 5.13: Respuesta *getUserInfo*

5.2.3.3 getToolsForSite

Servicio Web cuya finalidad es recuperar las herramientas de un sitio en el que está dado de alta un determinado usuario. Para cada una de estas herramientas se recupera el número de elementos que contienen. Se recupera la siguiente información:

- Id de la herramienta.
- Título de la herramienta.
- Número de elementos de cada herramienta.

Parámetros

sessionid, userid.

Respuesta

XML con la información de cada uno de las tareas del sitio.

```
<getToolsForSiteReturn xsi:type="xsd:string">
 <?xml version="1.0" encoding="UTF-8"?>
   <site id="6f59679d-0f39-46b2-944e-dce63aa5a31d">
       <tool id="c8a0afb8-6265-46f1-b47f-a25c7fce9226">
           <tool-id>sakai.schedule</tool-id>
           <tool-title>Calendario</tool-title>
           <tool-num>1</tool-num>
       </tool>
       <tool id="6d19f747-ac99-4537-84c0-e26e7ca9ccf6">
           <tool-id>sakai.announcements</tool-id>
           <tool-title>Anuncios</tool-title>
           <tool-num>3</tool-num>
       </tool>
       <tool id="0680a34a-7bdf-4b71-b735-ebbe10a2bef2">
           <tool-id>sakai.resources</tool-id>
           <tool-title>Recursos</tool-title>
           <tool-num>3</tool-num>
       </tool>
       <tool id="62353b08-1ec4-4f12-81f2-b292703d8420">
           <tool-id>sakai.assignment.grades</tool-id>
           <tool-title>Tareas</tool-title>
           <tool-num>1</tool-num>
       </tool>
   </site>
</getToolsForSiteReturn>
```

Figura 5.14: Respuesta *getToolsForSite*

5.2.3.4 getAnnouncementsForSite

Servicio Web cuya finalidad es recuperar los anuncios de un sitio en el que está dado de alta un determinado usuario. Se recupera la siguiente información:

- Id del anuncio.
- Título del anuncio.
- Cuerpo del anuncio.
- Fecha del anuncio.

- Emisor del anuncio.

Parámetros

sessionid, siteid.

Respuesta

XML con la información de cada uno de los anuncios del sitio.

```
<getAnnouncementsForSiteReturn xsi:type="xsd:string">
 <?xml version="1.0" encoding="UTF-8"?>
   <list>
       <item>
           <messageId>cc78181d-4091-4a39-97fd-9200a0827786</messageId>
           <messageTitle>Inicio curso</messageTitle>
           <messageBody>El curso comenzará el próximo dia 20 de Abril</messageBody>
           <messageDate>09-abr-2011 9:29</messageDate>
           <messageFrom>Patricio Letelier</messageFrom>
       </item>
       <item>
           <messageId>ad2fd243-20e3-4845-a19b-9f5e4cc2ea50</messageId>
           <messageTitle>Mensaje prueba desde iPhone</messageTitle>
           <messageBody>Este es un mensaje creado desde el iPhone.</messageBody>
           <messageDate>08-may-2011 19:47</messageDate>
           <messageFrom>Patricio Letelier</messageFrom>
       </item>
   </list>
</getAnnouncementsForSiteReturn>
```

Figura 5.15: Respuesta *getAnnouncementsForSite*

5.2.3.5 getResourcesForSite

Servicio Web cuya finalidad es recuperar los recursos de un sitio en el que está dado de alta un determinado usuario. Se recupera la siguiente información:

- Id del recurso.
- Nombre del recurso.
- Tipo de recurso.
- Url del archivo.

Parámetros

sessionid, siteid.

Respuesta

XML con la información de cada uno de los recursos del sitio.

```
<getResourcesForSiteReturn xsi:type="xsd:string">
 <?xml version="1.0" encoding="UTF-8" ?>
   <list>
       <resource>
           <id>/group/6f59679d-0f39-46b2-944e-dce63aa5a31d/HMI - Introduccion a RUP.doc</id>
           <name>HMI - Introducción a RUP.doc</name>
           <type>file</type>
           <url>http://localhost/access/content/group/
               6f59679d-0f39-46b2-944e-dce63aa5a31d/HMI%20-%20Introduccion%20a%20RUP.doc</url>
       </resource>
       <resource>
           <id>/group/6f59679d-0f39-46b2-944e-dce63aa5a31d/video.mp4</id>
           <name>videoHMI.mp4</name>
           <type>file</type>
           <url>http://localhost/access/content/group/
               6f59679d-0f39-46b2-944e-dce63aa5a31d/video.mp4</url>
       </resource>
   </list>
</getResourcesForSiteReturn>
```

Figura 5.16: Respuesta *getResourcesForSite*

5.2.3.6 getAssignementsForSite

Servicio Web cuya finalidad es recuperar las tareas de un sitio en el que está dado de alta un determinado usuario. De cada tarea se muestra:

- Id de la tarea.
- Título de la tarea.
- Instrucciones de la tarea.
- Fecha de inicio de la tarea.
- Fecha de fin de la tarea.

Parámetros

sessionid, siteid.

Respuesta

XML con la información de cada uno de las tareas del sitio.

```
<getAssignmentsForSiteReturn xsi:type="xsd:string">
 <?xml version="1.0" encoding="UTF-8"?>
   <list>
       <item>
           <assignmentId>05c96ae1-17cf-41a6-9b4f-cff27fdd1a38</assignmentId>
           <assignmentTitle>Tarea prueba</assignmentTitle>
           <assignmentInstructions>&amp;nbsp;Estas son las instrucciones de la tarea.
               Esta es una tarea de prueba. &amp;nbsp;Esta es una tarea de prueba.
               &amp;nbsp;Esta es una tarea de prueba.</assignmentInstructions>
           <assignmentReleaseDate>26-abr-2011 10:00</assignmentReleaseDate>
           <assignmentCloseDate>13-jun-2011 20:00</assignmentCloseDate>
       </item>
   </list>
</getAssignmentsForSiteReturn>
```

Figura 5.17: Respuesta *getAssignmentsForSite*

5.2.3.7 getCalendarEventsForSite

Servicio Web cuya finalidad es recuperar los eventos de calendario de un sitio en el que está dado de alta un determinado usuario.

- Id del evento.
- Id del creador del evento.
- Nombre del evento.
- Descripción del evento.
- Rango de fechas en que el evento tiene lugar.
- Lugar del evento.

Parámetros

sessionid, siteid.

Respuesta

XML con la información de cada uno de los eventos de calendario del sitio.

```
<getCalendarEventsForSiteReturn xsi:type="xsd:string">
 <?xml version="1.0" encoding="UTF-8"?>
   <list>
       <item>
           <id>ecd9b480-05fe-40c2-a2dd-f983505eea05</id>
           <creator>66400f91-18fb-495a-a4be-52dbb251485b</creator>
           <nom>Examen HMI 1</nom>
           <descrip> Evento de calendario para el examen HMI 1</descrip>
           <tipo>Exam</tipo>
           <range>13-jun-2011 8:00 ] 13-jun-2011 9:00</range>
           <location>Aula HMI</location>
       </item>
   </list>
</getCalendarEventsForSiteReturn>
```

Figura 5.18: Respuesta *getCalendarEventsForSite*

5.3 Estructuración del código

En este apartado comentaremos a grandes rasgos de que manera hemos diseñado la estructura del código. Para ello describiremos la estructura del proyecto y comentaremos algunos de los elementos que hemos utilizado para facilitar el desarrollo de la aplicación, ya sea con el objetivo de reutilizar código, hacerlo más entendible o facilitando la llamada a servicios Web.

5.3.1 Estructura del proyecto

Como hemos visto en el apartado 4.3.1, cualquier proyecto Titanium tiene una estructura básica común. Un proyecto consta de un conjunto de ficheros y carpetas, muchos de los cuales se generan automáticamente cuando se crea el proyecto.

La carpeta *Resources* es la carpeta más importante, ya que contiene todos los elementos que hemos desarrollado y que dotan de funcionalidad a la aplicación. Además del fichero *app.js,* que es el punto de entrada en la aplicación, tenemos dos carpetas en las que hemos distribuido los ficheros JavaScript con el código desarrollado: *main_windows* y *assets*. En la Figura 5.19 se muestra la estructura.

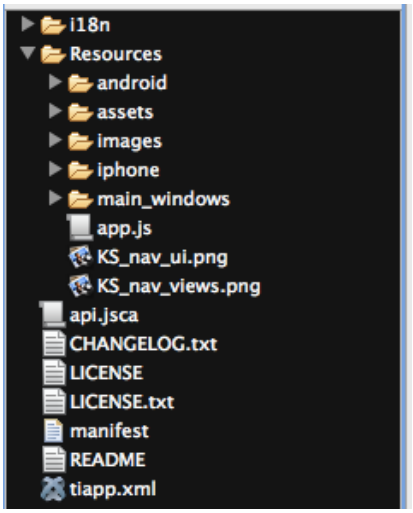
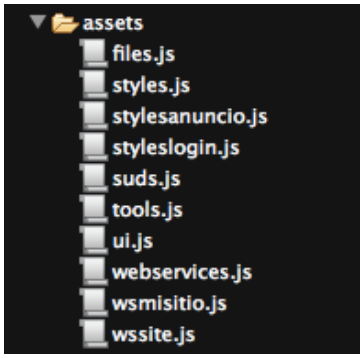


Figura 5.19: Estructura proyecto

En la carpeta *main_windows* se encuentran los distintos ficheros que contienen la logica de cada una de las ventanas de la aplicación. En cada uno de estos ficheros se diseña la interfaz de cada ventana y se recupera la información a mostrar en cada una de ellas.

En la carpeta *assets* se encuentran los distintos ficheros de herramientas y de estilo que se utilizan en la aplicación. En este capítulo analizaremos estos tipos de ficheros, detallando su función y características. Entre ellos podemos distinguir:

- Ficheros de estilo.
- Ficheros de herramientas.
- Ficheros de llamada a servicios Web.



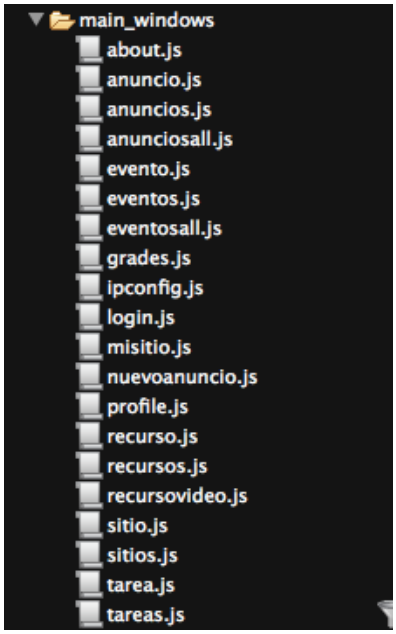


Figura 5.20: Carpetas *assets* y *main_windows*

Dentro de la carpeta *Resources* también nos encontramos con las carpetas *iphone* y *android*. Estas subcarpetas se utilizan para colocar elementos específicos de iOS y Android. Para el caso de SakMob se diseñó un icono y un *splash screen*, que se muestran en la Figura 5.21.

Figura 5.21: Icono y *Splash Screen*

Para que estos se visualicen correctamente en los distintos dispositivos y sus distintas resoluciones, se crean distintas versiones del icono y del *splash screen*:

- **iPhone:** Se crea un fichero con el nombre *appicon.png* para el icono y *Default.png* para el *splash screen*. Para crear una imagen distinta compatible con la resolución de la pantalla Retina del iPhone 4, se añade un @2x al nombre del fichero.
- **Android:** Se crea un fichero con el nombre *appicon.png* para el icono y *default.png* para el *splash screen*. Para dar soporte a las distintas resoluciones existentes en los dispositivos Android, existe una carpeta para cada una de ellas, como se muestra en la Figura 5.22. Las imágenes que se coloquen en estas carpetas serán utilizadas en los dispositivos de su correspondiente resolución.

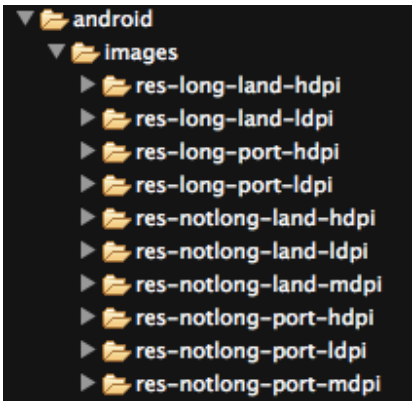


Figura 5.22: Carpeta imagenes Android

5.3.2 app.js

Como hemos comentado en el apartado 4.3.2, el archivo *app.js* es el punto de entrada en la aplicación, además de que podría contener toda la lógica de la aplicación.

Desde aquí definiremos distintos aspectos de la aplicación:

- Creación de la ventana inicial de la aplicación.
- Creación del *tabGroup* y de las pestañas que la componen.
- Definición de distintas propiedades de la aplicación, como la ip de la instancia de Sakai.
- Definición de eventos globales.

5.3.3 Ficheros de estilo

Con el fin de facilitar la creación de los distintos componentes gráficos, se han definido una serie de ficheros de estilo. Al diseñar los distintos componentes gráficos en Titanium, ya sean vistas, ventanas o filas de una tabla, se especifican una serie de propiedades para cada uno de ellos: distancia con el componente superior, anchura, altura, titulo, etc. El agrupamiento de este conjunto de propiedades de los distintos componentes tiene como consecuencia un mejor mantenimiento del código, además de producir un código más limpio. Dentro del fichero se crea un array de objetos, cada uno de los cuales se compone por un identificador y un conjunto de propiedades, tal y como se muestra en la Figura 5.23.

```
var stylelogin = {
   "titulo": {
       top: 15,
       text: L('login_title'),
       height:'auto',
       width:'auto',
       textAlign:'center',
       color:'#707070',
       font:{fontSize:18, fontWeight:'bold'}
   },
   "labeldni": {
       top:5,
       text: 'ID',
       color:'#707070',
       width: 40,
       height: 'auto'
   }
}
```

Figura 5.23: Fichero de estilo

Para acceder a estas propiedades lo primero que hay que hacer es importar la ruta del fichero de estilos al inicio del fichero donde vayamos a utilizarlo. Una vez hecho esto, la invocación del estilo se realiza a través del nombre del array y de la propiedad, como se muestra en la Figura 5.24.

```
Ti.include("../assets/styleslogin.js");

var titLabel = Titanium.UI.createLabel(stylelogin.titulo);
```

Figura 5.24: Uso de fichero de estilo

5.3.4 Fichero de herramientas

Siguiendo una lógica parecida a la seguida en los ficheros de estilo, se ha definido un fichero de herramientas. En el agrupamos distintos métodos cuya funcionalidad es requerida en distintos puntos de la aplicación. La funcionalidad que aportan estos métodos es muy diversa. La estructura del fichero sigue el mismo patrón que en el fichero de estilo: se crea un array de objetos, cada uno de los cuales se compone por un identificador y una función JavaScript.

A continuación se describen algunos de estos métodos:

- **iconoRecurso:** Método que recupera la ruta de un icono en base a una extensión.
- **urlTool:** Método que obtiene la ruta de un fichero JavaScript en base al id de una herramienta de Sakai.
- **iconoTool:** Método que recupera la ruta del icono de una herramienta de Sakai en base al id. Se muestra en la Figura 5.25.
- **crearDirUsuario:** Método que crea un directorio de usuario dentro la carpeta de la aplicación creada en el almacenamiento externo del dispositivo.

```
var tool = {
   iconoTool: function(string){
       switch (string) {
           case "sakai.assignment.grades":
               return '../images/tareas.png';
           case "sakai.announcements":
               return '../images/anuncios.png';
           case "sakai.schedule":
               return '../images/calendar.png';
           case "sakai.resources":
               return '../images/recursos.png';
           default:
               return '../images/sitio.gif';
       }
   }
}
```

Figura 5.25: Fichero de herramientas

Para acceder a estos métodos la lógica es la misma que para invocar los estilos: primero se importa la ruta del fichero de herramientas y después se invoca el método mediante el nombre del array y del método, como se muestra en la Figura 5.26.

```
var imagenTool = Titanium.UI.createImageView({
    image: tool.iconoTool(toolid),
    width: 16,
    height: 16,
    left: 10
});
```

Figura 5.26: Uso de fichero de herramientas

5.3.5 Eventos globales

Un elemento muy importante de cara al desarrollo de nuestra aplicación es el uso de eventos globales. El uso de este tipo de eventos es muy importante porque nos permite cambiar de contexto durante la ejecución de la aplicación.

El proceso se divide en dos partes:

- En una parte de la aplicación se lanza un evento mediante el objeto *Titanium.App.FireEvent*, especificando el nombre del evento y sus propiedades. El nombre del evento tiene que ser único, para no generar conflictos.
- En otra parte de la aplicación se define un *listener,* el cual contendrá la lógica asociada al evento. Se utiliza el objeto *Titanium.App.AddEventListener*, especificando el nombre del evento al que está asociado.

El uso de estos eventos se realiza en toda la aplicación. Para ilustrar el funcionamiento vamos a describir el ejemplo de la creación de un nuevo anuncio, cuyo caso de uso está descrito en el apartado 5.2.2.10. En la creación del botón que inicia el proceso de añadir un nuevo anuncio, se define el lanzamiento del evento newAnuncio en el momento en el que el usuario pulse el botón, como se muestra en la Figura 5.27.

```
createBotonAnadir: function(/*Object*/_args){
    var anadir = Titanium.UI.createButton({
        title: L('create'),
        enabled: false
    });
    anadir.addEventListener('click', function(){
        Ti.App.fireEvent('newAnuncio', {});
    });

    return anadir;
}
```

Figura 5.27: Lanzamiento de evento global

El *listener* de *newAnuncio* capturará el evento y la ejecución de la aplicación seguirá su camino. De esta manera podemos ir siguiendo un flujo de acciones sobre las distintas clases en las que tengamos que realizar alguna acción.

```
Ti.App.addEventListener('newAnuncio', function(event){
   Ti.API.info("Apreto boton crear");
   if(Ti.Network.online) {
       anadirAnuncio();
   }
   else {
       ui.comun.mostrarSinConexion();
   }
});
```

Figura 5.28: *Listener* de evento global

5.3.6 Ficheros de llamada a servicios Web

Como ya hemos comentado en el apartado 5.2.3, la aplicación hace uso de servicios Web que sirven para comunicarse con la instancia de Sakai. De una manera similar a como se hace en los ficheros de herramientas, se define un array de objetos, cada uno de los cuales se compone por un identificador y un función JavaScript. Uno de los objetos de este array contendrá la información para poder acceder a los servicios Web, como se muestra en la Figura 5.29. La IP de la instancia de Sakai está definida como una propiedad de la aplicación.

```
var ip = Titanium.App.Properties.getString("ip");
var ws = {
   param: {
       "sakailogin": {
           endpoint: 'http://'+ip+':8080/sakai-axis/SakaiLogin.jws',
           targetNamespace: 'http://'+ip+':8080/sakai-axis/SakaiLogin.jws'
       },
       "sakaisakmob": {
           endpoint: 'http://'+ip+':8080/sakai-axis/SakaiSakMob.jws',
           targetNamespace: 'http://'+ip+':8080/sakai-axis/SakaiSakMob.jws'
       }
   },
```

Figura 5.29: Parámetros llamada a servicios Web

Cada uno de los métodos se encarga de hacer una petición a Sakai y recuperar la respuesta de éste. Para ello se hace uso de la librería *suds*, la cual hemos visto en el apartado 4.4.2.3.

La estructura que siguen cada uno de los métodos de llamadas a servicios Web es el siguiente:

- Se crea un objeto *suds,* usando alguno de los parámetros de conexión definidos, como los que se muestran en la Figura 5.29.
- Se recogen los parámetros del método en el que nos encontramos para usarlos como parámetros del objeto *suds*. Por ejemplo, si se trata del método *loginUser* y hemos pasado como parámetros id y pin, éstos se usarán en *suds* para invocar al servicio Web.
- Se invoca el servicio Web correspondiente.
- Si obtenemos correctamente una respuesta del servicio Web, llamamos al evento global asociado, que se encontrará en la clase correspondiente.

En la Figura 5.30 se muestra como ejemplo parte del método encargado de hacer la petición de login a la instancia de Sakai.

```
loginUser: function(/*Object*/_args){
    var loading = ui.comun.crearLoading();
    loading.show(L('login_user')+'...');
    Titanium.App.Properties.setString("session_id", null);
    var callparams = {
        id: _args.id,
        pw: _args.pw
    };
    var suds = new SudsClient(ws.param.sakailogin);
    try {
        suds.invoke('login', callparams, function(xmlDoc){
            // Tratamiento de la respuesta.
            // Si se recibe una respuesta correcta
            // se lanza el evento correspodiente
            loading.hide();
        });
    }
    catch (e) {
        Ti.API.error('Error: ' + e);
    }
}
```

Figura 5.30: Método llamada a servicio Web

5.3.7 Internacionalización

Un punto importante a tener en cuenta en el desarrollo de una aplicación es el tema de la internacionalización. Titanium facilita el hecho de poder definir los textos de nuestra aplicación en distintos idiomas [17], como hemos comentado en el apartado 4.3.1.

Preparar nuestra aplicación para ser multiidioma es sencillo. El primer paso es crear una carpeta con el nombre *i18n* en el directorio raíz de nuestro proyecto. Una vez sabemos los idiomas a los que queremos dar soporte, creamos sus correspondientes carpetas, nombrando cada una de ellas con la abreviatura del idioma según el ISO 639-1. En la carpeta de cada idioma debe colocarse un fichero XML llamado strings.xml.

El fichero se estructura de la siguiente manera:

- **Clave:** La clave que identifica cada string es el atributo *name* de cada nodo.
- **Valor:** El valor es el texto que se encuentra dentro de cada nodo.

```
<?xml version="1.0" encoding="UTF-8"?>
<resources>
    <string name="sites">Sitios</string>
    <string name="refresh">Actualizar</string>
    <string name="ok">Aceptar</string>
    <string name="cancel">Cancelar</string>
    <string name="create">Crear</string>
    <string name="announc_new">Nuevo anuncio</string>
```

Figura 5.31: Fichero *strings.xml*

Una vez hemos definido un archivo strings.xml para cada uno de los idiomas, ya podemos hacer uso de ellos en la aplicación. La forma más usual de recuperar los textos es haciendo uso de la macro L(), tal y como se muestra en la Figura 5.32.

```
var dialogo = Titanium.UI.createOptionDialog({
    options: [L('ok'), L('cancel')],
    cancel: 1,
    title: L('logout_conf')
});
```

Figura 5.32: Uso del fichero *strings.xml*

5.4 Desarrollo y problemas asociados

Este apartado se centra en describir de una manera más detallada como ha sido el desarrollo de la aplicación. Centraremos el foco en el desarrollo de algunos casos de uso y detallaremos los distintos problemas que nos hemos encontrado, así como las diferencias entre el desarrollo para iOS y Android.

5.4.1 Aspectos a analizar

Para el desarrollo de cada una de los casos de uso nos vamos a centrar en una serie de aspectos generales de cada uno de ellos:

- **Interfaz:** En este apartado se explica que tipo de interfaz se ha diseñado para el correspondiente caso de uso, describiendo los distintos componentes gráficos y su distribución. Se adjuntan capturas de las distintas interfaces.
- **Funcionamiento:** En este apartado se describen las distintas acciones que el usuario realiza dentro de la funcionalidad asociada al caso de uso.
- **Desarrollo:** El desarrollo de un caso de uso implica la toma de determinadas decisiones de diseño y la escritura del correspondiente código. En este apartado describiremos estos aspectos detallando las APIs utilizadas.
- **Problemas:** Aquí se describen los distintos problemas encontrados durante el desarrollo de la funcionalidad de la aplicación.
- **iOS vs Android:** En este apartado se detallan las diferencias desarrolladas en el código para que la aplicación funcione correctamente en ambas plataformas.

5.4.2 Login

El punto de entrada en la aplicación es la pantalla de *login*. Desde esta pantalla se validará el usuario en el sistema Sakai. Una vez validado, accederá a su propia información.

5.4.2.1 Interfaz

Los componentes gráficos utilizados conforman una pantalla de *login* genérica.

Está compuesta por un cuadro de texto para el identificador del usuario y otro para el pin, además de un botón Entrar. Este botón inicia el proceso de entrada del usuario en la aplicación.

La pantalla de *login* tiene asociados una serie de diálogos de información, que se mostrarán en los siguientes casos en los casos en los que no haya red o se haya introducido el id o el pin de manera incorrecta.

Figura 5.33: Pantalla de *login*

5.4.2.2 Funcionamiento

Para que el usuario haga login en la aplicación, se realiza una petición a un servicio Web ya existentes en la instancia de Sakai. Los parámetros necesarios son el id y el pin, los cuales han sido introducidos por el usuario en sus respectivos cuadros de texto. Una vez enviada la petición, el servicio Web retorna como resultado un *sessionid*. Es importante resaltar la importancia de este dato, ya que será necesario para recuperar distinta información mientras estemos logueados en la aplicación. Si la identificación es correcta, la pantalla modal de *login* se cierra y se muestra la pantalla principal con la información correspondiente al usuario.

5.4.2.3 Desarrollo

Para diseñar la entrada en la aplicación mediante una pantalla de *login*, la opción elegida fue la utilización de una pantalla modal. Esto implica que la pantalla se abrirá sin estar asociado a ninguna pestaña. El método encargado de realizar la validación en Sakai se encuentra dentro del fichero de llamadas a servicios Web y se llama *loginUser*. Si la identificación es correcta, el servicio Web retornará un *sessionid*, el cual se almacenará en una propiedad de la aplicación para poder utilizarla en las llamadas a otros servicios Web. Dentro del método *loginUser* lanzaremos el evento *grantService*, tal y como hemos explicado en el apartado 5.3.5.

Desde el *listener* del evento cerraremos la pantalla de login y se crearán los componentes principales de la aplicación:

- El *tabGroup* que alojará las pestañas de la aplicación.
- Las pestañas, en las que se definen las ventanas que componen cada una de ellas.
- Las ventanas principales de la aplicación.
- El directorio de usuario, en el que se guardarán ficheros del usuario que se ha validado si así se requiere.

5.4.2.4 Problemas

Como hemos comentado en el apartado 4.3, el fichero de entrada en la aplicación es *app.js*. Una de las posibilidades contempladas al principio fue crear la ventana de login e incluirla en un *tabGroup*. Una vez logueado el usuario, este *tabGroup* se cerraría y se crearía otro con las pestañas encargadas de mostrar la información de Sakai para el usuario en cuestión. Esta opción se desechó por los problemas que ocasionaba el cerrar un *tabGroup* para crear otro, y las complicaciones que podría provocar para la realización del *logout*.

5.4.2.5 iOS vs Android

Cuando se diseña la interfaz de una aplicación para Android hay que tener en cuenta las múltiples resoluciones de pantalla disponibles en los distintos dispositivos existentes en el mercado. Para que la pantalla de *login* se pudiera visualizar correctamente en las distintas resoluciones, se definió una vista compuesta por todos los componentes gráficos de ésta. La disposición y el tamaño de esta vista viene determinado por la resolución del dispositivo correspondiente.

5.4.3 Configuración IP

Para poder validarse en la aplicación es necesario tener configurada la IP de acceso a la instancia de Sakai. Desde la pantalla de configuración, el usuario establecerá la IP sobre la que se realizarán todas las peticiones de información de la aplicación.

5.4.3.1 Interfaz

Para establecer la IP de la instancia de Sakai, el diseño de la interfaz se afronta de manera distinta entre iOS y Android. En el caso de iOS, la estrategia seguida consiste en la creación de una ventana independiente compuesta por un cuadro de texto y dos botones, uno para aceptar y otro para cancelar. Debajo del cuadro de texto se especifica el patrón del texto a insertar por parte del usuario. En el caso de Android, se muestra un dialogo desde la misma pantalla de *login,* compuesto por un cuadro de texto y un botón "Actualizar IP".

En la Figura 5.34 se muestra la interfaz tanto de iOS como de Android.

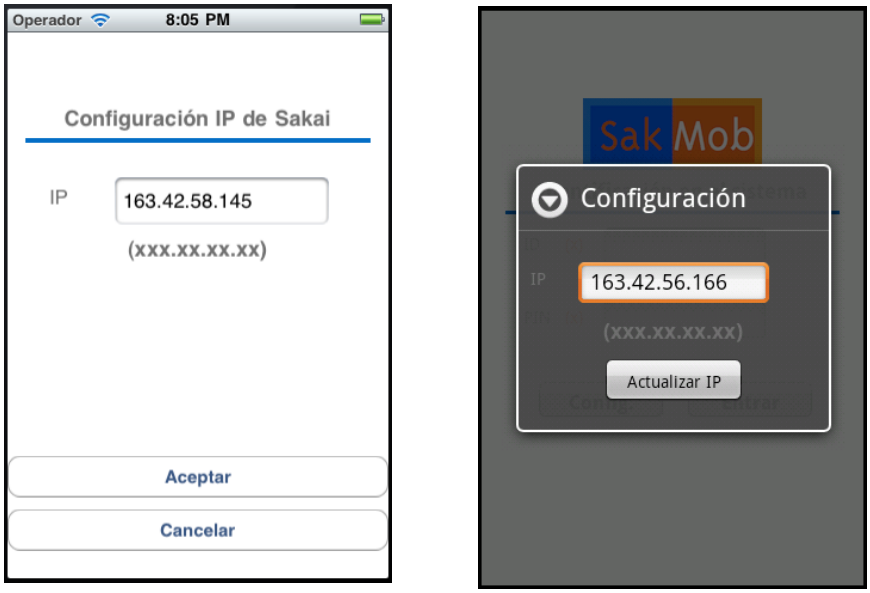


Figura 5.34: Pantalla de configuración

5.4.3.2 Funcionamiento

Dentro de la carpeta de la aplicación hay un fichero de configuración en formato JSON que contiene la información de la IP de la instancia de Titanium. El acceso a la configuración de esta información viene dado por el botón "Config." de la pantalla de *login*.

La forma de introducir la IP difiere entre las dos plataformas:

- **iOS:** En el caso de iOS se abre una nueva ventana, en la que el usuario establece la IP de la instancia de Sakai. Al pulsar "Aceptar" se actualiza la información del fichero.
- **Android**: En el caso de Android, en lugar de una nueva ventana aparece un dialogo sobre la pantalla de *login*. Después de introducir la información se pulsa el botón "Actualizar IP". Por último se pulsa el botón de navegación de Android, definido por una flecha, mediante el cual se retornará el control a la pantalla de *login*.

Una vez se ha introducido y validado la IP, se actualiza el contenido del fichero de configuración.

5.4.3.3 Desarrollo

Para el desarrollo de este punto un aspecto clave es el fichero de configuración. Cuando el usuario entra en la aplicación se comprueba si existe el fichero "config.json". Si no existe, se crea y se inicia el valor de la IP en blanco. Si existe el fichero, se recupera la información de éste y se asigna a la propiedad "ip" de la aplicación, tal y como se describe en el apartado 4.4.3.1. El cuadro de texto de la pantalla de configuración se inicializa con el valor de propiedad "ip".

Respecto a las interfaces, se han utilizado distintas APIs de Titanium. Para el caso de iOS se utiliza el API *Titanium.UI.Window* para crear una ventana independiente. En el caso de Android, se hace uso de un objeto *Titanium.UI.OptionsDialog* para crear un dialogo que se mostrará en la pantalla de *login*.

5.4.3.4 Problemas

Los problemas encontrados para dar soporte a la configuración de la IP estuvieron centrados en el diseño de la pantalla. En primer lugar, se diseñó la ventana de configuración como una pantalla independiente, la cual funcionaba perfectamente en iOS, a diferencia de lo que sucedia en Android. Los problemas residían en que la pantalla de *login* es una ventana modal, y al intentar abrir otra ventana desde ésta ocasiona problemas en Android. La solución que se encontró fue mostrar un dialogo en lugar de crear una ventana, haciendo uso de un objeto *Titanium.UI.OptionsDialog*. Al crear el dialogo se hace uso de la propiedad *androidView,* disponible solo en Android, mediante la cual se especifica el contenido del dialogo. La vista creada estará compuesta por el cuadro de texto de la IP y el botón para actualizar el fichero de configuración.

5.4.3.5 iOS vs Android

Cuando se diseña la interfaz de una aplicación con Titanium hay que tener en cuenta que determinados componentes gráficos no van a funcionar en Android y si que funcionarán en iOS, y viceversa. Como hemos comentado en el apartado anterior, se tuvieron que usar componentes gráficos distintos a la hora de diseñar la interfaz para la configuración de la IP de Sakai; abrir una pantalla desde una pantalla modal solo funciona en iOS, y especificar el contenido de un dialogo solo funciona en Android.

5.4.4 Carga de la información del usuario

Después de realizar satisfactoriamente el proceso de validación en Sakai, la aplicación muestra al usuario la pantalla principal. Esta pantalla consta de cuatro pestañas: MiSitio, Sitios, Perfil y Acerca de. A través de cada uno de estas pestañas accederemos a la información que nos pueda proporcionar Sakai.

5.4.4.1 Interfaz

La interfaz que se muestra al usuario está compuesta de diversos componentes gráficos. En la parte inferior de la pantalla se muestra el *tabGroup* con las distintas pestañas que lo componen. Las ventanas contenidas en cada una de estas cuatro pestañas se puede dividir en dos grupos: por un lado las pestañas MiSitio y Sitios y por otro las pestañas Perfil y Acerca de.

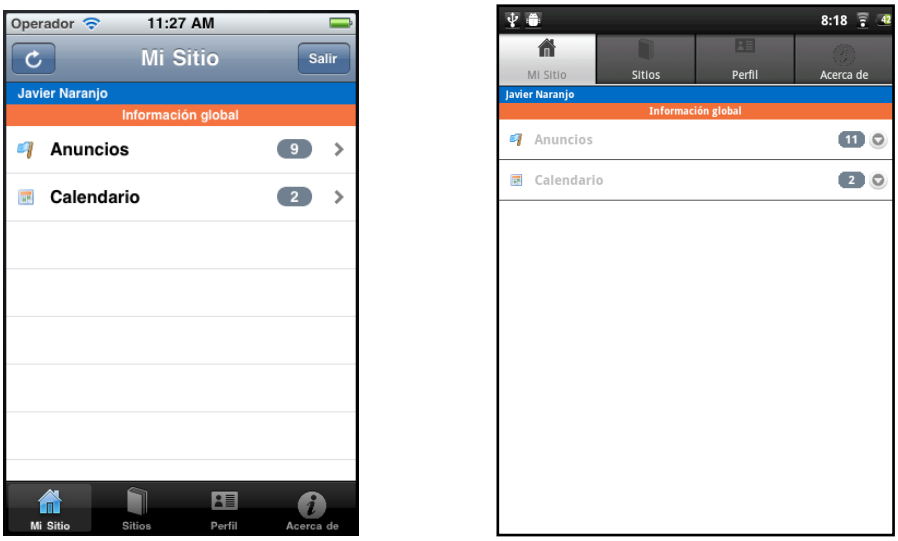


Figura 5.35: Pantalla *MiSitio*

Respecto a MiSitio y Sitios, en la parte superior aparece una banda de color azul en la que se muestra el nombre del usuario logueado. La parte central de la pantalla está compuesta por una tabla, en cuyo interior se mostrará la siguiente información:

- **MiSitio:** Esta tabla se encarga de mostrar información global del usuario respecto a los anuncios y eventos de calendario. En la parte derecha de la fila se muestran el número de elementos que contienen, como se ve en la Figura 5.35
- **Sitios:** Esta tabla muestra los sitios en los que el usuario está dado de alta, como se muestra en la Figura 5.36.

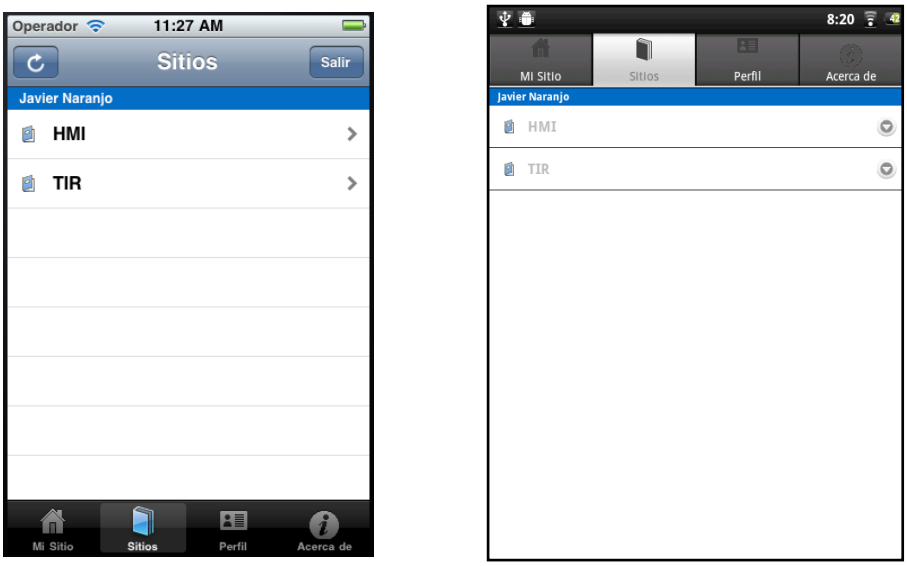


Figura 5.36: Pantalla *Sitios*

Respecto a Perfil y Acerca De, sus respectivas ventanas están compuestas por una vista que muestra una imagen y unas etiquetas con información. En el caso de la pestaña Perfil aparece una imagen del usuario, su nombre, su id y su email, mientras

que en la pestaña Acerca De se muestra el logo de la aplicación, junto a la versión de está y la información de los desarrolladores.

5.4.4.2 Funcionamiento

Cuando un usuario accede a la ventana principal después del proceso de validación, la pestaña seleccionada por defecto es MiSitio. El número de elementos que contiene cada una de las herramientas se recupera de un servicio Web creado explícitamente para esa tarea. Si se quiere consultar alguna de estas herramientas, se pulsa en la fila correspondiente y se abrirá una ventana con su información.

Por otro lado, en la pestaña Sitios se mostrarán aquellos sitios en los que el usuario está dado de alta. De la misma manera que en la pestaña MiSitio, podemos acceder a más información pulsando la fila correspondiente. En este caso se mostrarán las herramientas del sitio seleccionado. Al navegar entre los distintos sitios y herramientas sale a relucir una de las diferencias entre iOS y Android. Cuando accedemos a una determinada ventana, la forma de retornar a la ventana anterior difiere:

- **iOS:** La navegación se hace mediante el uso de los botones que van apareciendo en la parte superior izquierda de la pantalla. El texto que aparece en el botón es el nombre de la pantalla anterior.
- **Android**: La navegación se hace mediante el botón dedicado en los dispositivos Android para volver a la pantalla anterior. El botón está definido por una flecha. Dependiendo del dispositivo será un botón físico o uno integrado en la pantalla.

Respecto a la pestaña Perfil, en ella se mostrará información del usuario, como imagen del usuario, su nombre, su id y su email, la cual se recuperará de un servicio Web creado explícitamente para esa tarea.

5.4.4.3 Desarrollo

Después de que el usuario se haya logueado, desde el evento global *grantService* se abrirá el *tabGroup* que contiene las pestañas principales, en las cuales se definirán sus respectivas ventanas. En la creación de cada una de las pestañas se especifica el fichero JavaScript que contiene el código de su respectiva ventana: misitio.js, sitios.js, profile.js y about.js.

En el caso de misitio.js y sitios.js, el procedimiento seguido es similar. El flujo de acciones que se va sucediendo está basado en:

- El uso de métodos que realizan llamadas a servicios Web, cuyo funcionamiento hemos explicado en el apartado 5.3.6.
- El uso de eventos globales, cuyo funcionamiento hemos explicado en el apartado 5.3.5

Desde el fichero JavaScript de la ventana correspondiente, se invoca el método que realiza la petición al servicio Web. Una vez tenemos la respuesta, se lanza un evento que devolverá la información recogida a la ventana para que ésta la pueda

mostrar por pantalla. De esta forma se construye la interfaz de las ventanas que muestran información de Sakai.

En el caso de la pestaña Sitios, se realiza una petición al servicio *getSitesAndForUser,* cuya respuesta se muestra en 5.2.3.1. El XML de respuesta se recorre, y para cada uno de los nodos *item* se crea una fila en la tabla. Paralelamente se crea un objeto Sitio que contendrá su información correspondiente. Este objeto Sitio se usará para realizar las peticiones a los servicios Web de las distintas herramientas que componen un determinado sitio.

5.4.4.4 Problemas

Los problemas en este punto del desarrollo se centraron en encontrar el proceso óptimo mediante el cual recuperar información de Sakai y mostrarla por pantalla. En el caso de MiSitio y Sitios, la respuesta en XML del servicio Web sirve de base para poblar su respectiva tabla.

Como hemos comentado en el apartado 4.4.2, el tratamiento de la respuesta en formato XML presenta alguna dificultad. A continuación se muestra una parte del código en que se recorre el XML de respuesta del servicio Web *getSitesAndRoleForUser*, descrito en el apartado 5.2.3.1.

```
var xmlSites = Titanium.XML.parseString(dataEvento);
var items = xmlSites.documentElement.getElementsByTagName('item');
if (items != null && items.length > 0) {
    for (var i = 0; i < items.length; i++) {
        var linea = items.item(i).childNodes;
        var sitio = new Sitio(linea.item(0).text, linea.item(1).text,
        win.userid, linea.item(2).text);

        // Creación de una fila en cada pasado del bucle
        //...
        //...
    }
 }
```

Figura 5.37: Tratamiento XML *getSitesAndRoleForUser*

5.4.4.5 iOS vs Android

En este punto del desarrollo hay un par de detalles que hay que tener en cuenta para que la aplicación funcione perfectamente en ambas plataformas.

Por un lado, la diferencia existente a la hora de especificar la ruta de un fichero. Para abrir las ventanas asociadas a cada una de las filas de las tablas, se especifica la ruta del fichero correspondiente. En el caso de Android se tiene que especificar la ruta desde el directorio raíz del proyecto, mientras que en el caso de iOS desde la carpeta actual.

```
var urlSitio = (isAndroid) ? "main_windows/sitio.js" : "sitio.js";
```

Figura 5.38: Rutas ficheros iOS/Android

Por otro lado, los componentes gráficos utilizados para diseñar las opciones de Refrescar y Salir difieren entre ambas plataformas. Esto es debido a conceptos existentes dentro de sus respectivas guías de diseño. En el apartado sobre el *logout* ahondaremos en este aspecto.

5.4.5 Logout

Mientras un usuario está logueado en la aplicación, existe la posibilidad de hacer logout para poder validarnos con otro usuario. Desde la pestañas MiSitio y Sitios podemos cerrar sesión y volver a la pantalla de *login*.

5.4.5.1 Interfaz

Para seguir las guías de diseño de cada dispositivo, los elementos de interfaz usados para hacer logout difieren en iOS y Android. En iOS tenemos la barra de navegación en la parte superior, a la cual se le pueden añadir botones. En este caso se añade el botón Salir en la parte superior derecha.

En Android, existe un botón en los dispositivos que muestra un menú contextual en la pantalla en la que nos encontremos. Estas es una de las peculiaridades de Android que no tiene su correspondencia en iOS. La opción Salir se insertó en este menú, de la misma manera que otras opciones de la aplicación.

Respecto a los diálogos de confirmación, cada plataforma muestra su propio dialogo nativo.

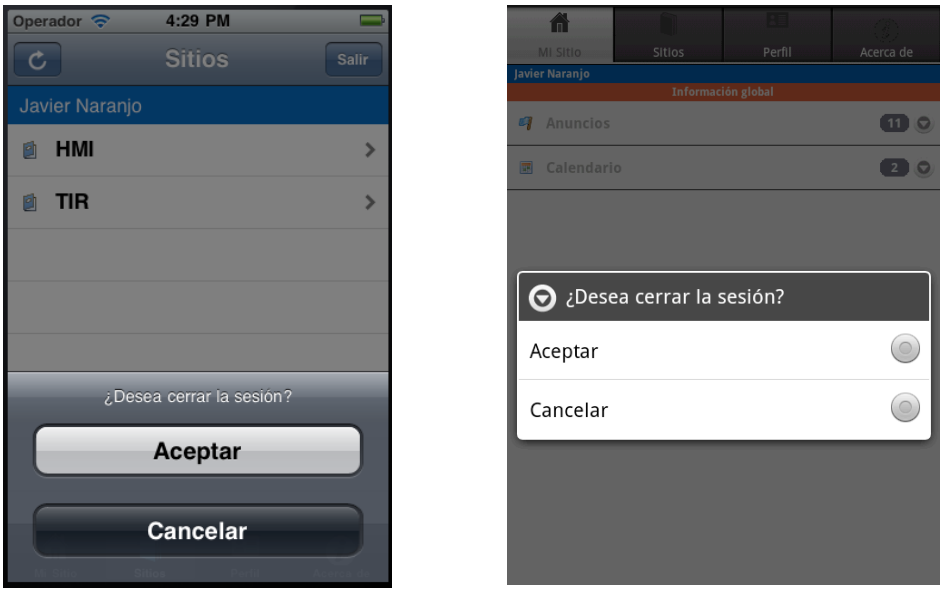


Figura 5.39: Dialogo confirmación *logout*

5.4.5.2 Funcionamiento

Para realizar el logout, el usuario tiene que estar logueado en Sakai. Desde la pestaña MiSitio y Sitios tenemos acceso a la opción *Salir*. Una vez hemos pulsado esta opción, la aplicación muestra al usuario una dialogo de confirmación. Si el usuario acepta, se realiza una petición al servicio Web encargado del *logout*. Este servicio Web es uno de los ya existentes en la instancia de Sakai. Una vez obtenemos la respuesta indicando que se ha realizado correctamente el logout, se cierran las ventanas principales de la aplicación junto con el *tabGroup* y el sistema vuelve a la pantalla de *login*.

5.4.5.3 Desarrollo

EL proceso que se desencadena cuando el usuario decide cerrar sesión es similar al proceso de login. Dentro del método *logoutUser* del fichero de llamadas a servicios Web se lanzará el evento global *logoutSession,* cuyo *listener* se encuentra en el archivo *app.js*.

Desde el *listener* de *logoutSession* lanzaremos otros eventos globales que serán capturados en cada una de las ventanas que tenemos que cerrar y que realizan las siguientes acciones:

- **closeMiSitio:** Limpia la ventana MiSitio. Para ello quita el nombre de la banda azul y vacía la tabla de elementos.
- **closeSitios:** Limpia la ventana Sitios. Para ello quita el nombre de la banda azul y vacía la tabla de elementos.
- **closeProfile:** Borra el texto de los *labels* que contienen la información de usuario.

Por último, se abre la ventana modal de *login* a la espera de que se valide de nuevo un usuario.

5.4.5.4 Problemas

Los problemas en este punto fueron varios, debido a que al cerrar sesión la aplicación se tenía que quedar en un estado que permitiera un nuevo acceso a la aplicación con otro usuario. Los primeros intentos estuvieron centrados en el cierre de las ventanas principales, con la idea de volverlas a crear cuando un usuario se volviera a validar.

Uno de los intentos fue crear un array de ventanas. La idea era que cuando el usuario hiciera *logout,* se recorrerían cada una de sus ventanas y se cerraran. Pero este intento no fue satisfactorio porque al volver a iniciar sesión con otro usuario la aplicación no funcionaba correctamente.

La solución finalmente utilizada fue la comentada en el apartado 5.4.5.3, mediante la cual las ventanas principales no se destruían, sino que se eliminaba toda la información contenida en sus componentes de interfaz, como tablas, labels , etc.

5.4.5.5 iOS vs Android

Como hemos comentado en el apartado de interfaz para el logout, difiere la manera en que se muestra la opción de *Salir* al usuario entre iOS y Android.

Para diferenciar estas dos maneras, en el código se hizo uso de la variable booleana que determina si la aplicación en ejecución es Android o no. En el caso de iOS se crea un botón y se asigna al atributo *rightNavButton* de la ventana correspondiente.

```
var salir = Titanium.UI.createButton({
    title: L('logout')
});
salir.addEventListener('click', function() {
    // Creación del dialogo de confirmación y de su listener
    //...
    //...
});
// Asignación del botón Salir
win.rightNavButton = salir;
```

Figura 5.40: Opción Salir iOS

En el caso de Android, se hace uso del atributo *activity.onCreateOptionsMenu* asignándole una función en la que se crea el menú y sus ítems correspondientes.

```
win.activity.onCreateOptionsMenu = function(e) {
    var menu = e.menu;
    m1.setIcon('../images/salir.png');
    m1.addEventListener('click', function(e) {
        // Creación del dialogo de confirmación y de su listener
        //...
        //...
    });
};
```

Figura 5.41: Opción Salir Android

5.4.6 Descarga de ficheros

Una de las funcionalidades que incluimos en SakMob es la posibilidad de visualizar los recursos a los que tiene acceso un usuario de Sakai. La idea es descargar los ficheros a una carpeta local en el teléfono para posteriormente poder abrir el fichero. Como veremos a continuación, no todos los recursos se pueden visualizar.

5.4.6.1 Interfaz

En la pantalla en la que se listan los recursos de un sitio, se muestra una barra de progreso en la parte inferior. Cuando seleccionamos un recurso de la lista, se inicia el proceso de descarga del recurso y conforme se va descargando se muestra el avance por pantalla. Una vez se ha descargado el recurso, se abre una ventana para visualizarlo. Existen dos posibilidades:

- Si el recurso es un video, se abre una ventana con el reproductor de video predeterminado del dispositivo, el cual es invocado mediante el API de Titanium.
- Si el recurso es un documento (PDF, doc, ppt, etc.), se abre una ventana compuesta por un *webView* (navegador incrustado en una vista).

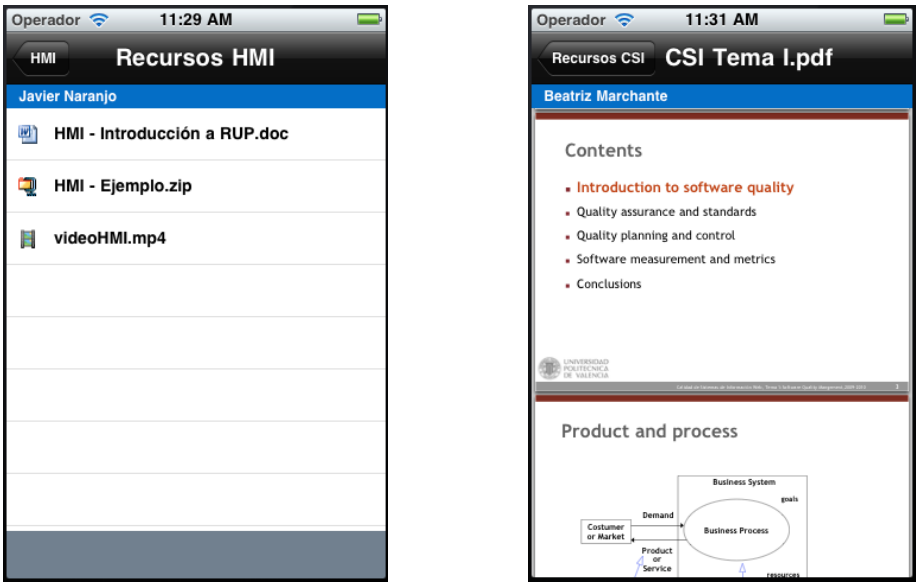


Figura 5.42: Pantallas recursos

5.4.6.2 Desarrollo

Cuando un usuario se valida en la aplicación, la idea es crear una carpeta para el usuario en el directorio de la aplicación. Para ello, se hace uso del atributo Titanium.Filesystem.applicationDataDirectory, que ya vimos en el apartado 4.4.3.2, para posteriormente crear una carpeta con el identificador del usuario. De esta manera tendríamos una carpeta propia para cada usuario.

Cuando el usuario se encuentra en la pantalla de los recursos de un sitio, al pulsar sobre un recurso éste se guardará en la citada carpeta. Para realizar la descarga se hace uso del API *Titanium.Network.createHTTPClient()*, realizando una petición GET a Sakai. Como el recurso es privado para el usuario validado, es necesario establecer una *cookie* en la conexión para poder descargarlo. La *cookie* tiene como atributo 'JSESSIONID' y como valor un string con la siguiente estructura: 'sessionid.serverid'. La parte 'sessionid' hace referencia al identificador de sesión actual del usuario, y 'serverid' al identificador del servidor Sakai, el cual se encuentra especificado en el archivo Sakai.properties. Una vez definida la *cookie* se asigna a la cabecera de la petición mediante el método *setRequestHeader* del objeto *HTTPClient*.

5.4.6.3 Problemas

La creación de una carpeta de usuario en la carpeta de la aplicación plantea problemas en Android. Como consecuencia de esto, la mejor opción es crear la carpeta donde el usuario se descargará los recursos en el almacenamiento externo del teléfono. Para ello, lo primero es comprobar si está presente este tipo de almacenamiento mediante el atributo *Ti.Filesystem.isExternalStoragePresent*. Una vez nos hemos comprobado que está presente, se obtiene la ruta mediante el método *Ti.Filesystem.getExternalStorageDirectory()* y se crea una carpeta con el identificador del usuario. En el interior de este directorio se almacenarán los distintos recursos que queramos visualizar.

5.4.6.4 iOS vs Android

La descarga de los ficheros se realiza correctamente en las dos plataformas, tanto en iOS como en Android. La visualización de los recursos es lo que difiere entre ambas plataformas. En iOS se visualizan correctamente tanto los videos como los distintos tipos de documentos. En Android, los videos se visualizan correctamente, pero los documentos no. Después de realizar muchas pruebas, consultar en la comunidad de desarrolladores de Titanium, y visitar distintos foros, vimos que mucha gente tenía problemas al visualizar PDFs y otros documentos en Android. Como consecuencia de esto, los únicos ficheros que se permitirá visualizar a los usuarios de Android serán los ficheros de video.

5.5 Conclusiones

En este capítulo nos hemos centrado en la aplicación desarrollada en el marco de la Tesina: SakMob.

Un aspecto clave en el desarrollo de cualquier aplicación es su especificación. En primer lugar hemos hecho la especificación de requisitos, en la que hemos enumerado las principales funcionalidades que tiene que tener la aplicación. A continuación hemos realizado la especificación de casos de uso, en la que se han descrito las distintas interacciones que se producen en el sistema. Para ello hemos especificado los diagramas de casos de uso, la descripción de cada uno de ellos, los actores relacionados, el escenario principal y las pruebas de aceptación. Otro aspecto incluido dentro del apartado de la especificación es la descripción de los distintos servicios Web desarrollados en Sakai, mediante los cuales obtenemos la información necesaria para la aplicación. Para cada uno de ellos se detalla su descripción, que información se quiere recuperar, que parámetros se utilizan y cuál es la respuesta XML.

Después de realizar la especificación de la aplicación, el siguiente paso es centrarse en los distintos elementos que han rodeado al desarrollo. De cara a poder entender mejor como ha sido todo el proceso de desarrollo, en el apartado 5.3 se han descrito la utilización de distintos ficheros. Estos ficheros cumplen distintas funciones, ya sea agrupar las propiedades de distintos componentes de interfaz, centralizar las llamadas a los servicios Web de Sakai o contener métodos que serán utilizados en más

de una ocasión. Otro aspecto clave tratado en este apartado es el uso de eventos globales, que nos permiten la ejecución entre distintos ficheros de determinadas acciones.

El siguiente aspecto analizado ha sido el desarrollo de algunos casos de uso. Para cada uno se ha descrito como es la interfaz utilizada, cual es el funcionamiento, en que ha consistido el desarrollo, que problemas se han encontrado y se ha comparado el desarrollo entre iOS y Android.

A pesar de que el resultado final de la aplicación ha sido satisfactorio, es importante destacar los problemas encontrados durante el desarrollo. Uno de los aspectos a mejorar en Titanium es el hecho de que en ocasiones la aplicación deja de funcionar o se produce un error al intentar implementar algo y la búsqueda de la solución suele ser bastante costosa. El uso de la sección *Q&A,* en la cual los usuarios preguntan dudas y reciben respuestas, a veces resulta confusa y no aporta la solución apropiada. Muchas veces es necesario realizar extensas búsquedas que no producen frutos.

Capítulo 6

6. Evaluación de la aplicación

6.1 Introducción

Una vez ya hemos visto en qué consiste la aplicación, cual es su especificación, el desarrollo que se ha seguido y los problemas que se han encontrado comparando el desarrollo para iOS y Android, el siguiente paso es evaluar la aplicación.

Un aspecto fundamental en el desarrollo de aplicaciones móviles es la realización de pruebas conforme va avanzando el desarrollo. Mediante los simuladores de las distintas plataformas podemos probar la aplicación conforme le vamos añadiendo funcionalidad.

Entre las distintas posibilidades de evaluación de una aplicación nos vamos a centrar en la evaluación de usabilidad. Para ella será necesaria la colaboración de un grupo de usuarios, de distintas características, que se presten realizar la prueba y rellenar el cuestionario.

Como hemos comentado, las pruebas en el simulador son importantes, pero para realizar la evaluación de la usabilidad es necesario realizar las pruebas en dispositivos reales, tanto del sistema iOS como del sistema Android. Gracias a esta evaluación de la aplicación podremos valorar de una manera más objetiva la calidad de la aplicación generada por Titanium, además de analizar las diferencias entre la aplicación generada para iOS y para Android

En los siguientes apartados veremos como se ha probado la aplicación SakMob, tanto en dispositivos reales como en el simulador. Además, describiremos el proceso seguido para realizar la evaluación de usabilidad de la aplicación.

6.2 Pruebas en dispositivos

Un aspecto fundamental en el desarrollo de aplicaciones móviles es la realización de pruebas conforme va avanzando el desarrollo. Desde el editor de Titanium podemos lanzar la aplicación en el simulador y también podemos empaquetar la aplicación para probarla directamente en el dispositivo [18]. Como requisito para la

realización de estas pruebas, tenemos que tener instaladas las herramientas de desarrollo (SDK) de iOS y Android.

6.2.1 Pruebas en simulador

Una vez se tienen instaladas las herramientas de desarrollo tanto de iOS como de Android, el desarrollador tiene la opción de realizar pruebas con el simulador. La configuración del simulador para su ejecución desde el editor se limita a unas pocas opciones. A continuación vamos a describir como han sido las pruebas para ambas plataformas.

6.2.1.1 Simulador iOS

De cara a utilizar el simulador de iOS se tienen que especificar un pequeño conjunto de opciones: sobre qué tipo de dispositivo vamos a ejecutar (iPhone/iPad), sobre que versión del *firmware* y que nivel de log queremos que se muestra en la consola del editor de Titanium.

Figura 6.1: Simuladores iOS

El funcionamiento del simulador de iOS es bastante óptimo. El tiempo de retardo del simulador en iniciarse no es muy elevado, y la velocidad en que la aplicación se ejecuta es correcta.

A pesar de esto, hay que resaltar que las primeras veces que se lanza el simulador después de realizar determinados cambios en el código no funciona de la manera esperada: se cierra sin motivo aparente, la aplicación no se ejecuta con fluidez, etc. Una vez se lanza y cierra el simulador una serie de veces, el funcionamiento del simulador es satisfactorio.

6.2.1.2 Simulador Android

Para la utilización del simulador de Android, también se tienen que especificar una serie de opciones de igual manera que se hacía en iOS. La diferencia en este caso estriba en que en lugar de elegir dispositivo en el que ejecutar, elegimos cual es la resolución de pantalla en la que queremos realizar la prueba.

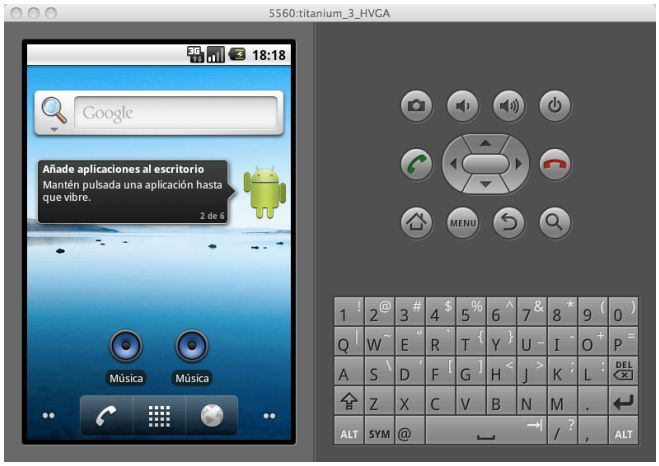


Figura 6.2: Simulador Android

El simulador de Android no presenta un comportamiento tan óptimo como el de iOS. El tiempo necesario para que se inicie el simulador es muy elevado, llegando a ser de 20 minutos en algunos casos.

Se da la circunstancia de que en muchos casos la aplicación no funciona correctamente sin motivo aparente. Por ejemplo, cuando el usuario introduce los datos para validarse en SakMob y pulsa la opción Entrar, la aplicación se queda indefinidamente en la ventana de carga. Si esta misma prueba se hace sobre un dispositivo Android, SakMob funciona perfectamente. Un problema derivado de esto ha sido la obtención de capturas de pantalla. La manera de obtener capturas de pantalla es a través de la ejecución de la aplicación en el simulador. Debido a estos problemas en el simulador, algunas capturas de SakMob no se podían realizar. La solución pasó por conectar el dispositivo real al ordenador mediante un cable USB, para posteriormente capturar las pantallas.

6.2.2 Pruebas en dispositivos

Como hemos comentado en el apartado anterior, las pruebas en el simulador son muy importantes porque nos permiten comprobar el funcionamiento de la aplicación conforme avanza el desarrollo de una forma sencilla y en algunos casos rápido. Hay que tener en cuenta que el comportamiento de la aplicación sobre el simulador no es exactamente igual al comportamiento que tendrá sobre un dispositivo real. Por esta razón es fundamental la realización de pruebas sobre los dispositivos. A continuación vamos a describir como han sido las pruebas para ambas plataformas.

6.2.2.1 Dispositivo iOS

Para probar una aplicación en un dispositivo iOS hay que seguir un complejo proceso. Apple exige que el desarrollador complete una serie de pasos:

- El usuario tiene que estar dado de alta como desarrollador.
- El dispositivo en el que queremos probar tiene que estar registrado en Apple.
- Obtener el *WWDR Intermediate Certificate* de Apple.
- Obtener el certificado de desarrollo de Apple.
- Obtener el *Development Provisioning Profile* de Apple.
- Subir el *Development Provisioning Profile.*

Una vez que hemos completado todos los pasos necesarios, el editor de Titanium detecta que los hemos realizado y nos permite probar sobre el dispositivo en cuestión. El dispositivo tiene que estar conectado mediante USB y sincronizado mediante *iTunes*, ya que desde ahí es desde donde se instalara la aplicación. Cuando pulsamos la opción de ejecutar en el dispositivo desde el editor de Titanium se produce la generación del ejecutable. A continuación la aplicación se importará automáticamente a *iTunes*, sincronizándose con el dispositivo.

Respecto a las pruebas de SakMob en dispositivos iOS los resultados fueron satisfactorios. Los dispositivos utilizados fueron:

- iPhone 3GS con iOS 4.3.
- iPad con iOS 4.3.

6.2.2.2 Dispositivo Android

La instalación de la aplicación en un dispositivo Android para poder probarla no requiere de ninguna configuración adicional. Lo único que hay que hacer es conectarlo mediante USB y pulsar la opción de probar en un dispositivo Android. Una vez hecho esto, Titanium genera el ejecutable y lo instala directamente en el dispositivo.

En las distintas pruebas realizadas en dispositivos Android el resultado fue satisfactorio. Los dispositivos utilizados fueron:

- Archos 70 Internet Tablet con Android 2.2.
- HTC Wildfire con Android 2.2.

6.3 Evaluación de usabilidad

En esta sección vamos a describir la evaluación de usabilidad diseñada para SakMob. Por un lado detallaremos el guión que deben seguir los usuarios para probar la aplicación. Por otro lado se presentará el tipo de cuestionario de usabilidad que el usuario deberá realizar una vez haya probado la aplicación. La realización de esta evaluación nos permitirá validar el desarrollo multiplataforma obtenido a partir de Titanium.

Para la realización de las pruebas se van a utilizar distintos dispositivos, tanto de la plataforma iOS, como de la plataforma Android. De esta manera podremos analizar también las diferentes valoraciones que puede tener la aplicación en función del dispositivo.

6.3.1 Participantes

El número total de participantes en la prueba es de diez personas, de los cuales seis son hombres y cuatro mujeres. La edad de los participantes está entre los 25 y los 36 años, siendo la media de edad 30 años. La profesión de los participantes es bastante diversa: estudiantes, desempleados, trabajadores en la construcción, etc.

Participante	Sexo	Profesión	Área de actividad	Edad	Nivel de educación	Smartphone
1	Hombre	Otros	Educación	36	Doctorado	Si
2	Hombre	Estudiante	TIC	27	Universitario	No
3	Hombre	Estudiante	TIC	30	Universitario	Si
4	Mujer	Desempleada		26	Universitario	Si
5	Hombre	Empleado	TIC	29	Universitario	Si
6	Hombre	Empleado	Construcción	35	F.P.	Si
7	Mujer	Estudiante	Psicología	26	Universitario	Si
8	Hombre	Estudiante	Optometría	29	Universitario	Si
9	Hombre	Desempleado		30	Universitario	Si
10	Mujer	Empleada	Riegos Laborales	32	Universitario	No

Tabla 6.1: Participantes evaluación de usabilidad

El nivel educativo de los participantes mayoritariamente es universitario, la mitad de los cuales tienen estudios relacionados con la ingeniería informática. La mayoría de los participantes son usuarios de *smartphones*, lo que facilitó la realización de las tareas. Un par de participantes no tenían experiencia previa en el uso de este tipo de dispositivos, lo que hizo necesaria la prestación de ayuda durante el ejercicio.

En líneas generales, el grupo de participantes es bastante diverso, aunque con características comunes. Con estos participantes conseguimos que la aplicación pueda ser probada sin excesivas explicaciones previas, pero con dificultad variable para los participantes.

6.3.2 Dispositivos

Los dispositivos usados en la realización de la evaluación fueron tres:

- iPhone 3GS con iOS 4.3.
- Archos 70 Internet Tablet con Android 2.2.
- HTC Wildfire con Android 2.2.

Cada uno de los participantes realizó la prueba sobre los tres dispositivos disponibles: dos teléfonos y un *tablet*. El objetivo es que cada usuario pueda probar la aplicación en distintos entornos, y que sepa valorar las diferencias de usabilidad existentes entre las plataformas iOS y Android, y entre usar un *tablet* y un teléfono. Con estos tres dispositivos se intenta abarcar tres contextos bien distintos. Por un lado, con el iPhone se prueba un *smartphone* de gama alta con el sistema iOS. Por otro lado, con la Archos 70 se prueba un *tablet* con Android con una pantalla más grande que un teléfono, con las ventajas que comporta. Por último, con el HTC Wildfire se prueba un móvil Android de gama baja, que nos permite comprobar que la aplicación es operativa en cualquier dispositivo.

6.3.3 Guión

A la hora de evaluar una aplicación, el usuario normalmente no la conoce previamente, por lo que es importante ofrecerle algún tipo de ayuda. Para ello se define un guión con las distintas tareas que deben llevar a cabo los usuarios. Este guión debe mostrar al usuario de una manera clara como es el funcionamiento de la aplicación, además de permitirle probar todos sus elementos. El objetivo es proporcionar una idea bien definida de qué hace la aplicación, cómo lo hace y qué funcionalidades tiene.

Las tareas definidas para el usuario son las siguientes:

- Validarse en la aplicación con el usuario asignado.
- Averiguar en que asignaturas está matriculado.
- Averiguar el número de anuncios que hay en cada una de las asignaturas.
- Averiguar que asignatura tiene un evento programado para hoy y en que consiste.
- Buscar el video referenciado en el evento, visualizarlo y seguir las instrucciones.
- Consultar la tarea requerida en el video.
- Refrescar la información y comprobar si hay un anuncio publicado en la última hora.
- Consultar el anuncio más reciente.

Una vez hecha concluidas las tareas, el siguiente paso es rellenar el cuestionario de evaluación.

6.3.4 Tipo de cuestionario

Para realizar la evaluación de usabilidad, el tipo de cuestionario elegido es el IBM CSUQ (*Computer Satisfaction Usability Questionnaire*) [30]. El IBM CSUQ es un instrumento para medir la satisfacción del usuario respecto a la usabilidad de un sistema, en el contexto de estudios de usabilidad basados en escenarios.

El CSUQ se divide en cuatro partes, cada una de las cuales se está basada en una escala de satisfacción de siete puntos:

- OVERALL: El resultado total de satisfacción (Preguntas 1-18).
- SYSUSE: El resultado de la utilidad del sistema (Preguntas 1-8).

- INFOQUAL: El resultado de la calidad de la información (Preguntas 9-15).
- INTERQUAL: El resultado de la calidad de la interfaz (Preguntas 16-18).

Una vez los usuarios han realizado las tareas, rellenarán cada una de las 18 preguntas de las que se compone el cuestionario CSUQ. A parte de contestar a estas preguntas, los participantes rellenaran una serie de datos complementarios que ayudarán al posterior análisis, como la edad, la formación académica, el grado de conocimiento de las plataformas, etc. Con las respuestas obtenidas, se realizará un análisis de los resultados. En los siguientes apartados se tratará esta información.

6.3.5 Resultados

Una vez realizadas las pruebas y rellenados los cuestionarios, el siguiente paso es mostrar los resultados para su posterior análisis. Como hemos comentado en el apartado 6.3.2, los participantes realizaron la prueba sobre los tres dispositivos disponibles: dos teléfonos y un *tablet*.

Respecto a la realización de las pruebas existen condicionantes que hay que tener en cuenta. El hecho de que cada usuario realice la prueba con los tres dispositivos puede provocar valoraciones no del todo objetivas. La primera prueba que realiza un participante con uno de los tres dispositivos siempre entraña más dificultad debido al desconocimiento de la aplicación y de las tareas a realizar. Una vez el usuario se dispone a realizar la segunda prueba, su grado de satisfacción es mayor. Debido a esto, el orden en que los participantes usaban los dispositivos variaba de un participante a otro. Como ejemplo, el participante 1 siguió la secuencia iPhone -> Archos 70 -> HTC Wildfire. En cambio, el orden de uso del participante 3 fue Archos 70 -> HTC Wildfire -> iPhone.

iPhone	Media	Mediana	Desv. Típica
SYSUSE	6,06	6,00	0,61
INFOQUAL	5,37	6,00	1,23
INTERQUAL	5,63	6,00	1,23
OVERALL	5,70	6,00	0,48

Tabla 6.2: Resultados CSUQ iPhone 3GS

En líneas generales, los resultados muestran una mejor valoración de la usabilidad en el iPhone, seguido de cerca del *tablet* Archos 70. El teléfono HTC Wildfire es el que peor puntuación tiene en cada uno de las partes de la escala CSUQ.

Archos 70	Media	Mediana	Desv. Típica
SYSUSE	5,74	6,00	0,88
INFOQUAL	4,90	5,00	1,39
INTERQUAL	5,50	5,00	1,39
OVERALL	5,40	5,00	0,84

Tabla 6.3: Resultados CSUQ Archos 70

Respecto a las distintas partes en las que está dividido el cuestionario CSUQ, la parte mejor valorada en los distintos dispositivos es la utilidad del sistema. En cambio, la parte peor valorada es la que determina la calidad de la información presentada al usuario.

HTC Wildfire	Media	Mediana	Desv. Típica
SYSUSE	5,01	5,00	0,82
INFOQUAL	4,79	5,00	1,19
INTERQUAL	4,90	5,00	1,19
OVERALL	5,20	5,00	0.64

Tabla 6.4: Resultados CSUQ HTC Wildfire

6.3.5.1 Análisis de los resultados

El conjunto de tareas a realizar tiene como objetivo familiarizar al participante con las distintas funcionalidades de la aplicación. Una vez el participante conoce el funcionamiento de la aplicación y ha probado su implementación en los distintos dispositivos, ya está en disposición de validar el desarrollo multiplataforma realizado con Titanium. Para ello es muy importante el hecho de probar la aplicación generada en los dispositivos disponibles: dos teléfonos (uno con iOS y el otro con Android) y un *tablet* (con Android). De esta manera se consigue validar la adaptación a los distintos entornos-dispositivos.

El análisis estadístico de los datos indica que el aspecto más destacado del desarrollo multiplataforma es la utilidad del sistema. Los usuarios valoran que el desarrollo realizado cumple con los requisitos esperados de la aplicación. Siguiendo el mismo patrón que en resto de partes del cuestionario, el mejor valorado fue el iPhone.

Otro de los aspectos destacados es la interfaz de la aplicación. El uso de componentes nativos por parte de Titanium es valorado positivamente por los participantes en cada uno de los dispositivos. De entre los tres, destaca la interfaz generada para iOS en el iPhone. La adaptación de la interfaz fue muy bien valorada en los distintos dispositivos, a excepción del teléfono HTC, debido a sus limitadas características. Su baja resolución y su pequeña pantalla influyeron negativamente en la valoración de la interfaz.

Como hemos comentado en el apartado anterior, el apartado peor valorado fue el referente a la calidad de la información. Ante determinados comportamientos erróneos de la aplicación, ésta no muestra información clara de cual es el problema o como hay que solucionarlo. Este es uno de los problemas derivados del desarrollo con Titanium. Cuando se produce algún no controlable el usuario no sabe como solucionarlo.

Respecto a los distintos tipos de usuarios, es importante resaltar como los dos participantes que no tienen un *smartphone* fueron los que mejor valoraron los tres dispositivos en cada una de las partes del cuestionario.

En líneas generales podemos concluir que los resultados de la evaluación fueron positivos.

6.4 Conclusiones

En este capítulo nos hemos centrado en la evaluación de la aplicación desarrollada.

En primer lugar hemos tratado el tema de las pruebas de la aplicación, tanto en el simulador como en el dispositivo. Las pruebas en el simulador son muy importantes porque nos permiten comprobar el funcionamiento de la aplicación conforme avanza el desarrollo. Pero es importante resaltar que la aplicación no se comporta igual sobre el simulador que sobre un dispositivo real. Por esta razón es fundamental la realización de pruebas sobre los dispositivos reales.

A continuación hemos descrito la evaluación de usabilidad realizada sobre la aplicación. Por un lado se ha detallado el conjunto de tareas que debían realizar los usuarios sobre los distintos dispositivos. Por otro se ha detallado el tipo de cuestionario de evaluación, los resultados obtenidos y su análisis. La realización de esta evaluación nos permite validar el desarrollo multiplataforma realizado con Titanium.

Capítulo 7

7. Conclusiones y trabajos futuros

7.1 Conclusiones

Dada la necesidad del proyecto europeo MOBIP de dar soporte a la formación en nuevas tecnologías para sus usuarios, éste hace uso de la solución *e-learning* Sakai. Sakai es un software educativo de código abierto, implantado sobretodo en instituciones de enseñanza superior. Este software posee múltiples funcionalidades de comunicación entre profesores y alumnos que pueden ser implementadas en una aplicación móvil, como la publicación de anuncios, aviso sobre determinados eventos, gestión de trabajos, etc.

Debido a la importancia de facilitar el acceso a los contenidos formativos de Sakai en cualquier momento y lugar, se planteó la posibilidad de proporcionar acceso a través de dispositivos móviles. En la actualidad, la fragmentación en el desarrollo de aplicaciones para dispositivos móviles es un problema muy presente. La aparición y proliferación de los distintos sistemas operativos y sus respectivas tiendas de aplicaciones, han hecho que el desarrollador tenga que enfrentarse a múltiples desafíos ya que si quiere tener disponible su aplicación para las distintas plataformas tendrá que realizar más de un desarrollo. Una de las posibles soluciones a este problema es el uso de los llamados *Frameworks* multiplataforma de desarrollo de aplicaciones móviles.

El primer objetivo que se planteó para la realización de la Tesina fue realizar un estudio entre los distintos *Frameworks* existentes, centrándonos en los aspectos a resaltar de cada una, además de mostrar tablas comparativas de las principales características y funcionalidades soportadas. El estudio se centra en las tres plataformas más comentadas: PhoneGap, Titanium y Rhomobile. Son las plataformas con mayor número de desarrolladores, además de ser las más comentadas y mejor analizadas en las distintas referencias existentes, y las que poseen una mayor número de aplicaciones en la App Store y el Android Market. Del análisis de estas plataformas se puede afirmar que:

- No hay ninguna plataforma objetivamente superior a las demás. Cada uno tiene sus puntos fuertes y sus puntos débiles.

- Las distintas plataformas soportan la mayoría de funcionalidades de los dispositivos.

- La elección de una plataforma vendrá determinada por las prioridades del desarrollador y el tipo de aplicación a desarrollar en cada caso.

Una vez realizado el estudio de las distintas plataformas y definido el contexto de la aplicación a realizar, la elección de una plataforma era el paso siguiente. El por qué de la elección de Titanium viene determinado principalmente por tres factores:

- Aspecto "nativo" de las aplicaciones. Titanium hace uso de componentes de interfaz nativos, otorgando a las aplicaciones un aspecto idéntico al de una aplicación desarrollada de una manera tradicional.

- Crecimiento espectacular de la plataforma. Este hecho viene refrendado por la gran cantidad de desarrollares que forman parte de la comunidad.

- Viabilidad futura de la plataforma. La compra de la compañía Aptana por más de 60 millones de dólares demuestra la ambición y las ganas de crecer de Titanium.

Con Titanium como entorno multiplataforma elegido, se abordó el objetivo de desarrollar clientes adaptados a las características de los distintos dispositivos móviles y que accedan a los recursos formativos de Sakai. El nombre elegido para la aplicación a desarrollar fue SakMob.

En base al trabajo realizado durante el desarrollo, el objetivo era crear un conjunto de recomendaciones o *best practices* que faciliten la toma de decisiones a las que se enfrentará un desarrollador novel en a la hora de enfrentarse a un desarrollo. Para ello se describió como ha sido el desarrollo de algunos casos de uso, detallando los distintos problemas encontrados, así como las diferencias entre el desarrollo para iOS y Android. Además, se enumeró el conjunto de técnicas y patrones de diseño que facilitan el desarrollo dentro de la plataforma Titanium, haciendo uso de distintos ficheros para ello. Estos ficheros cumplen distintas funciones, ya sea agrupar las propiedades de distintos componentes de interfaz, centralizar las llamadas a los servicios Web de Sakai o contener métodos que serán utilizados en más de una ocasión. Otro aspecto clave tratado es el uso de eventos globales, que nos permiten la ejecución de determinadas acciones entre distintos ficheros.

Con la aplicación ya desarrollada, el siguiente paso es validar el desarrollo multiplataforma obtenido. Para ello lo más importante es determinar si los clientes generados por Titanium para los distintos dispositivos hacen lo que se supone que deben hacer. Los aspectos a analizados son:

- En líneas generales, la funcionalidad obtenida es la especificada en los requisitos. La aplicación generada consta de las funciones requeridas, a excepción de la visualización de determinados recursos en Android.
- Adaptación a los distintos entornos-dispositivos. El uso de componentes nativos por parte de Titanium es un aspecto positivo que demuestra la adaptación al entorno de los desarrollos. Además, la interfaz es fácilmente adaptable a distintos tipos de pantalla y de resolución.

- La usabilidad de la aplicación es buena en los distintos dispositivos, como se extrae de la evaluación de usabilidad realizada sobre SakMob.

Pero no todo son aspectos positivos en el desarrollo con Titanium. Una vez dentro del desarrollo de una aplicación real surgen problemas no previstos. Uno de los principales focos de conflicto es el hecho de que existen diferencias entre el desarrollo para iOS y el desarrollo para Android. El uso de las distintas APIs de Titanium es casi siempre óptimo en iOS. En cambio, para Android los problemas son mucho más frecuentes. La conclusión evidente es que los creadores de Titanium han puesto mucho más énfasis sobre iOS, y el soporte de Android está un punto por detrás.

Además de esto, durante el desarrollo, es común la aparición de determinados problemas cuya solución no es trivial. En estos casos son necesarias múltiples búsquedas en la sección *Q&A* de la página de Titanium, en la cual los usuarios preguntan dudas y reciben respuestas, cuyo uso a veces resulta confuso y no aporta la solución apropiada. La duda que se plantea es saber si en proyectos grandes estas dificultades serán insalvables o no.

Además, es importante destacar el hecho de que la fragmentación no desaparece por completo, siempre hay aspectos a tener en cuanto durante el desarrollo:

- Existen determinadas funcionalidades de Titanium soportadas en una plataforma y en otra no. Por ejemplo, la visualización de PDFs y otros tipos de documentos funciona de manera óptima en iOS. En cambio, la visualización de este tipo de documentos no está soportada en Android.

- La fragmentación propia de la plataforma Android. Esto provoca que haya aspectos a tener en cuenta durante el desarrollo, como las múltiples resoluciones de pantalla disponibles en Android, los distintos tipos de almacenamiento interno, etc.

Como comentario final, es importante resaltar la satisfacción por parte de Samoo del trabajo realizado en esta Tesina. El hecho de abordar el desarrollo multiplataforma de un cliente móvil de Sakai es de gran interés para ellos, ya que es algo muy útil y que ahorra un gran trabajo a la hora de diseñar aplicaciones nativas. Desde Samoo podrán usar este trabajo como base para ampliar la calidad del servicio Sakai que proporcionan.

7.2 Trabajos futuros

Dentro de los posibles trabajos futuros nos encontramos con dos posibles caminos: profundizar en el trabajo con la plataforma Titanium o probar a desarrollar con otros entornos multiplataforma.

Con respecto a futuras líneas de trabajo con Titanium, merece la pena tener en cuenta la evolución del soporte para Blackberry, ya que actualmente se encuentra en fase Beta. El soporte a una tercera plataforma haría mucho más atractiva la propuesta de Titanium. Respecto a SakMob, existen muchas posibles funcionalidades de Sakai cuya

inclusión podría ser interesante. Otro posible futuro trabajo podría consistir en desarrollar una aplicación que haga uso de otras APIs, como el uso de mapas o geolocalización.

El desarrollo de aplicaciones con otros entornos multiplataforma, nos permitiría profundizar en otros enfoques dentro del desarrollo de aplicaciones móviles y comparar la calidad de las aplicaciones resultantes. De esta manera, tendríamos una visión más global de la funcionalidad de los distintos Frameworks.

Referencias

[1] Rajapakse, D. C. Fragmentation of Mobile Applications. Vers. 2.6. April 28, 2008.
http://www.comp.nus.edu.sg/~damithch/df/device-fragmentation.htm

[2] Allen, S., Graupera, V., Lundrigan, L. Pro Smartphone Cross-Platform Development: iPhone, Blackberry, Windows Mobile and Android Development and Distribution. Apress, 2010

[3] PhoneGap by Nitobi.
http://www.phonegap.com

[4] Rhomobile – Cross-Platform Mobile Development.
http://rhomobile.com

[5] Titanium Mobile Application Development by Appcelerator.
http://www.appcelerator.com/products/titanium-mobile-application-development/

[6] Mosync – The open source solution for cross-platform mobile development.
http://www.mosync.com

[7] Bedrock - Cross-platform mobile middleware solution.
http://www.metismo.com

[8] Alchemo – Java ME (J2ME) to BREW, iPhone, Android Flash and Windows Mobile Cross Compiler.
http://www.innaworks.com/alchemo-java-me-j2me-to-brew-android-iphone-flash-windows-mobile-cross-compiler

[9] JMango – Secure Mobile Application Solutions.
http://www.jmango.net

[10] Marmalade - Cross-Platform Mobile Application Development.
http://www.madewithmarmalade.com

[11] QuickConnectFamily – Hybrid mobile application development framework.
http://www.quickconnectfamily.org

[12] The Titanium Architecture.
http://wiki.appcelerator.org/display/guides/The+Titanium+Architecture

[13] Titanium – Working with Local Data.
http://wiki.appcelerator.org/display/guides/Working+with+Local+Data

[14] Titanium – Working with Remote Data.
http://wiki.appcelerator.org/display/guides/Working+with+Remote+Data

[15] U.S. Smartphone Market: Who's the Most Wanted. Nielsen, April 26, 2011.
http://blog.nielsen.com/nielsenwire/?p=27418

[16] Google's Android becomes the world's leading smartphone platform. Canalys, January 31, 2011.

http://www.canalys.com/newsroom/google%E2%80%99s-android-becomes-world%E2%80%99s-leading-smart-phone-platform

[17] Titanium – Internationalizing.

http://wiki.appcelerator.org/display/guides/Internationalizing+your+Application

[18] Titanium – Testing your application.

http://wiki.appcelerator.org/display/guides/Testing+your+Applications

[19] Titanium – The applicaction project structure.

http://wiki.appcelerator.org/display/guides/The+Application+Project+Structure

[20] Titanium – TableViews.

http://wiki.appcelerator.org/display/guides/Using+TableViews

[21] Peterson, K. The Sakai Project: Community Source Software for Academic Institutions. The Senate Source, April 2005.

[22] Sakai Project – Collaboration and learning, for educators by educators.

http://www.sakaiproject.org

[23] Sakai CLE Mobile – A cross-platform mobile application for the Sakai CLE (ie Sakai 2.x). Summer 2011 (May-August).

https://confluence.sakaiproject.org/display/CLEMBL/Home

[24] Aula TIC PYMEs de la USC. La movilidad como tendencia y como modelo de negocio. Agosto 2006.

[25] MOBIP (Europe Innova) – Mobile Services Innovation Platform.

http://www.europe-innova.eu/web/guest/innovation-in-services/kis-innovation-platform/mobip/about;jsessionid=04F89A483801A2CD28C3D1DE6D1A2B45

[26] Itaca – Instituto de Investigación y Desarrollo de la Universidad Politécnica de Valencia

http://www.itaca.upv.es/view.php/Principal

[27] Samoo – Marca de soluciones e-learning

http://samoo.es/

[28] Samoo Sakai

http://samoo.es/index.php/es/productos/sakai

[29] Samoo Sakai Español

http://samoo.es/index.php/es/component/content/article/17-roknewsrotator/61-sakai-en-espanol

[30] Lewis, J. R. IBM Computer Usability Satisfaction Questionnaires: Psychometric Evaluation and Instructions for Use. ACM - International Journal of Human-Computer Interaction. 1995.