Proyecto Fin de Carrera
Ingeniería de Telecomunicación

Desarrollo de Aplicación Multiplataforma en JavaScript con Appcelerator

Autor: Rafael Ángel Segura Giraldo
Tutora: Isabel Román Martínez

Departamento de Ingeniería Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Proyecto Fin de Carrera
Ingeniería de Telecomunicación

Desarrollo de Aplicación Multiplataforma en JavaScript con Appcelerator

Autor:
Rafael Ángel Segura Giraldo

Tutora:
Isabel Román Martínez
Profesora Colaboradora de Universidad

Departamento de Ingeniería Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2014

Proyecto Fin de Carrera: Desarrollo de Aplicación Multiplataforma en JavaScript con Appcelerator

Autor: Rafael Ángel Segura Giraldo

Tutora: Isabel Román Martínez

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2014

El Secretario del Tribunal

A mi familia

A Beatriz

Agradecimientos

.

Son muchos los días que han pasado desde que llegué a Sevilla para comenzar mi etapa universitaria.

En estos 9 años he pasado momentos difíciles…mi primer año…pero tenía junto a mí a mis padres, que con sus palabras y su cariño han conseguido que mi futuro se haga realidad.

Mis padres, Maribel y Rafael, siempre atentos sin dejar que mis ánimos decayeran, siempre con las palabras adecuadas para seguir adelante.

En estas líneas les quiero dar las gracias, hoy soy Ingeniero y todo gracias al esfuerzo, y las esperanzas que han depositado en mí.

Deciros que os quiero y os lo agradezco de todo corazón, sacrificáis vuestro tiempo por vuestros hijos sin pedir nada a cambio, cada llamada cada mensaje siempre en el momento adecuado hacen que todos los agradecimientos sean pocos, gracias de nuevo.

Tampoco me olvido de mi hermana, Maribel, amiga y confidente siempre dispuesta a escucharme y ayudarme en lo que necesitara. Sus consejos y sus palabras hicieron que fuera dando pasos en mis estudios, construyendo mi futuro. Escribiendo estas líneas pienso en cada momento que me has ayudado y no tendría tiempo para agradecértelo en esta vida.

Decir que te quiero, gracias por estar ahí y cuenta conmigo para todo lo que necesites.

Mi novia, Beatriz, tú entraste en mi vida hace ya casi cuatro años, tu sonrisa y tus palabras siempre han mantenido mis ganas de seguir delante de buscar el futuro para poder seguir juntos. Aunque nos han separado muchos kilómetros has estado cerca junto a mí, comprendiendo que la distancia era por mi carrera, comprendiéndome y apoyándome.

Gracias por estar ahí en los días más difíciles y en los más felices, por escucharme en mi días más negros y susurrarme "Tú puedes" y simplemente abrazarme con tu dulzura.

Gracias por ser mi compañera, mi amiga, mi novia, ser mi auxilio y estar a mi lado. Has hecho que estos últimos años sean los más felices y gracias por tus palabras de ánimo siempre que las necesitabas.

Gracias, te quiero.

Por último gracias a toda mi familia que ha estado a mi lado en estos años y siempre atentos.

Y gracias a mis amigos, cordobeses, onubenses y sevillanos, porque siempre habéis estado cerca aunque no pasáramos todo el tiempo deseado juntos.

Rafael Ángel Segura Giraldo

Sevilla, 2014

Índice

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

1 INTRODUCCIÓN

Cada día las personas se comprometen más con el medio ambiente y buscan aprovechar las fuentes de energía renovables. Para conseguir este propósito existen multitud de investigadores que buscan alternativas a los combustibles fósiles de los que tanto dependemos actualmente.

1.1 Contexto

El proyecto FERROSMARTGRID, tiene por objetivo Especificar y desarrollar un sistema para optimizar el aprovechamiento de la energía eléctrica en la red de transporte ferroviario. En este sistema participan vehículos eléctricos de particulares que puedan servir como almacenamiento de carga y, al mismo tiempo, pueden utilizar el sistema para la recarga de sus baterías. Además de estos dos servicios en un futuro se añadirán nuevos servicios que sean interesantes para el usuario.

Por su alcance y complejidad técnica, requiere desarrollos en ámbitos muy diversos. Uno de los bloques que se identificó desde el comienzo del proyecto fue el que se denominó red de usuario que contempla los subsistemas y procedimientos relacionados con la provisión de servicios a los usuarios de vehículos eléctricos del sistema.

La correcta definición y programación de las interfaces de comunicación entre los distintos sistemas que componen la red de usuario es un punto crucial para su correcta integración.

1.2 Objetivo

El presente proyecto tiene por objetivo la creación de una aplicación de usuario multiplataforma que se conecte a la red de usuario para ofrecer al usuario servicios finales relacionados con la gestión de reservas de plazas en las estaciones de recarga del sistema FERROSMARTGRID.

Una aplicación multiplataforma se puede definir como una aplicación creada y desarrollada en un lenguaje general y que pueda ser usada en cualquier dispositivo independientemente del sistema operativo que se utilice.

El interés en escribir una aplicación una vez y que ésta funcione en diferentes plataformas – *"write once, run everywhere"* – ha provocado la aparición de sistemas capaces de adaptar un lenguaje de alto nivel al específico de cada plataforma. Para empresas con la necesidad de una aplicación el uso de estos marcos de trabajo multiplataforma puede conseguir un ahorro de costes. Al desarrollar una única aplicación y necesitar así menor número de desarrolladores, menor tiempo de desarrollo… y un aumento de beneficios (al tener un mercado potencial mucho más amplio).

El proyecto está dividido en dos fases.

- La primera fase se destina a la búsqueda, recopilación y comparación de los sistemas actuales que dan la posibilidad de crear una aplicación para varias plataformas.
- La segunda fase es una fase más técnica. Tras la elección de una de las soluciones posibles desarrollar la aplicación y ponerla en funcionamiento. Profundizando en las características,

virtudes y desventajas, del entorno desarrollado

- En los anexos se han creado guías del programador y del usuario que faciliten a los interesados la introducción a esta tecnología.

En conclusión este proyecto tiene como finalidad la creación de una aplicación multiplataforma para la gestión de reservas de plazas en estaciones de recarga de vehículos eléctricos.

2 ESTADO DEL ARTE

Antes de comenzar el desarrollo de una aplicación compatible en todas las plataformas, se debe plantear una cuestión, **¿qué tecnología se debe utilizar?** Para poder responder a esta pregunta se ha realizado un proceso de búsqueda y análisis de las principales tecnologías existentes hasta 2013.

El primer paso para decidir que tecnología usar es conocer el número de plataformas existentes y decidir si es productivo centrarse en todas o solo en varias. Actualmente las plataformas predominantes en el mercado Android e iOS, conviven con BlackBerry RIM y Windows Phone, y en un nivel de cuota de mercado inferior se pueden encontrar otras olvidadas o casi desconocidas como son Bada, Symbian, Mozilla OS, o Ubuntu Touch.

Otro aspecto fundamental al desarrollar una aplicación, además de la penetración en la población, es su coste y el tiempo que llevaría desarrollarla.

A nivel de programación hay que tener en cuenta que no todas las tecnologías que se presentan en el proyecto permiten el acceso a las APIs internas, por ejemplo, GPS, cámara, teléfono, notificaciones, calendario… o bien necesitan un alto nivel de recursos para la ejecución de la aplicación.

Las opciones que se plantean a continuación cumplen algunas características pero en ningún caso existe una solución ideal.

2.1 Aplicaciones web

En la actualidad todos los dispositivos, Smartphone o Tablet, poseen un navegador HTML5, por lo que si el objetivo de la aplicación fuese llegar al mayor número de usuarios simplemente se podría crear una aplicación web y usarla desde el navegador de los dispositivos.

Figura 1.Lenguajes web

En una primera instancia se podría considerar la mejor opción para la creación de aplicaciones multiplataforma por las ventajas que presenta:

- Alto grado de penetración a bajo coste: Es una aplicación que se crea para un navegador por lo que abarcaría todas las plataformas que existen incluidos los Sistemas Operativos como Windows, Linux o Mac.
- Desarrollo y diseño sencillos: El desarrollador elige el lenguaje que quiere usar, Java, PHP, entre otros.
- Adaptación al entorno: Existe algunas opciones que pueden hacer que una aplicación web se parezca más a una aplicación nativa, por ejemplo usando un icono de aplicación o activando la opción de pantalla completa, entre otras opciones. Adaptar una aplicación web a diferentes resoluciones o tamaños de pantalla se consigue escribiendo un CSS - *Cascading Style Sheets*- por cada dispositivo.

El diseño de aplicaciones web presenta desventajas que también se deben tener en cuenta a la hora de decidir:

- Desarrollo: Antes se ha mencionado el abanico de lenguajes posibles, pero este punto a favor se ve ennegrecido debido a que no existe posibilidad de usar las Apis nativas de los dispositivos. Abarcar los diversos sistemas provoca que las aplicaciones web tengan un rendimiento ínfimo respecto a las aplicaciones nativas, por ejemplo cada petición que se realice dentro de la aplicación implicará una petición al servidor y una recarga de la página. Estos aspectos hacen que la fluidez o la experiencia de usuario nunca estén cercanas a la de una aplicación nativa.
- Accesibilidad: Un factor negativo a la hora de la experiencia de usuario es que éste debe conocer la URL, para poder introducirla en el navegador. Una vez realizado el primer acceso a la aplicación web el usuario, para no tener que recordar la URL, podría guardarla en favoritos o crear un acceso directo para poder acceder a ella en un momento posterior. Todos los usuarios que deseen tener un acceso más cómodo deben realizar estos pasos, es un camino tedioso en comparación con la instalación de una aplicación desde el *market* de cualquier plataforma que hará que muchos de ellos busquen alternativas.
- Rentabilidad: En la actualidad el pago a través de las diferentes plataformas móviles se ha unido a la existencia de una tienda de aplicaciones, *App Store* o *Andorid Market*, es un método cómodo y sencillo tanto para el usuario como para el desarrollador, en cambio en las aplicaciones web se pierde el poder usar este mecanismo porque cada aplicación deberá ser abonada por su propio medio.

2.2 Aplicaciones Nativas

Las aplicaciones nativas son la solución ideal ya que están desarrolladas con herramientas específicas para cada sistema; con su propio lenguaje:

- Objective-C en iOS.
- Java en Android.
- C# y .NET en Windows Phone.

Además poseen un entorno de desarrollo específico, diferente para cada uno:

- Xcode para iOS.
- ADT para Eclipse para Android.

También poseen APIs propias para acceder a características específicas del Smartphone.

Figura 2.Plataformas Móviles

A nivel de usuario las aplicaciones nativas mejoran la experiencia de usuario, mejor fluidez, un diseño acorde con el sistema operativo, una interacción más optimizada y acceso a todas las funciones del dispositivo.

Al igual que las ventajas son muy llamativas existen inconvenientes por los cuales se buscan las alternativas.

La primera gran desventaja es para el desarrollador, no le basta con conocer un lenguaje de programación si no que es necesario que conozca cada lenguaje, esto tiene una alta curva de aprendizaje y de tiempo dedicado. La necesidad de crear un diseño para cada entorno con sus normas de estilo hace que aumente el tiempo de desarrollo.

En cuanto a las herramientas de desarrollo, Android permite instalar su entorno de desarrollo en cualquier sistema operativo mientras que iOS solo permite utilizar Mac, previo pago de una licencia anual para poder probarlas en el dispositivo personal.

Los factores mencionados anteriormente hacen que el coste de una aplicación multiplataforma

aumente considerablemente haciendo que muchas empresas se planteen si el desembolso es necesario o excesivo.

2.3 Aplicaciones híbridas

Las aplicaciones híbridas buscan dar solución a la premisa mencionada en la introducción -*"write once, run everywhere"*- con un software único y un lenguaje determinado que permita crear una aplicación multiplataforma.

En las aplicaciones híbridas se pueden encontrar varias clasificaciones. La primera diferenciación se hace respecto a la necesidad o no de tener que usar un Mac, en este segmento solo existe un entorno de desarrollo que nos dé la opción de no usar un Mac.

2.3.1 Adobe Air

Figura 3.Logo Adobe Air

Actualmente es la única herramienta que permite construir aplicaciones para iOS sin la necesidad de compilarlo en un Mac ni usar Xcode.

Para el desarrollo de aplicaciones se utiliza Flex 4, un grupo de tecnologías para el desarrollo de aplicaciones enriquecidas de internet, es decir, aplicaciones web que tienen la mayoría de las características de las aplicaciones de escritorio tradicionales. Estas aplicaciones utilizan un navegador web estandarizado para ejecutarse y por medio de complementos o mediante una máquina virtual se agregan las características adicionales.

El lenguaje que utiliza Flex 4 es ActionScript, es un lenguaje que mejora la eficiencia de las aplicaciones basadas en Flash, está destinado a crear scripts en Flash. Es un lenguaje orientado a objetos con características parecidas a Java y otros lenguajes orientados a objetos.

Actualmente en Adobe Air se pueden desarrollar aplicaciones para iOS, Android y Blackberry así como para los sistemas de escritorio Windows, Mac y Linux. Las aplicaciones de escritorio que se realizan en Adobe Air se ejecutan instrucción a instrucción.

Una desventaja es que no se puede crear una aplicación única para dispositivos móviles y sistemas de escritorio porque existen aspectos visuales que son diferentes.

Las características principales que este sistema presenta son:

- Multiplataforma: Como ya se ha comentado anteriormente, se pueden desarrollar aplicaciones

para dispositivos móviles y sistemas de escritorio.

- Presenta un lenguaje muy potente, ActionScript, capaz de usar patrones y estructuras complejas en el desarrollo de aplicaciones.
- Documentación: Tanto el IDE como Flex 4 están muy avanzados por lo que es sencillo encontrar información ya que Adobe ha apostado fuertemente por este sistema.
- Como ya se ha mencionado antes Flash Builder 4.5, el entorno de desarrollo, no requiere el uso de Xcode ni de un Mac.

Existen contras que también se tienen que tener en cuenta si se desea trabajar con este sistema:

- Precio: El coste de Flash Builder 4.5 es alto, aunque es posible usar Flex SDK que es OpenSource basado en Java.
- Experiencia de usuario: El rendimiento de los elementos nativos, APIs del sistema, no consiguen la fluidez de la programación nativa y presenta un aspecto diferente a las líneas que están siguiendo los sistemas nativos. En las aplicaciones de escritorio presenta un alto consumo de CPU.
- Desarrollo: Con este sistema no se puede acceder a toda la gama de dispositivos Android, debido a la necesidad de una arquitectura ARM 7 para su funcionamiento, esta arquitectura está presente en los dispositivos Android de gama alta.

Como apunte final, se pueden desarrollar y definir vistas de manera gráfica usando el editor visual de MXML con Flash Builder.

Los demás sistemas, como se ha comentado anteriormente, hacen uso de Xcode y de un Mac para el desarrollo de aplicaciones multiplataforma.

Dentro de las aplicaciones híbridas, y descartando Adobe Air se puede distinguir entre dos tipos de aplicaciones híbridas: basadas en web app e interpretadas.

2.3.2 Aplicaciones Híbridas: Basadas en web app

2.3.2.1 eMobc

Figura 4.Logo eMobc

Es una iniciativa recién aparecida que actualmente carece de documentación. Se encuentra en fase de desarrollo.

Es un framework de licencia libre (Affero GPLv3) para el desarrollo multiplataforma ue permite desarrollar pantallas de aplicaciones para iOS, Android, HTML 5 y Jquery Mobile, para poder crear aplicaciones móviles usando el patrón MVC [Modelo-Vista-Controlador]. En cada plataforma se encuentra un orquestador para la gestión de la navegación por las pantallas. La vista que se muestra en

cada pantalla es definida por datos que están contenidos en un archivo XML.

Para ayudar a desarrollar aplicaciones móviles multiplataforma se ha definido un lenguaje llamado embML, basado en XML. Gracias a él se pueden definir aspectos presentes en una aplicación móvil como son pantallas, menús, apariencia o contenido, entre otros.

Una de las ventajas que presenta es que todas las APIs se han creado en código nativo para cada plataforma.

Actualmente el problema que presenta es que está en fase desarrollo, las versiones del framework se encuentra en la versión 0.1.

2.3.2.2 PhoneGap

Figura 5.Logo PhoneGap

Es un proyecto que comenzó bajo licencias de software libre, pero en Octubre de 2011 Adobe adquirió Nitobi, poseedor de PhoneGap. Aunque en un primer momento en la comunidad de desarrolladores se generó una gran incertidumbre por la posibilidad de que se convirtiera en una tecnología propietaria, Adobe donó PhoneGap a la fundación Apache, conservando de esta forma la integridad libre del proyecto.

Apache renombró el proyecto como Apache Cordova, aunque en la actualidad la marca de referencia sigue siendo PhoneGap.

Las aplicaciones creadas con PhoneGap son desarrolladas usando exclusivamente HTML5, CSS3 y Javascript, código que será ejecutado en un component WebKit del móvil. Además se incluyen unas librerías Javascript desarrolladas en el lenguaje específico de cada plataforma para poder acceder a las APIs internas de cada dispositivo.

Al ser una tecnología que se basa en el componente WebKit del móvil una aplicación se puede ver como una serie de páginas web que se almacenan y se empaquetan dentro de esta aplicación con una apariencia de navegador web. Para el diseño gráfico al tratarse de una aplicación web se pueden usar

frameworks web mobile como jQuery o Sencha Touch.

Este sistema presenta una serie de ventajas importantes para el desarrollo multiplataforma:

- Plataformas: PhoneGap es la tecnología que más plataformas abarca al ejecutarse dentro de un navegador web. Se puede desarrollar aplicaciones para iOS, Android, Windows Phone, BlackBerry, Symbian, WebOS y Palm.
- No es necesario usar un Smartphone o en su defecto un simulador durante la depuración de la aplicación ya que se basa en un entorno web y se puede ejecutar en cualquier navegador que se tenga instalado.
- Lenguajes de alto nivel: Para los desarrolladores web es una manera de poder introducirse en el campo de las aplicaciones, y al ser de alto nivel la curva de aprendizaje es menor incluso para programadores que vengan de otro tipo de lenguajes.
- Documentación: Al ser propiedad de Adobe y donado a Apache la documentación está organizada y accesible incluyendo ejemplos que ayudan a la hora de comenzar en esta tecnología.
- Coste: este entorno es totalmente gratuito, por lo que se hace atractivo para los desarrolladores que quieran adentrarse en el mundo de las aplicaciones.

Existen otras características negativas que también se deben tener en cuenta para decantarse por este sistema de desarrollo.

- Plataformas: Para trabajar con cada plataforma se debe usar un sistema distinto. Para iOS se tiene que usar Xcode, como ya se comentó solo disponible en MAC, para Android se debe usar Eclipse, para Blackberry no existe aún un entorno definido, y todos llevan asociado al proyecto una plantilla proporcionada por PhoneGap. Una vez escrito el código será necesario compilarlo para cada sistema que se desee utilizar.
- Aplicación: La aplicación es una página web, por lo que su aspecto no se asemejará al de una aplicación tradicional. Para conseguir una apariencia de aplicación nativa se deberán usar frameworks de HTML como Sencha Touch, jQuery mobile, Jo, Sproutcore, XUI o jQTouch.
- Rendimiento: Al igual que se comentó en otras soluciones el rendimiento está muy por debajo del rendimiento de una aplicación nativa, se debe a que al ser desarrollado en lenguaje web, HTML, CSS y Javascript, debe ser leído e interpretado por el motor del navegador cada vez que se inicia la aplicación.

2.3.3 Aplicaciones Híbridas: Interpretadas

2.3.3.1 Appcelerator

Appcelerator es una compañía tecnológica dedicada al desarrollo de suite informáticas, desarrollo y análisis de aplicaciones móviles. En 2012 decidió centrar su negocio en el desarrollo de aplicaciones móviles y comenzaron a desarrollar Titanium.

Figura 6.Appcelerator y Titanium

Titanium es un IDE [Entorno de Desarrollo Integrado] de licencia abierta basado en Eclipse y Aptana que permite la creación de aplicaciones móviles, para Android, iOS, Tizen, Blackberry además de aplicaciones web, usando exclusivamente Javascript. Titanium transforma y compila el código en Javascript antes de empaquetarlo en una aplicación nativa. Este código será ejecutado al abrir la aplicación en una máquina Javascript propia del dispositivo, JavaScriptCore en iOS y Mozilla Rhino en Android y BlackBerry, ambos intérpretes del Webkit.

Las aplicaciones son programadas totalmente en Javascript. Aunque Appcelerator proporciona una serie de librerías que sirven para acceder a los controles nativos del dispositivo, consiguiendo acercarse a una apariencia prácticamente nativa. Con estas librerías se consigue un mejor rendimiento y que la experiencia de usuario sea más satisfactoria.

Al igual que las anteriores plataformas Appcelerator presenta características muy positivas.

- Rendimiento: Gracias a la interpretación del código en tiempo de ejecución el rendimiento mejora notablemente aunque sin llegar al punto de una aplicación nativa.
- Coste: Titanium es gratuito, simplemente se tendrán que pagar algunas APIs que se encuentran en un mercado propio de Appcelerator.
- Diseño: El diseño se aleja de la maquetación web acercándose al aspecto nativo.
- En cuanto a documentación posee una documentación completa además de una comunidad activa para el aporte de soluciones y ayuda a los desarrolladores.

Como todo software no nativo presenta características negativas.

- Se siguen necesitando las plataformas propias de los sistemas para poder compilar la aplicación, es decir, Xcode en caso de iOS o el SDK de Android.
- El diseño y maquetación de los diferentes componentes visuales y los controles se debe realizar con código.
- Si se usa jQuery o cualquier otra librería de Javascript para usar DOM en aplicaciones de escritorio los archivos HTML, JS y demás tipos de archivos aparecen en claro en la capeta de aplicación.

2.3.3.2 Xamarin

Xamarin fue creada en mayo de 2011. Es una empresa que busca, igual que las opciones presentadas anteriormente, dar una solución al desarrollo de aplicaciones multiplataforma.

Figura 7.Logo Xamarin

El concepto de Xamarin es sencillo: usando los lenguajes C# y .NET compilar aplicaciones nativas para Android o iOS.

Una de las ventajas que presenta el uso de C# y .NET es que son los lenguajes de programación de Windows Phone, Windows 8 y sirven para el desarrollo web, por lo que abarca una amplia gama de dispositivos. Al ser el lenguaje usado en Windows existe mucho código reutilizable con una amplia comunidad de desarrolladores detrás.

Xamarin también proporciona un acceso total a las APIs de los sistemas.

Como C# y .NET tienen un parecido conceptual a Java cuando se desarrollen aplicaciones con Xamarin aparecerán APIs con una implementación similar.

Existen aspectos negativos:

- El código desarrollado para las interfaces gráficas no podrá ser reutilizado.
- El desarrollador tendrá que tener la capacidad de desacoplar esta funcionalidad.

Otro aspecto negativo es el espacio que ocupa una aplicación desarrollada en Xamarin, mayor que en otros casos, lo que afecta a los tiempos de descarga y al almacenamiento.

Otro de los inconvenientes que presenta Xamarin es la necesidad de aprender a usar las APIs tanto de Android como de iOS, por lo que la curva de aprendizaje se multiplica por las plataformas para las que se quiera desarrollar la aplicación.

2.3.3.3 Ansa Corona

Corona es un framework desarrollado en Lua. Lua es un lenguaje basado en script, rápido y potente si se tiene por objetivo desarrollar juegos o aplicaciones que requieran un alto rendimiento en gráficos.

Figura 8.Logo lenguaje Lua

Una de las desventajas es que no posee un IDE, aunque si posee un intérprete-emulador. Existe una versión gratuita que permite construir las aplicaciones pero el precio de la licencia es bastante alto.

Las ventajas que tiene esta tecnología son:

- Un motor gráfico y físico ideal para desarrollo de aplicaciones gráficas.
- Posee un lenguaje sencillo, Lua, con una curva de aprendizaje baja.
- Existe un gran catálogo de documentación, ejemplos y plantillas.

Pero también presenta desventajas.

- Coste: El precio a pagar por una licencia es alto si no se poseen unos ingresos y si no se le va a sacar un buen rendimiento.
- Específico: Aunque se pueden desarrollar aplicaciones de cualquier tipo, es una tecnología pensada para aplicaciones gráficas y juegos.

3 MATERIAL Y MÉTODOS

3.1 Elección del entorno de desarrollo

Tras haber conocido los diferentes entornos de desarrollo, en este proyecto se han elegido dos de las opciones disponibles para compararlas. Las razones por las que se ha decidido quedarse con las dos opciones han sido:

- Coste: El coste de la solución debe ser gratuito por lo que los proyectos Adobe Air y Ansa Corona han sido descartados.
- Tiempo de desarrollo y aprendizaje: La línea de aprendizaje debe ser moderada. Por lo que se descarta el uso del desarrollo de aplicaciones nativas y de Xamarin debido a que se debería adquirir conocimientos de varios lenguajes.
- Por último se ha descartado el desarrollo de una aplicación web debido a que se busca una solución más cercana a una aplicación nativa.

A continuación se comparan las dos opciones elegidas: PhoneGap y Appcelerator.

	PhoneGap	**Appcelerator**
Plataformas	Andorid, iOS, Blackberry, Tizen, Firefox OS, Amazon OS, Windows, Mac, Linux	Android, iOS, Blackberry, Tizen, Windows, Mac, Linux
IDE	Depende de la plataforma: Eclipse, Xcode. Blacberry no posee SDK	Titanium Studio (basado en Eclipse)
Lenguaje de desarrollo	Javascript	Javascript
Layout	HTML5 /CSS3	No
APIs	HTML y frameworks externos como Sencha Touc, jQuery Mobile, etc.	Nativas, HTML y otros framework externos
Rendimiento	Fluido, muy buena experiencia de usuario	Ejecutado en el motor del navegador, rendimiento medio
Documentación	Buena	Buena
Comunidad	Buena	Buena
Gráficos	Bajo	Bajo
Código fuente visible	Si	En las plataformas de escritorio si, en las plataformas móviles no
Compatibilidad	Todos los dispositivos	Todos los dispositivos

Tabla 1.Comparativa Phonegap y Appcelerator

Tras el análisis comparativo se puede ver en la Tabla 1 que ambas tecnologías usan el lenguaje Javascript para su desarrollo, son gratuitas y compatibles en todos los dispositivos. El uso de un lenguaje de alto nivel como es Javascript hace que la línea de aprendizaje no tenga una alta pendiente. El carácter gratuito ha sido un factor imprescindible para este proyecto, porque lo que se busca es dar una opción de bajo coste accesible a todos los desarrolladores.

Otros factores determinantes y de los cuales no se puede discriminar ninguna de las dos plataformas son la documentación y la comunidad de desarrolladores. Para Appcelerator en su página web se puede encontrar un enlace a la documentación:

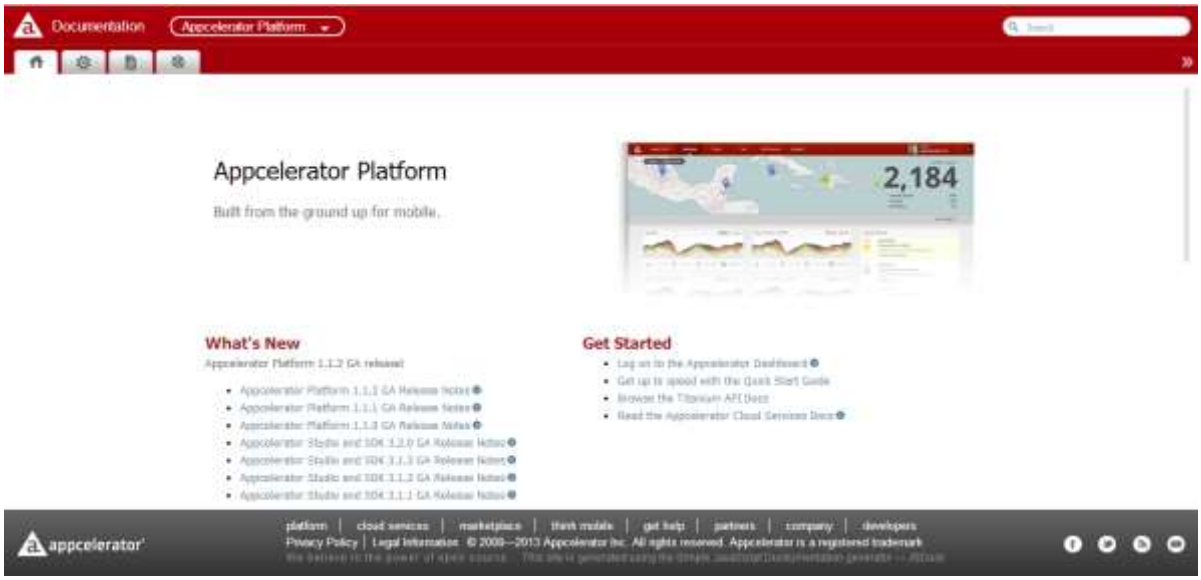


Figura 9.Documentación Appcelerator

http://docs.appcelerator.com/

Con tutoriales y librerías para el uso de APIs nativas de los sistemas.

Lo mismo ocurre en el caso de PhoneGap encontrando toda la documentación en su página oficial:

http://docs.phonegap.com/en/edge/guide_platforms_index.md.html

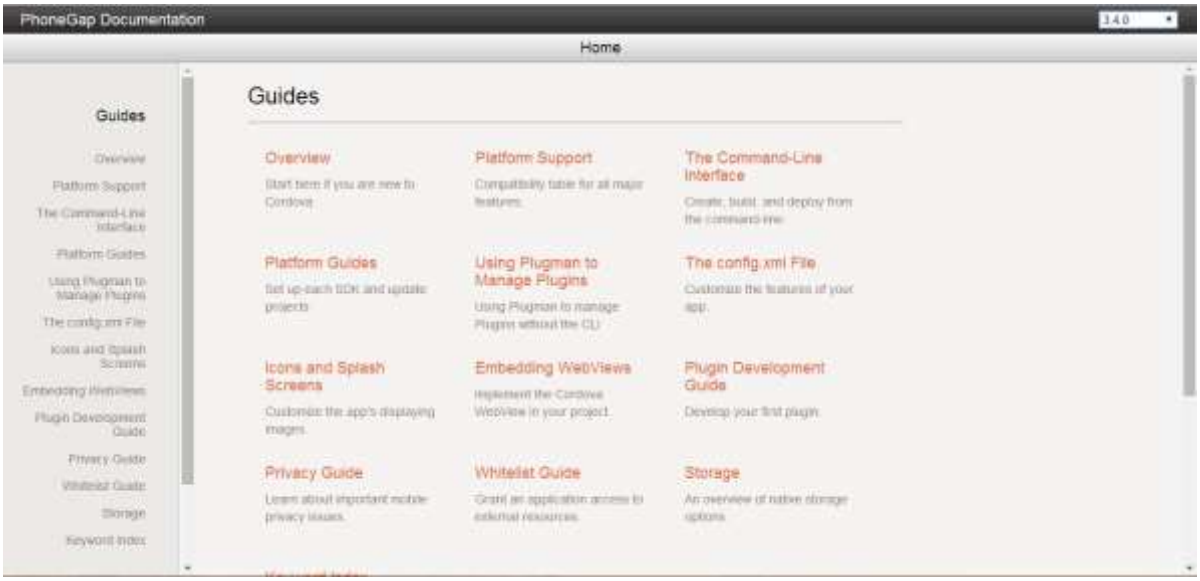


Figura 10.Documentación PhoneGap

En cuanto a la comunidad de desarrolladores cada día es más amplia y al ser una comunidad abierta existen muchos foros dónde los desarrolladores comparten su experiencia y prestan su ayuda desinteresadamente a otros desarrolladores.

Una de las primeras diferencias que se encuentran es el IDE que se debe usar.

- Titanium: En Appcelerator han desarrollado un IDE basado en eclipse compatible con todos los sistemas operativos y para todas las plataformas llamado Titanium, sobre el que se trabaja directamente, aunque es necesario instalar Xcode, en el caso de iOS, o el SDK de Android.
- Cordova Shell Tools: Phonegap, al ser una distribución del proyecto Apache Cordova, hace uso de Cordova Shell Tools. Esta herramienta es una interfaz de línea de comandos para poder crear, construir y ejecutar las aplicaciones desarrolladas. A diferencia de Appcelerator que solo hace uso de los SDK de cada plataforma Phonegap necesita el IDE de cada una para desarrollar la aplicación.

En este caso Titanium da una cobertura más global para el desarrollo de una aplicación multiplataforma.

La otra diferencia determinante es la experiencia del usuario. Al final el objetivo de toda aplicación es que el usuario la use, pero el triunfo o fracaso de la misma depende de factores como la rapidez de inicio, la fluidez entre los diferentes menús, y otros factores apreciables para el usuario más relacionados con el diseño de la aplicación que con la tecnología usada.

- PhoneGap: El rendimiento en este caso se ve disminuido debido a que las aplicaciones se ejecutarán dentro de un “envoltorio” específico para cada plataforma, es decir, se empaquetan y se ejecutan sobre el motor del navegador web, a simple vista se podrá ver como una aplicación pero su interior es una serie de páginas web, es decir una web empaquetada en una aplicación nativa. Aunque sea una web se pueden usar APIs del sistema aunque el tiempo de espera es

mayor. Es idónea para aplicaciones sencillas.

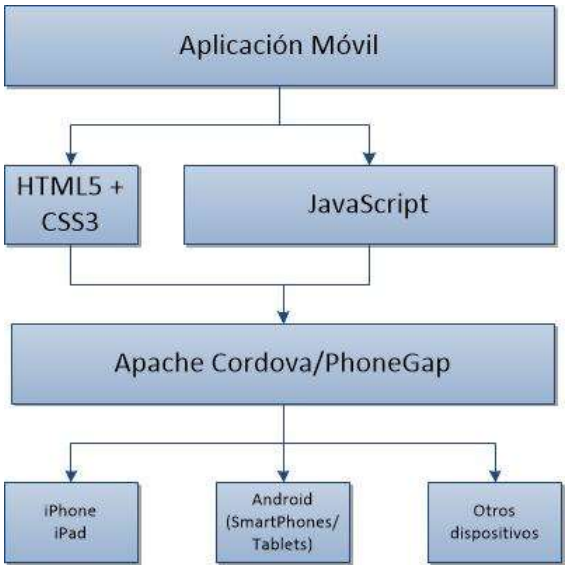


Figura 11.Arquitectura de Phonegap

- Appcelerator: En este caso el rendimiento puede acercarse al de una aplicación nativa aunque nunca se llegará a este punto. A diferencia de en PhoneGap en este caso la aplicación desarrollada en Javascript es interpretada en tiempo de ejecución. No se hace uso de un motor web que muestra diferentes páginas web, sino que se usa una máquina virtual de Javascript para interpretar el código mostrando una aplicación con apariencia nativa. Esto hace que los tiempos de respuesta sean mucho mayores.

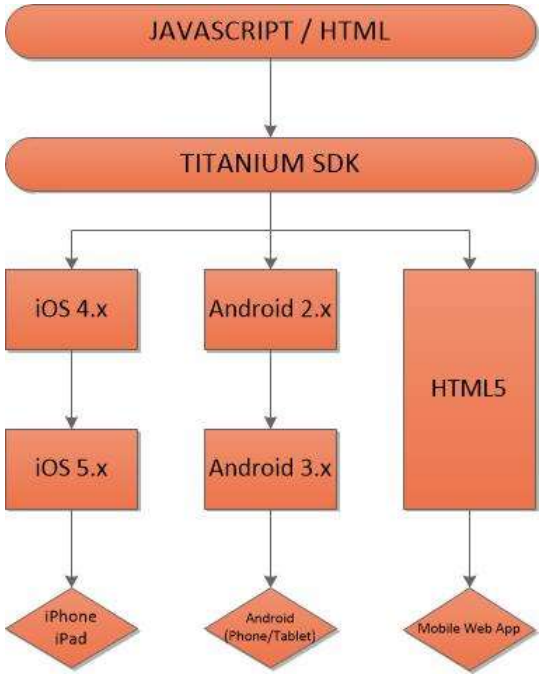


Figura 12.Arquitectura Appcelerator

La solución elegida será Titanium de Appcelerator, basando la elección en:

- Primero la experiencia de usuario.
- Usar un solo IDE para su desarrollo.

3.2 Herramientas de desarrollo

Además del entorno de desarrollo elegido, Appceelerator, en este proyecto se hace uso de otras herramientas. El lenguaje de programación usado será javascript, la sintaxis de envío de información JSON y el sistema de comunicación REST. A continuación se detalla el conjunto de herramientas que se han utilizado para el desarrollo del proyecto.

3.2.1 Titanium

Titanium Studio es un entorno de desarrollo para crear aplicaciones para las diferentes plataformas. Está basado en el IDE Eclipse, se puede conectar a los servicios en nube que ofrece Appcelerator y contiene las librerías de Appcelerator para el desarrollo de las aplicaciones.

Figura 13.Titanium Studio

El objetivo de Titanium es proveer un lenguaje de alto nivel, Javascript, para dar respuesta al problema del desarrollo multiplataforma. Titanium se puede entender como un marco de trabajo para la escritura de aplicaciones nativas sin la necesidad de usar un lenguaje específico para cada plataforma.

Figura 14.Javascript como lenguaje de desarrollo

Una de las cosas que se debe tener en cuenta es que se tienen que instalar los SDKs de cada plataforma, pero el desarrollo se realiza totalmente en Titanium sin la necesidad de aprender a utilizar los entornos de cada plataforma. Esto se debe a que el objetivo es conseguir una aplicación nativa a través de un código escrito en Javascript.

Titanium ofrece una sencilla interfaz para gestionar todos los proyectos que se tengan, también ofrece las herramientas necesarias para ejecutar una aplicación en un simulador o en un dispositivo o incluso para publicar una aplicación en App Store de Apple o en Google Play de Android.

Titanium tiene un corrector automático de sintaxis. Es fácil que durante la programación se olvide una coma, o un punto y coma por ejemplo, para ello Titanium Studio realiza una comprobación de la sintaxis y comprueba si es correcta, esta propiedad la hereda de Eclipse.

Otra propiedad que posee Titanium es la ayuda en tiempo de codificación, se comporta de manera similar a Eclipse, permite el autocompletado de instrucciones de código como se ve en la figura 15.

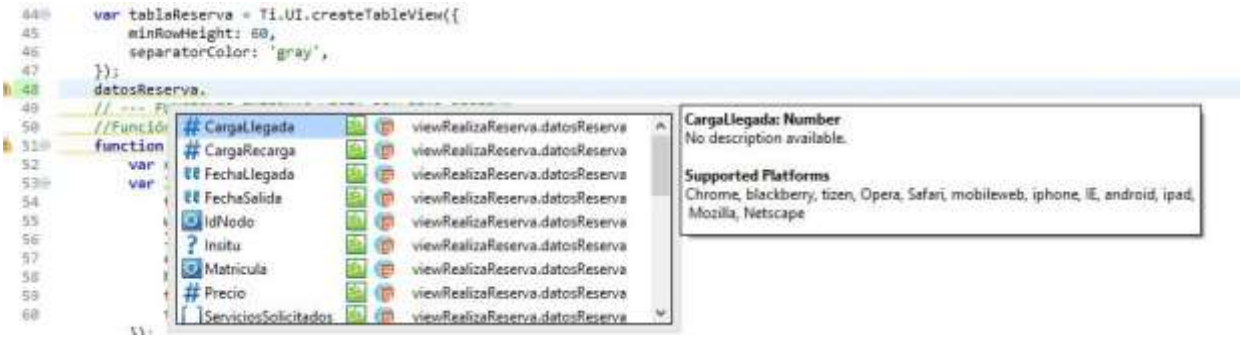


Figura 15.Autocompletado de instrucciones en Titanium

En cuanto a la depuración Titanium Studio provee de herramientas de ayuda al programador. Se pueden definir puntos de ruptura y ver los valores de variables. La posibilidad de ver múltiples valores cuando se ejecuta el programa es de mucha ayuda para grandes aplicaciones en las que algunas

variables pueden ir cambiando de valor.

Cuando se construye la aplicación Titanium a través del código Javascript construye un proyecto nativo. Este proceso se realiza con un script definido en las propiedades del proyecto. Titanium construye el proyecto en el entorno nativo que se ha elegido y después las herramientas de compilación se encargan de convertir la aplicación en código nativo binario.

En Android, Titanium ayuda a la gestión del simulador de Android. Esto es debido a que Android trabaja en diferentes tipos de hardware y de configuraciones de pantallas. Es bueno por lo tanto en el caso de Android probar la aplicación para varios tipos de pantallas.

3.2.2 Javascript

Javascript es un lenguaje de programación interpretado. Surgió para conseguir páginas web dinámicas, se buscaba que en las páginas web existiera una interacción del usuario o que se realizaran automatismos sencillos.

Se utiliza principalmente para el lado del cliente mejorando la interfaz de usuario en las páginas web, se podría usar en el lado del servidor pero no es lo normal, también se usa para aplicaciones de escritorio, los llamados widgets, o en documentos PDF.

Javascript no se puede considerar un lenguaje como C, C++ o Java. Es un lenguaje de script u orientado a documentos, aunque se diseñó con una sintaxis similar al lenguaje C y con nombres y convenciones de Java. Pero el propósito y la semántica de Java no están relacionados con Javascript.

Algunas características de síntesis y semántica son:

- Soporta la estructura de programación C.
- El alcance de las variables es sólo en la función donde fueron definidas/declaradas, en cambio en C se pueden crear variables globales o pasar variables por referencia a otras funciones en las que pueden ser modificadas.
- Está formado casi en su totalidad por objetos (arrays asociativos).
- Las propiedades y los valores de los objetos pueden ser creados, cambiados o eliminados en tiempo de ejecución. Esta propiedad hace que no sea necesario declarar el tipo de una variable ya que esta puede almacenar varios tipos en tiempo de ejecución.
- No se tienen en cuentan espacios o nuevas líneas, aunque por estilo se usan.
- Se distingue entre mayúsculas y minúsculas.
- No es necesario terminar cada sentencia con el carácter punto y coma, aunque se recomienda su uso para seguir unas normas de estilo.
- Al igual que en C o Java se pueden incluir comentarios.

3.2.3 JSON

JSON (JavaScript Object Notation) es un formato ligero de intercambio de datos. Leerlo y escribirlo es simple para humanos, mientras que para las máquinas es simple interpretarlo y generarlo.

Es un lenguaje diseñado para el intercambio de datos, con un formato de texto que utiliza convenciones que son ampliamente conocidas por los programadores, incluyendo C, C++, C#, Java, JavaScript, Perl, Python, y muchos otros.

JSON tiene dos componentes fundamentales:

- Una colección de pares nombre/valor. Esto se conoce en otros lenguajes como registro, estructura, diccionario, tabla hash, lista de claves o un arreglo asociativo.
- Una lista ordenada de valores. En la mayoría de los lenguajes, esto se implementa como arrays, vectores, listas o secuencias.

Estas dos estructuras son universales ya que todos los lenguajes de programación las soportan de una forma u otra.

En JSON, se presentan de esta forma:

- Un objeto es un conjunto desordenado de pares nombre/valor. Un objeto se encontrará entre llaves. Cada nombre es seguido por dos puntos y los pares nombre/valor están separados por coma.

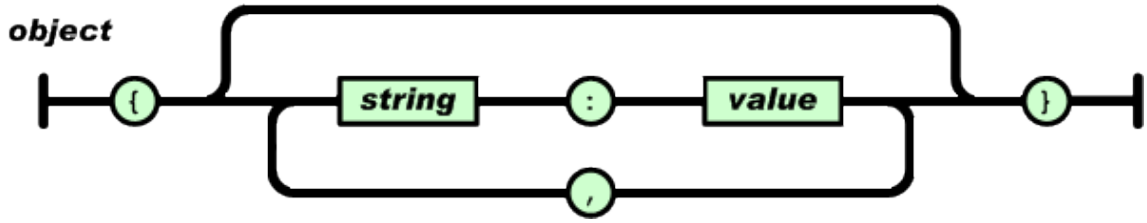


Figura 16.Objeto en JSON

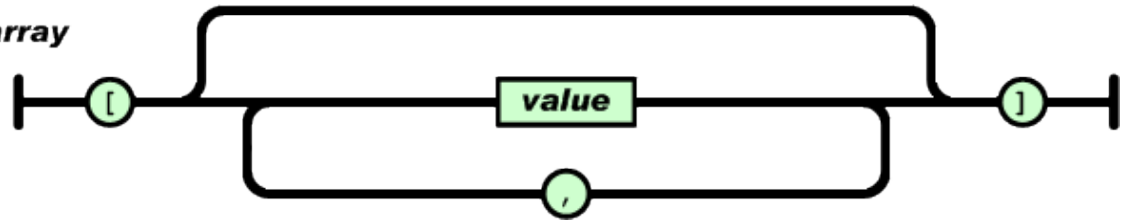
- Un array es una colección de valores. Un array se encuentra entre corchetes y los valores se separan por coma.
- 

Figura 17.Array en JSON

- Un valor puede ser una cadena de caracteres con comillas dobles, un número, un booleano (true o false), null, un objeto o un array. Además estas estructuras pueden anidarse.

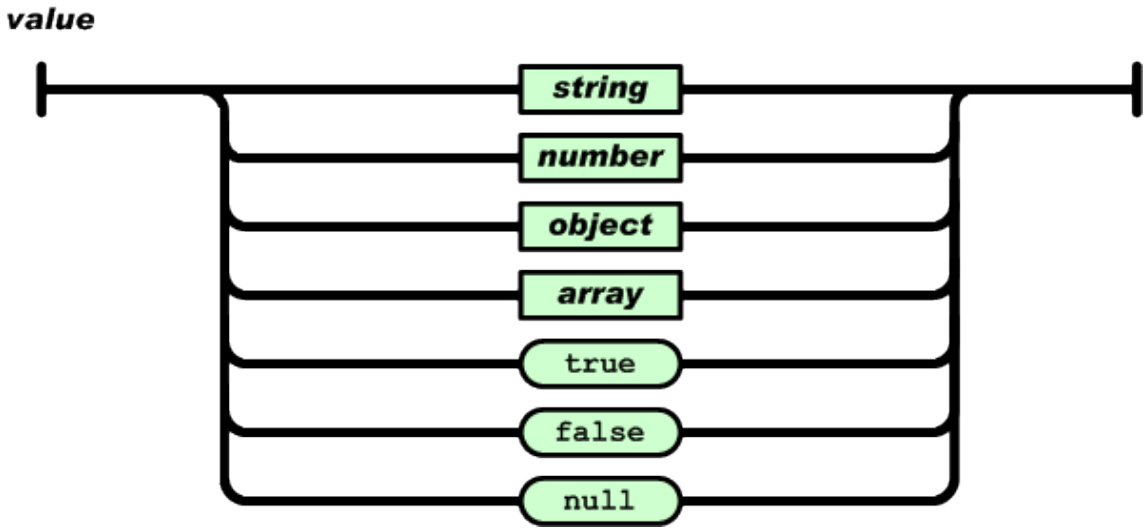


Figura 18.Valor en JSON

- Una cadena de caracteres es una colección de cero o más caracteres Unicode, encerrados entre comillas dobles, usando barras divisorias invertidas como escape, parecida a una cadena de caracteres C o Java. Un carácter está representado por una cadena de caracteres de un único carácter.
- Un número es similar a un número C o Java, excepto que no se usan los formatos octales y hexadecimales.
- Los espacios en blanco pueden insertarse entre cualquier par de símbolos.

Esta simplicidad ha dado lugar a la generalización de su uso, especialmente como alternativa a XML en AJAX (1). Una de las supuestas ventajas de JSON sobre XML como formato de intercambio de datos en este contexto es que es mucho más sencillo escribir un analizador sintáctico (*parser*) de JSON. En JavaScript, un texto JSON se puede analizar fácilmente usando la función *eval()*, lo cual ha sido fundamental para que JSON haya sido aceptado por parte de la comunidad de desarrolladores AJAX, debido a la ubicuidad de intérpretes JavaScript en casi cualquier navegador web.

En la práctica, los argumentos a favor de la facilidad de desarrollo de analizadores o del rendimiento de los mismos son poco relevantes, debido a las cuestiones de seguridad que plantea el uso de *eval()* y el auge del procesamiento nativo de XML incorporado en los navegadores modernos. Por esa razón, JSON se emplea habitualmente en entornos donde el volumen de datos entre cliente y servidor es de vital importancia, cuando la fuente de datos es explícitamente de fiar y donde no es importante el no disponer de procesamiento XSLT para manipular los datos en el cliente.

En este proyecto se ha elegido JSON debido a que el servidor al que se conecta la aplicación usa esta tecnología.

3.2.4 REST

Al igual que se ha utilizado JSON para la representación de datos, REST ha sido impuesto por el servidor al que se harán las peticiones.

REST -*Representational State Transfer*- define una serie de principios arquitectónicos por los cuales se diseñan servicios web haciendo foco en los recursos del sistema, incluyendo cómo se accede al estado de dichos recursos y cómo se transfieren por HTTP hacia clientes escritos en diversos lenguajes.

Originalmente REST estaba referido a un conjunto de principios de arquitectura. En la actualidad se usa en un sentido más amplio para describir cualquier interfaz web simple que utiliza HTTP, sin las abstracciones adicionales de los protocolos basados en patrones de intercambio de mensajes como el protocolo de servicios web SOAP.

Se pueden diseñar sistemas de servicios web de acuerdo con el estilo arquitectural REST y también es posible diseñar interfaces XML/HTTP de acuerdo con el estilo de llamada a procedimiento remoto RPC (aunque RPC no es un ejemplo de REST), pero sin usar SOAP. Estos dos usos diferentes del término REST causan cierta confusión en las discusiones técnicas. Los sistemas que se rigen por los principios REST son llamados normalmente RESTful.

La escalabilidad alcanzada se debe a ciertas claves en su diseño:

- Un protocolo cliente/servidor sin estado: cada mensaje HTTP contiene toda la información necesaria para comprender la petición. Como resultado, ni el cliente ni el servidor necesitan recordar ningún estado de las comunicaciones entre mensajes. Sin embargo, en la práctica, muchas aplicaciones basadas en HTTP utilizan cookies y otros mecanismos para mantener el estado de la sesión (algunas de estas prácticas, como la reescritura de URLs, no son permitidas por REST)
- Un conjunto de operaciones bien definidas que se aplican a todos los recursos de información: HTTP en sí define un conjunto pequeño de operaciones, las más importantes son POST, GET, PUT y DELETE. Con frecuencia estas operaciones se equiparan a las operaciones CRUD en bases de datos (ABMC en castellano: Alta, Baja, Modificación y Consulta) que se requieren para la persistencia de datos, aunque POST no encaja exactamente en este esquema.
- Una sintaxis universal para identificar los recursos. En un sistema REST, cada recurso se puede direccionar únicamente a través de su URI.
- El uso de hipermedios como motor para poder indicar las diferentes acciones. Como resultado de esto, es posible navegar de un recurso REST a muchos otros, simplemente siguiendo enlaces sin requerir el uso de registros u otra infraestructura adicional.

3.2.5 Dispositivos de prueba

Se han realizado pruebas de uso en dos sistemas Android, Nexus 4 y Samsung Galaxy Tab 2 10.1, no se han podido realizar las pruebas en un iPhone ya que se debería pagar la licencia de desarrollador y sería necesario un Mac para compilar la aplicación y subirla al Apple Store.

3.2.5.1 Tablet Android: Samsung Galaxy Tab 2 10.1

Se ha elegido este elemento como principal porque cada día más se avanza al uso de tabletas en el entorno doméstico, tanto en el exterior como en el interior. Por lo tanto se usará para el desarrollo, depuración y puesta en marcha. Se utiliza dicha tableta para conectarse al servidor de prueba y a un

servidor real con WiFi.

La tableta de la que se dispone para el desarrollo del proyecto es una Samsung Galaxy Tab 2 10.1, figura 19.

Figura 19.Samsung Galaxy Tab2 10.1

La tabla 2 recoge las principales características de la tableta.

Pantalla	10,1″ Resolución 1280x800 píxeles TFT PLS
Tamaño	256.6 x 175.3 x 9.7 mm
Peso	587 g
Cámaras	3 MP (principal) 0.3 MP (delantera)
Memoria	16 GB de almacenamiento 1 GB de RAM
CPU	CPU dualcore a 1 GHz
Sensores	Micrófono Acelerómetro Brújula Luz ambiental Barómetro

	Giróscopo GPS
Conexiones inalámbricas	Wi-Fi (802.11 a/b/g/n) Bluetooth
Conectividad	USB 2.0 OTG Conector para auriculares de 3,5 mm
Batería	7000 mAh

Tabla 2.Características Samsung Galaxy Tab2 10.1

3.2.5.2 Smartphone Android: LG Nexus 4

El siguiente elemento necesario para el desarrollo, depuración y puesta en marcha de este proyecto consiste en un Smartphone con Android. Al igual que la tableta se utiliza dicho Smartphone para conectarse al servidor de prueba y a un servidor real tanto en WiFi como en red móvil.

El Smartphone del que se dispone se trata del LG Nexus 4, figura 20.

Figura 20.LG Nexus 4

La tabla 3 recoge las principales características del Smartphone.

Pantalla	4,7″ Resolución 1280x720 píxeles (320dpi) WXGA IPS Gorilla Glass2
Tamaño	133.9 x 68.7 x 9.1 mm
Peso	139 g

Cámaras	8 MP (principal) 1.3 MP (delantera)
Memoria	16 GB de almacenamiento 2 GB de RAM
CPU	Qualcomm Snapdragon S4 Pro 1.5 GHz
Sensores	Micrófono Acelerómetro Brújula Luz ambiental Barómetro Giróscopo GPS
Red	GSM/UMTS/HSPA+ libre GSM/EDGE/GPRS (850, 900, 1800, 1900 MHz) 3G (850, 900, 1700, 1900, 2100 MHz) HSPA+ 42
Conexiones inalámbricas	Compatible con carga sin cables Wi-Fi (802.11 a/b/g/n) NFC (Android Beam) Bluetooth
Conectividad	Micro USB SlimPort HDMI Conector para auriculares de 3,5 mm
Batería	2100 mAh

Tabla 3.Características LG Nexus 4

4 ENTORNO DE DESARROLLO

Tras tener definidos los elementos que se van a usar para en el desarrollo de la aplicación, es necesario explicar cómo se deben instalar y configurar.

El desarrollo del proyecto se ha realizado completamente en Windows, ya que es el sistema al que se tenía acceso. Como ya se expresó anteriormente solo Adobe Air permite crear una aplicación para iOS en un sistema que no sea Mac, por tanto se ha optado por crear una máquina virtual y ejecutar el sistema operativo de Apple en ella para comprobar la compatibilidad de la aplicación en iOS pero no realizando el desarrollo completo de la misma.

4.1 Primeros pasos en Titanium Studio

4.1.1 Descarga e Instalación

Para poder ejecutar Titanium Studio, el sistema debe tener los siguientes requisitos:

- Sistema Operativo: Una versión reciente de Windows, OS X o Ubuntu.
- Memoria: 2 Gb de RAM (Memoria disponible, no memoria total)
- Java Runtime: Oracle JDK.

Para Windows, la versión de 32 bit de Java JDK sirve para que funcione Titanium de 32 bit y de 64bit.

Para descargarse Titanium Studio antes se tiene que crear una cuenta de Appcelerator, esta cuenta es gratuita y nos dará acceso a una tienda de paquetes, gratuitos y de pago, con diferentes funciones ya desarrolladas.

Una vez registrado simplemente habrá que descargarse la versión de Titanium Studio apropiada para el sistema operativo que se vaya a utilizar, figura 21.

Figura 21. Elección de SO para descargar Titanium Studio

En el caso de este proyecto se ha descargado la versión de Windows, una vez finalizada la descarga simplemente se deberá ejecutar el instalador y seguir las instrucciones. El instalador de Windows además descarga e instala Java runtime, figura 22.

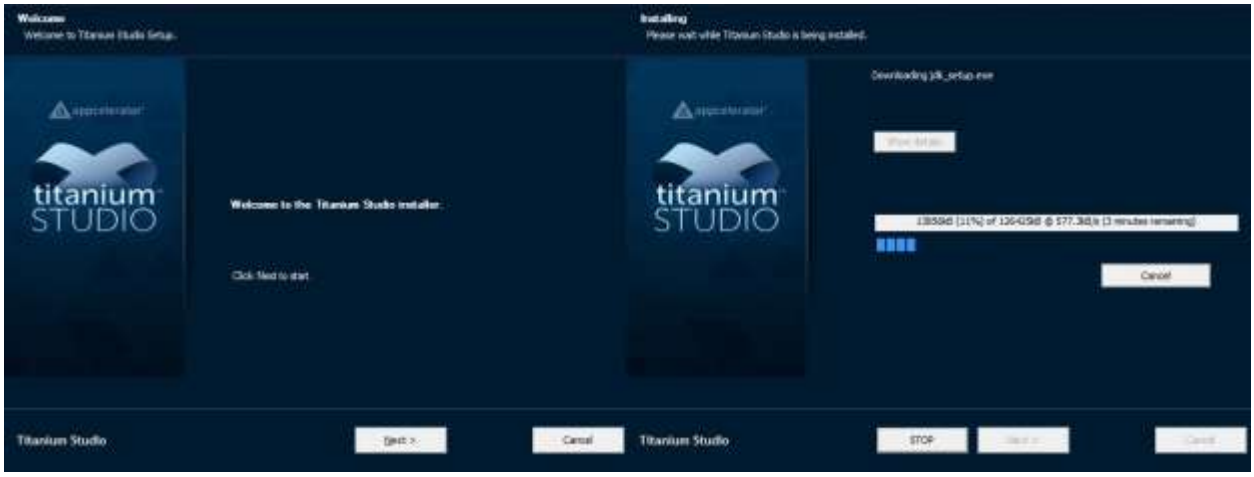


Figura 22.Instalación de Titanium Studio

4.1.2 Primera ejecución de Titanium Studio

Una vez terminada la instalación se entra en Titanium Studio, la primera vez que se inicia Titanium pedirá que se defina un espacio de trabajo, dando la opción de dejarlo como predeterminado o preguntar cada vez que se vuelva a iniciar el espacio de trabajo.

Una vez que se elige el espacio de trabajo se solicitan el usuario y la contraseña con la que anteriormente se hizo el registro en Appcelerator.

Figura 23.Registro de usuario en Titanium Studio

4.1.3 Instalación de los SDKs de las plataformas

Es necesario instalar los SDKs de cada plataforma antes de poder empezar a desarrollar una aplicación para esto existen dos opciones:

- Instalando previamente las herramientas como se explicado en el próximo punto.
- Instalando las herramientas necesarias desde el propio Titanium Studio.

4.1.3.1 Instalación de SDK nativos.

Como primera opción se muestra la instalación del SDK de Android, desde el propio sistema operativo. Google facilita ADT (Android Developer Tools) un plugin para Eclipse que provee un entorno de desarrollo (SDK) profesional para la creación de aplicaciones Android nativas. Es un paquete que incluye un IDE de Java con todas las herramientas necesarias para crear aplicaciones Android.

La instalación de ADT es sencilla:

- Entrar en la página de desarrolladores de Android, dónde se encuentra toda la información para la creación de aplicaciones Android: Desarrollo Android
- Descargar el software para el sistema operativo que se vaya a utilizar. En el caso de este proyecto se ha usado Windows 8 de 64bit, figura 24.

Platform	Package	Size	MD5 Checksum
Windows 32-bit	adt-bundle-windows-x86-20140321.zip	535085536 bytes	b61495a6bf591cc374c31bce4fc46ec0
Windows 64-bit	adt-bundle-windows-x86_64-20140321.zip	535287324 bytes	a6f4699bbdc5a29b371ed60610535651
Mac OS X 64-bit	adt-bundle-mac-x86_64-20140321.zip	501955296 bytes	4a08649cea9b098cdf7349f452294014
Linux 32-bit	adt-bundle-linux-x86-20140321.zip	527971926 bytes	943ae4d28fe7c79108c8bf2aafd5e6d2
Linux 64-bit	adt-bundle-linux-x86_64-20140321.zip	528187678 bytes	f2a2153b5c7dbaeb86b550bf4f770c36

Figura 24.Paquete de instalación ADT de Android

- Una vez descargado solo se tendrá que descomprimir en el directorio deseado.

4.1.3.2 Instalación de SDKs desde Titanium Studio

Para instalarlo directamente desde Titanium, la primera vez que se inicia se comprueba las plataformas que permite el propio sistema donde se está trabajando y aparece un dialogo de configuración.

El diálogo que aparece indica qué plataformas están instaladas y cuáles no. Aunque en un primer momento no se desee configurar siempre al iniciar Titanium aparecerá una pantalla principal dónde se podrán configurar nuevos SDKs y comprobar los ya instalados, figura 25.

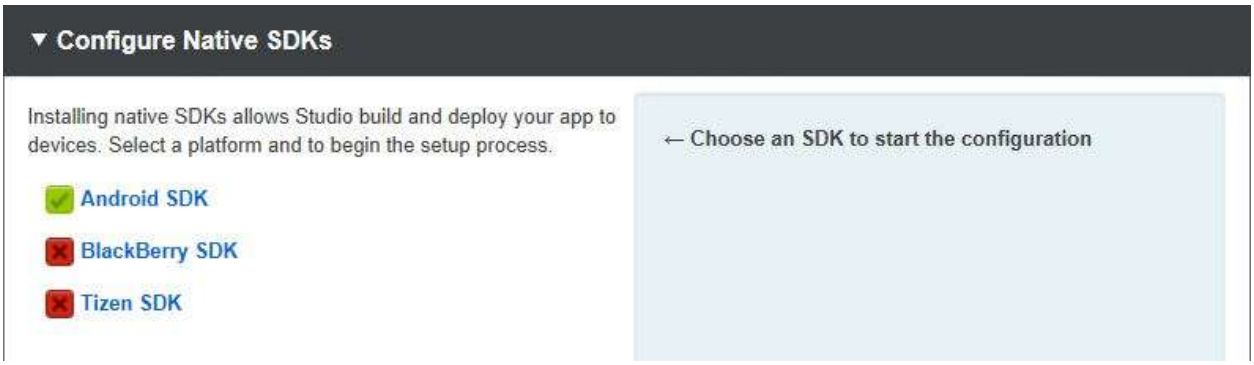


Figura 25.Configuración de SDKs Nativos en Titanium Studio

Las configuraciones de cada plataforma son transparentes, Titanium Studio descarga el software y automáticamente lo instala y lo configura.

4.1.4 Creación de un proyecto

Para empezar un nuevo proyecto en la pantalla que aparece inicialmente que Titanium llama ***Dashboard***, es una pantalla principal que te dan acciones de configuración, se elegirá ***Develop > Default Project***, figura 26. Existen más opciones pero al desarrollar una aplicación desde cero es preferible comenzar con un proyecto vacío.

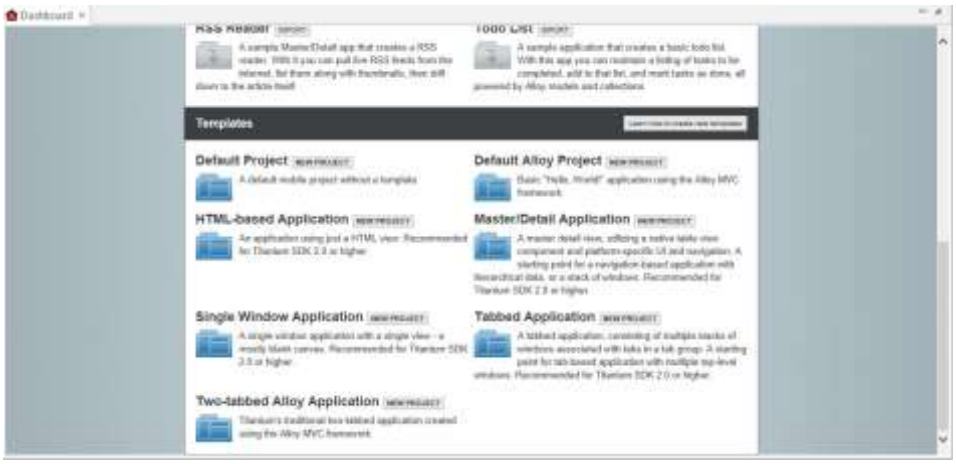


Figura 26.Pantalla de menú del Titanium Studio

Tras crear un nuevo proyecto aparece una ventana de configuración donde se define, figura 27:

- El nombre del proyecto.
- El identificador de la aplicación: este parámetro se usa para cuando se publique la aplicación.
- URL: Este parámetro es opcional, se puede indicar una URL por si es una empresa o un desarrollador, también se puede indicar para contactar con el desarrollador por posibles fallos.
- Versión del SDK de Titanium: para poder elegir una versión concreta cuando hay varias instaladas. La aplicación que se ha realizado se ha usado la versión 3.2.1GA.
- Plataformas: Permite elegir las plataformas para las que se va a desarrollar la aplicación.

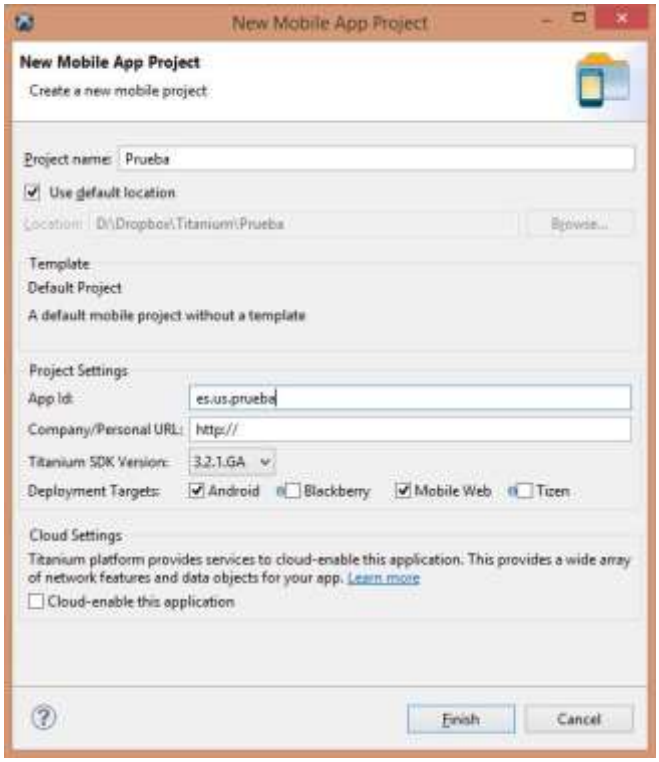


Figura 27.Configuración de un nuevo proyecto en Titanium Studio

A continuación se pulsará finalizar y aparecerá la carpeta del proyecto, con el archivo de configuración, llamado ***tiapp.xml***, que podrá visualizarse de dos formas, figura 28:

- En formato xml
- En formato gráfico

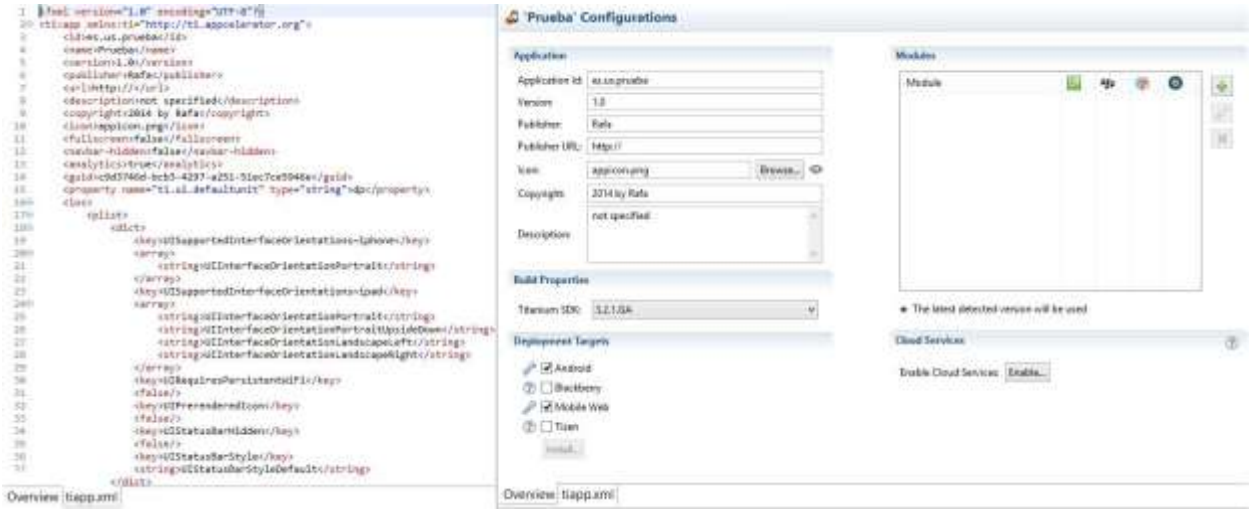


Figura 28.Archivo de configuración: tiapp.xml

Este archivo de configuración permite realizar cambios en las plataformas de desarrollo, introducir módulos para usar APIs del sistema nativo, o configurar temas visuales. También permite cambiar parámetros para la publicación de la aplicación final.

Además del fichero de configuración se crean un archivo para redactar la licencia de uso de la aplicación y el archivo principal de la aplicación llamado ***app.js***.

A partir de este momento ya se puede comenzar a programar.

4.2 Configuración Smartphone y Tablet

Android permite a los desarrolladores usar su dispositivo para realizar pruebas de las aplicaciones creadas.

Para poder usar todas las opciones relacionadas con la depuración en el dispositivo, las entradas al sistema y el control de las aplicaciones desde la versión Ice Cream Sandwich apareció una nueva opción en el menú de Ajustes llamada Opciones de desarrollo. En esta sección se pueden activar o desactivar aspectos del teléfono para que determinadas aplicaciones funcionen a pleno rendimiento o para que puedan acceder a ciertas zonas del sistema.

En la figura 29 se pueden ver las opciones de desarrollo disponibles que se explican a continuación.

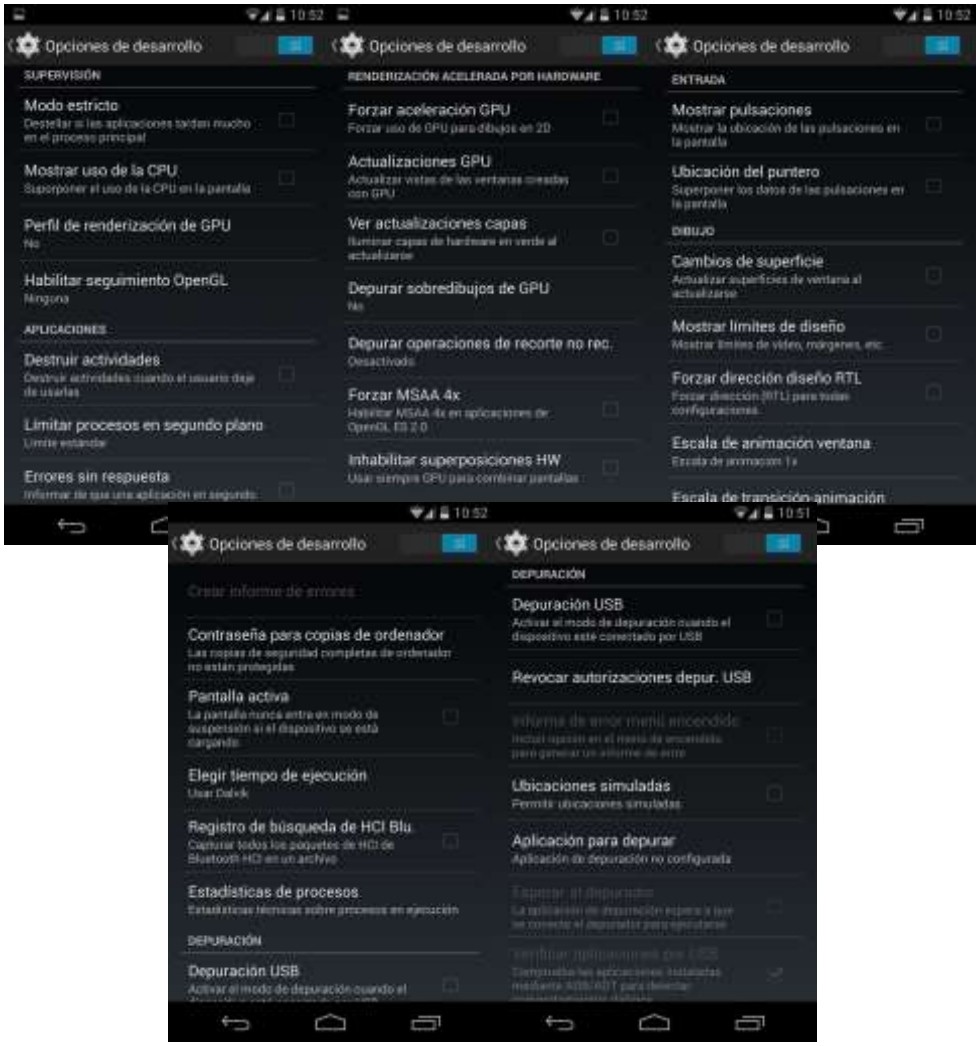


Figura 29.Menú de configuración de depuración en Android

- Contraseñas para copias de ordenador: Permite hacer una copia de seguridad completa del teléfono en el ordenador. Pedirá una contraseña para ejecutar la copia. Esta opción es necesaria para ejecutar ciertos programas.
- Pantalla activa: Permite desactivar el apagado automático de la pantalla, cuando el teléfono esté en carga o esté ejecutando procesos en primer plano.
- Depuración USB: Permite que algunas aplicaciones accedan a todo el sistema, así como acceder a ciertos aspectos del sistema en modo depuración.
- Aplicación para depurar: Por si se quiere establecer una aplicación para depurar.
- Mostrar pulsaciones / Ubicación del puntero: Sirve para ver si el terminal reconoce exactamente el punto donde se pone el dedo en la pantalla. Las pulsaciones se mostrarán como un círculo azul y, en la parte superior de la pantalla, saldrán las coordenadas del puntero. Una opción muy útil para ver si la pantalla reconoce bien las pulsaciones y conocer las coordenadas en la pantalla de éstas.
- Mostrar límites de diseño: Si se activa, se verán todos los márgenes de los elementos de la pantalla.
- Actualizaciones GPU *(Unidad de procesamiento gráfico)*: Actualizará las vistas de las ventanas creadas con GPU cada vez que se abran o se cambie la posición de la pantalla. Esto ayudará, en algunos casos, a aliviar la carga del procesador.
- Forzar aceleración GPU: Mejora el rendimiento del sistema bastante al dar mayor trabajo a la GPU y liberar la CPU.
- Cambios de superficie: Cada vez que el sistema reinicie una parte de la pantalla, ésta se reiniciará. Si se activa la opción se verá cómo el teléfono se llena de colores y cambian rápidamente debido a lo citado anteriormente, no es recomendable activarla.
- Escalas varias: Simplemente la escala a la que se quieren ver las distintas opciones que se detallan. En ella se podrán alterar las velocidades a las que se muestra una ventana emergente, por ejemplo. Si se pone la escala de animación de ventana a 10x, esta ventana se abrirá mucho más lento.
- Inhabilitar superposiciones HW: Activa la GPU para que se encargue de la combinación de las distintas pantallas.
- Modo estricto: Si una aplicación se queda colgada, o tarda mucho en desarrollar su labor, destellará en la pantalla principal y se podrá cerrar sin problema.
- Mostrar uso de la CPU: Mostrará los valores del uso de la CPU que tenga el terminal en cada momento en un lateral de la pantalla.
- Habilitar seguimientos OpenGL: Se podrán Destruir actividades, activar el seguimiento de ciertos procesos del sistema, como el View, el Activity Manager o el Sync Manager. Esta opción permite al sistema destruir los procesos inactivos, es decir, matará las aplicaciones en cuanto se dejen de usar.

- Limitar procesos en segundo plano: Permite determinar el número de procesos en segundo plano que se le permite al sistema ejecutar al mismo tiempo. La variación va desde cero hasta cuatro. Aunque existe la posibilidad de dejar un límite estándar, que es el que el sistema considera oportuno en cada momento.
- Errores sin respuesta: Si está activo informará, con una notificación emergente, siempre que una aplicación no responda.

5 DESARROLLO DE LA APLICACIÓN

5.1 Introducción

En este proyecto fin de carrera se ha desarrollado una aplicación móvil con el objetivo de permitir al usuario gestionar de manera sencilla las reservas que se realizan en un aparcamiento para vehículos eléctricos.

Las características más importantes de la aplicación son:

- Posicionamiento del usuario y relación de su ubicación con los distintos Nodos de recarga.
- Información de cada Nodo y los servicios que ofrece.
- Gestión de reservas:
    - Realizar una reserva.
    - Anular una reserva.
    - Consultar reservas.
        - Usadas
        - Anuladas
        - Próximas reservas.
- Consultar recibos.

Para analizar la estructura de la aplicación y el funcionamiento de cada una de las características anteriores con detalle, vamos a hacer una separación por funcionalidades básicas:

- Comunicación con el servidor.
- Diseño de las interfaces de usuario de la aplicación.

Antes de desarrollar cada punto, hay que comentar que la aplicación está desarrollada para el sistema operativo Android. La razón es simple, la aplicación partió como un proyecto multiplataforma, aunque se han realizado pruebas en un simulador de iOS no se ha podido trabajar en esta plataforma debido a la carencia del equipo necesario para desarrollar la aplicación en ambas plataformas y se ha tenido que recurrir a una máquina virtual con el sistema operativo de Mac para ver su funcionamiento. Por esta misma razón, no debe extrañar al lector que en lo sucesivo se dé mayor importancia a la descripción de la implementación para la plataforma Android que para iOS.

5.2 Comunicación con el servidor

La comunicación de la aplicación con el servidor SRG –*Sistema de Reserva Guiado*- se hace a través de RESTful, para dicha comunicación se van a usar dos peticiones:

- GET: Para consultar y leer recursos

- POST: Para crear recursos

En la aplicación que se ha realizado se han utilizado tres funciones para realizar las peticiones al servidor:

- Tipo GET: ***consulta_abonado.js*** y ***anular_reserva.js***.
- Tipo POST: ***postConexion.js***.

Todas las comunicaciones que se realizan en la aplicación sirven para obtener la información que solicita el usuario, el GET se usará para realizar confirmación de acciones que lleva a cabo el usuario y el POST se usará cuando el usuario haga peticiones al servidor.

5.2.1 Formato de envío y recepción de datos

Para el envío de los datos hay que convertirlos primero a JSON. En la aplicación se ha decidido crear objetos que tengan como propiedades los campos que se tienen que enviar en JSON, como se ve en la figura 30.

```
//Formato datos de envio
var datosConsultaReserva = {
      IdUsuario: id_usuario,
       Matricula: matricula,
       FechaDesde: fechaInicio,
       FechaHasta: fechaFin,
       Usada: Usada,
       Valida: Valida,
};
```

Figura 30.Código formato datos de envío

Esta forma de representar los datos hace más sencillo su acceso a través del código y su conversión para posteriormente enviarlos.

Para convertirlos a JSON simplemente se tendrá que usar el objeto JSON. El objeto JSON está definido en las librerías de Javascript.

El objeto JSON contiene métodos para analizar los datos codificados en JSON -*JavaScript Object Notation*- y para la conversión de los valores a JSON. Los dos métodos hacen que la conversión de datos de un objeto JavaScript a JSON o de JSON a un objeto a Javascript sea inmediata. Sus dos métodos se pueden ver en la figura 31.

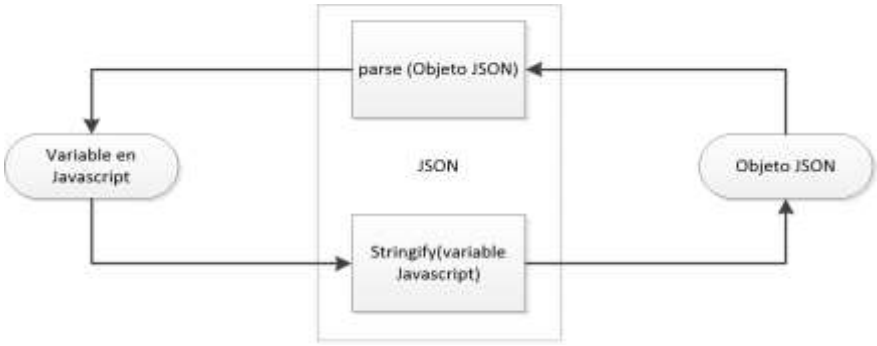


Figura 31.Objeto JSON

JSON es una sintaxis para serializar objetos, matrices, números, cadenas, booleanos y NULL.

Para convertir un objeto Javascript a JSON simplemente se tendrá que usar un método de la clase JSON, ***JSON.stringify()***, que da como resultado un JSON, y al que se le pasa como parámetro ***datosConsultaReserva***, el objeto de Javascript, en el caso en que se quiera hacer una reserva.

Al igual que el envío de datos, durante la ejecución de la aplicación se reciben respuestas que deben ser convertidas a una variable de Javascript desde JSON, se usará el método ***parse*** de la clase JSON para hacer el procedimiento inverso.

Tras recibir la respuesta del SRG, el JSON que se recibe se convierte y se guarda en el objeto de Javascript ***datosRecibidos,*** en el caso que la respuesta sea positiva, o ***datosRechazados,*** en el caso en que la respuesta sea negativa. La respuesta se devuelve a la función que ha llamado a la función de conexión, figura 32.

```
//Creo un objeto de consulta.
objconsulta = new consulta_abonado(args.user,args.uri,function(dato){
     idUsuario = dato.IdAbonado;
```

Figura 32.Llamada a la función consulta abonado

Se puede acceder a un campo del objeto JSON convertido a un objeto Javascript como si fuera una propiedad del objeto Javascript, un ejemplo se puede ver en la figura 32 cuando el objeto *dato* tiene como propiedad ***IdAbonado***. El objeto JSON que se ha recibido es el que se muestra en la figura 33.

```
        --- Abonado ---
{
        IdAbonado : "123456"
}
```

Figura 33.JSON "Consulta de Abonado"

5.2.2 URIs

Son las direcciones que necesita la aplicación para comunicarse con el servidor. Las URIs cambian según la petición que se quiera realizar en cada momento, por esta razón se decidió en el

desarrollo de la misma que todas las URIs comunes fueran incluidas en una variable de Javascript, figura 34.

```
//Objeto que almacena las direcciones para conectarme al servidor.
var uri = {
        uri_base: 'http://subsistemareservaguiado.appspot.com/IHM/',
        reserva: 'UserReserve',
        infoReserva: 'UserInfoReserve',
        plazas: 'RequestPlaces',
        anular: 'NullReserve/',
        usuario: 'suscriber/',
};
```

Figura 34.Variable JavaScript URIs

De esta forma se consigue que el código sea más independiente por si en el futuro se tuviera que cambiar algunas de las URIs.

La unión de la uri_base con cada URI muestran la información que podrá obtener el usuario.

- Para realizar una reserva se usará ***uri. uri_base+uri.reserva***.
- Para consultar reservas se usará ***uri. uri_base+uri.infoReserva***.
- Para consultar las plazas que existen para un momento determinado se usará ***uri. uri_base+uri.plazas***.
- El usuario podrá anular una reserva con ***uri. uri_base+uri.anular***.
- La aplicación podrá consultar si el usuario está abonado con ***uri. uri_base+ uri.suscriber/***.

5.2.3 Definición de una conexión

Para crear una conexión es necesario hacer uso de las librerías de Titanium, para crear la variable Javascript ***connectConsulta***, figura 35. Con la librería Titanium.Network se crea la conexión, en el caso de la comunicación con el servidor se necesita crear una conexión HTTP. La conexión tendrá dos propiedades imprescindibles:

- ***onload***: Esta propiedad define las acciones que se llevarán a cabo si la respuesta ha sido afirmativa. En el caso de la aplicación desarrollada se convertirán los datos recibidos en JSON a una variable Javascript o se leerá el estado de la comunicación.
- ***onerror***: Al contrario que ***onload*** esta propiedad define las acciones en caso de error. También en este caso se leerá el estado de la comunicación o si llega un objeto JSON se convertirá a un objeto Javascript y se procederá a su interpretación.

```
//Definición de una conexión para peticiones http
var connectConsulta = Titanium.Network.createHTTPClient({
// Llamamos a la funcion cuando la respuesta se ha producido
          onload : function(e) {
                    //Llamo al evento para que analice los datos recibidos
                    Titanium.API.info(this.responseText);
                    var idAbonado = JSON.parse(this.responseText);
                    //Devuelve el valor para comprobar si se ha anulado correctamente
                    callback(idAbonado);
          },
          // Se llama a la funcion error cuando se produce un error o se acaba el timeout
          onerror : function(e) {
                    alert('No se ha podido realizar la consulta de matrícula abonada');
          },
          timeout : 30000 // En milisegundos
});
```

Figura 35.Definición de una conexión para peticiones http

5.2.4 Petición GET

Las peticiones realizadas con GET se componen con una URI, será la unión de la ***uri_base*** con otros elementos, como por ejemplo el caso de consulta abonado:

'http://subsistemareservaguiado.appspot.com/IHM/suscriber/1234ABC'

La respuesta no tiene que ser una respuesta simple, por ejemplo 200OK o 404, podrá llevar asociada un objeto JSON.

Esta petición es la más sencilla de las dos, y sólo se tendrá que hacer uso de dos métodos del objeto ***HTTPClient***, figura 36.

```
//Conexión
// Preparación de la conexión.
connectConsulta.open("GET", uri.uri_base + uri.usuario + matricula);
Titanium.API.info(uri.uri_base + uri.usuario + matricula);
// Envio de la petición.
connectConsulta.send();
```

Figura 36.Petición GET

Como se ve en la figura 36 se usa el método ***open***, donde se define la petición que se desea realizar, GET en este caso, y la URI del servidor a la que enviar la petición.

En un segundo paso se llama al método ***send*** para enviar la petición. Tras el envío de la petición se queda a la espera de recibir la respuesta por parte del servidor. Para el procesamiento de la respuesta se usarán las propiedades ***onload*** u ***onerror*** según proceda.

5.2.5 Petición POST

Esta petición está compuesta por la URI del servidor y un objeto JSON. Para la respuesta ocurre lo mismo que en el caso de la petición GET, el cliente puede recibir una respuesta simple, 200OK o 404, o un objeto JSON.

Esta petición tiene los métodos ***open*** y ***send*** pero necesitará un método intermedio, como se ve en el código de la figura 37.

```
//Conexión
// Preparación de la conexión.
conexionPost.open("POST", uri_base + uri_complemento);
Titanium.API.info(uri_base + uri_complemento);
// Envio de la petición.
//Definición del tipo de datos que se envían
conexionPost.setRequestHeader("Content-Type", "application/json; charset=utf-8");
datos_env_JSON=JSON.stringify(datos_env);
Titanium.API.info(datos_env);
conexionPost.send(datos_env);
```

Figura 37.Conexión POST

Como se ve en la figura 37, el objeto Javascript ***datos_env*** se convierte a un objeto JSON y se almacena para posteriormente enviarlo al servidor.

El método ***setRequestHeader***, del objeto ***Titanium.Network.HTTPClient***, tiene dos parámetros, ***Content-Type*** que lleva los datos de la cabecera y ***aplication/json; charset=utf-8*** que define como van codificados los datos que se envían. Este método siempre se tendrá que invocar después del método ***open***.

A diferencia que en la petición GET para la petición POST se tendrán que incluir los datos que se envían como parámetro del método ***send***.

La respuesta por parte del servidor es gestionada igual que en la petición GET.

5.3 Diseño de la aplicación

La aplicación se basa en ventanas, el usuario podrá visualizar información o realizar diferentes acciones en cada ventana. Para conseguir este fin se han creado diferentes ventanas, y cada ventana irá asociada a un fichero diferente para una mejor organización de código, como se ve en la figura 38.

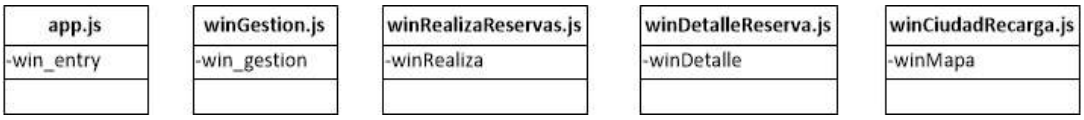


Figura 38.Archivos de la aplicación

A continuación se va a detallar cada fichero y su ventana asociada, explicando el contenido que se presente en cada una de ellas.

5.3.1 App.js

Es el fichero que comienza a leer la máquina virtual de Javascript de cada plataforma. Este fichero siempre está presente cuando se crea un nuevo proyecto.

En este fichero se abre una ventana por defecto. Si no existe una matrícula o usuario almacenado abrirá una ventana que permite al usuario registrarse, sino se abrirá directamente la ventana de ***win_gestion***.

Figura 39.Win_entry

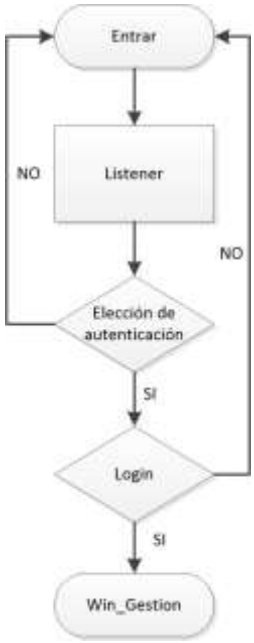


Figura 40.Diagrama de flujo de identificación

Para crear una ventana Titanium posee una librería propia, ***Titanium.UI***, que permite incluir elementos como una ventana, botones, cuadros de texto o similar.

Una ventana es un objeto de Javascript, el desarrollador podrá definir unas propiedades para el estilo. La ventana ***win_entry*** se define en el siguiente código, figura 41.

```
//Ventana de bienvenida donde el usuario se registra
var win_entry = Titanium.UI.createWindow({
        tittle:'Win_entry',
        backgroundImage: '/fondo.png'
});
```

Figura 41.Código para crear la ventana win_entry

El código muestra cómo crear una ventana con dos propiedades, el nombre y una imagen para el fondo, cabe la posibilidad de definir el tamaño, por defecto la ventana ocupa toda la pantalla, o colores para el fondo (entre otras propiedades).

La ventana para autenticarse, ***win_entry***, presenta al usuario las opciones que tiene para identificarse y poder acceder a las capacidades de la aplicación. Puede hacerlo mediante su identificador de usuario o bien mediante un número de matrícula, en la versión que se ha desarrollado no se ha introducido ningún método de autenticación ya que el servidor no lo solicita, por lo que la seguridad sería a nivel local. Para esta lista se crea con un elemento ***picker*** y se añade una matriz de datos que contienen las opciones, en la figura 42 se puede ver el código.

Una vez elegido el método de autenticación se escribirá la identificación de usuario o la matrícula en el cuadro de texto.

```
//Cuadro de diálogo para elegir si se realiza la entrada al
sistema con Usuario o Matrícula
var pickerIdentificacion = Ti.UI.createPicker({
          backgroundColor: 'white',
          bottom: '75%',
});

//Opciones posibles
var data = [];
          data[0]=Ti.UI.createPickerRow({title:'Elige una
opción'});
          data[1]=Ti.UI.createPickerRow({title:'Id. Usuario'});
          data[2]=Ti.UI.createPickerRow({title:'Matrícula'});
pickerIdentificacion.add(data);
pickerIdentificacion.selectionIndicator = true;
```

Figura 42.Lista de Identificación

Una vez que el usuario ha escrito una de las dos opciones, al pulsar el botón Entrar se lanza un evento, definido en el código de la figura 43, el cuál llama a una función dónde se comprueba si el usuario es abonado, en caso negativo aparece una mensaje, para este mensaje se usa un cuadro de diálogo con el comando ***alert***.

```
//Acciones al pulsar el botón de entrada
button_enter.addEventListener('click',function(){
        var input_number = Ti.UI.createTextField();
        var dialogCarga = Ti.UI.createOptionDialog();
…
                fbutton_enter({user: box_user.value, uri: uri, identificacion: box_user.hintText});
        };
…
```

Figura 43.Evento que se lanza tras pulsar el botón entrar

En la figura 43 se llama a la función ***fbutton_enter*** y se le pasan como parámetros:

- El valor capturado en el cuadro de texto de la identificación.
- El objeto JavaScript que contiene las URIs.
- El tipo de identificación que el usuario ha elegido.

Si hay éxito se procede a cerrar la ventana de autenticación, ***win_entry***, y se abre una nueva ventana, ***win_gestion***.

5.3.2 winGestion.js

Este fichero crea la ventana principal, ***win_gestion***, en esta ventana se muestran los datos que solicite el usuario en cada momento seleccionando la opción en el menú, este menú será deslizante –menú slider -, se explica en el Anexo II, figura 44, por el que se podrá navegar por las diferentes

opciones, una vez elegida una opción se cargará en la ventana, ***win_gestion***, la vista de la opción elegida, la creación de una vista se explica en el Anexo II.

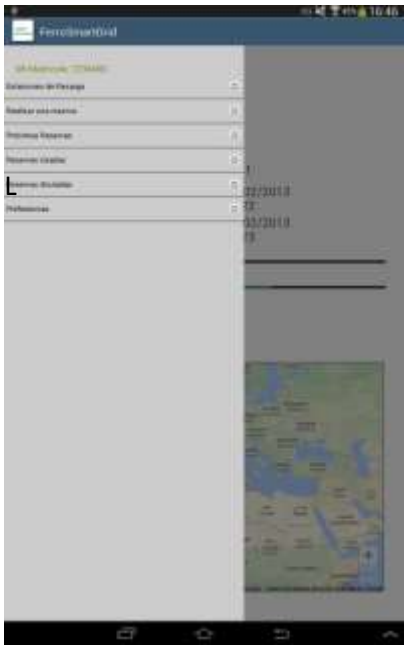


Figura 44.Menú

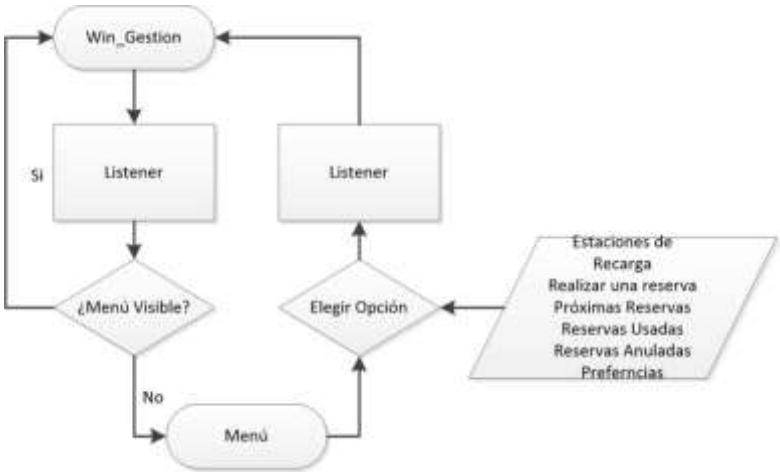


Figura 45.Diagrama de flujo del menú

La ventana ***win_gestion*** se ha definido en blanco, dónde se presentarán diferentes vistas, Appcelerator permite añadir a las ventanas vistas –***view***-, cuadro de textos, tablas, imágenes, incluso mapas. Todo lo que se crea deberá ser añadido después a la ventana en la que se está trabajando para que el usuario pueda visualizarlo. Para añadir un nuevo objeto se tendrá que usar el método ***add*** como se ve en la figura 46.

```
win_gestion.add(viewInformacion);
win_gestion.add(sombra);
win_gestion.add(menuView);
```

Figura 46.Método add

En la figura 46 se ve que se añade el menú y ***viewInformacion***, la vista que mostrará la interfaz elegida en el menú. Para la aplicación se han creado las siguientes vistas, cada una asociada a un fichero.

5.3.2.1 viewBienvenida

La primera vez que se accede a la aplicación y tras identificarse se mostrará en la ventana ***win_gestion*** una vista de la próxima reserva del usuario, mostrando los datos más importantes, figura 47.

Para crear los elementos se utilizan las bibliotecas de Titanium, y los datos se toman de la respuesta recibida del servidor tras una petición (de tipo POST).

Figura 47.viewBienvenida

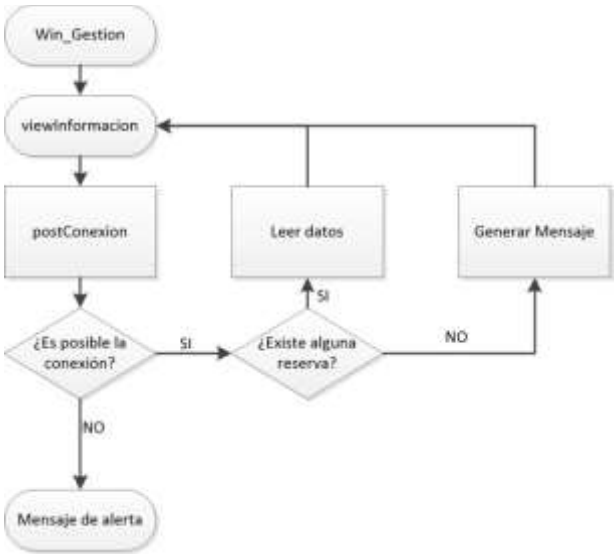


Figura 48.Diagrama de flujo de viewBienvenida

Las demás vistas de la aplicación son llamadas por un evento ***listener*** cuando se pulsa una de las opciones de menú, es decir, recoge la pulsación sobre el ítem del menú.

5.3.2.2 viewTodasReservas

Esta vista representa tres vistas del menú, 'Próximas Reservas', 'Reservas Usadas', 'Reservas Anuladas'. Se muestra una tabla en la que, cada fila de la tabla contendrá una reserva con unos datos que se han solicitado al servidor por una petición de tipo POST, figura 49.

Figura 49.viewTodasReservas

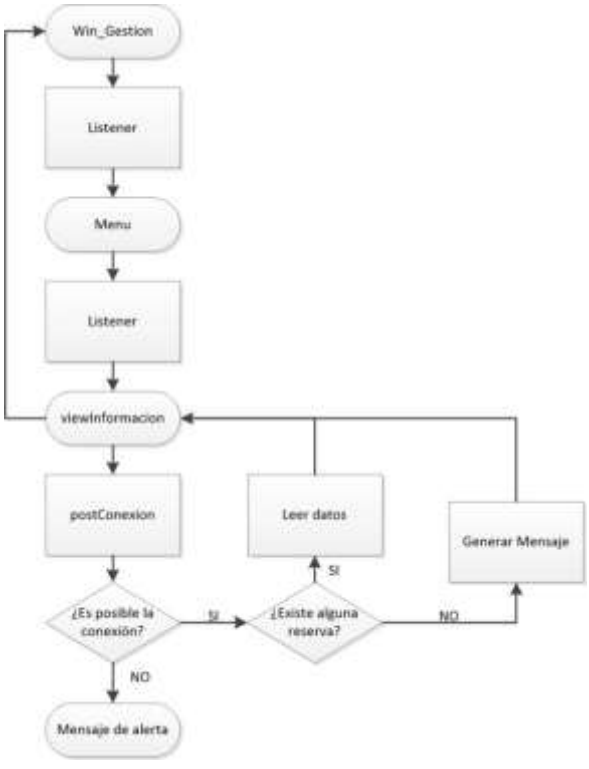


Figura 50.Diagrama de flujo de viewTodasReservas

En la figura 51 se puede ver la petición para que solicitar información sobre las reservas del usuario. Esta petición viene definida principalmente por el par Usada/Valida:

- ***Reservas Usadas***: Usada y Valida a ***true***, mostrará las reservas que se han utilizado.
- ***Reservas Anuladas***: Usada y Valida a ***false***, muestra todas las reservas que el usuario realizó pero que luego anuló.
- ***Próximas Reservas***: Usada a ***false*** y Valida a ***true***, da las próximas reservas que tiene el usuario para las fechas posteriores a la fecha actual.
- El par ***true/false*** es inviable por que no puede existir una reserva usada e invalida.

Para el objeto JSON se crea una variable Javascript, los valores se pasan como argumentos cuando se llama a la función ***viewTodasReservas***, figura 47.

```
viewInformacion = new
viewTodasReservas(args.matricula,args.id_usuario,args.uri,
"010119990000","311220992359",true,true,e.rowData.title);
```

Figura 51.Función para ver las reservas de un usuario

Estos valores son almacenados junto al campo correspondiente y posteriormente en la petición POST se convertirán a un objeto JSON.

En la versión final que se ha creado las fechas son constantes ya que las consultas que se realizan no se realizan usando un filtro de fecha. Esta sería una línea futura para continuar con su desarrollo.

```
function
viewTodasReservas(matricula,id_usuario,uri,fechaInicio,fechaFin,Usada,Val
ida,reserva){

…

//Formato datos de envio
var datosConsultaReserva = {
          IdUsuario: id_usuario,
          Matricula: matricula,
          FechaDesde: fechaInicio,
          FechaHasta: fechaFin,
          Usada: Usada,
        Valida: Valida,
};
```

Figura 52.Formato de datos de envío de la petición de reservas

Tras recibir la respuesta y añadir todos los elementos, la vista se devuelve a la función de ***win_Gestion*** y se añade a la ventana para que el usuario pueda ver el resultado final.

Una vez creada la tabla con todas las reservas el usuario tiene la posibilidad de acceder a cada reserva por separado, cada línea de la tabla estará esperando, con el evento ***listener***, para ejecutar una nueva ventana. Para generar esta nueva ventana se usa la función ***winDetalleReserva***. Esta ventana

mostrará la información de la reserva de una manera más detallada y permitirá al usuario anular la reserva o ver el recibo de la reserva si el usuario ha selecciona ver ***Próximas Reservas*** o ***Reservas Usadas*** respectivamente, figura 53.

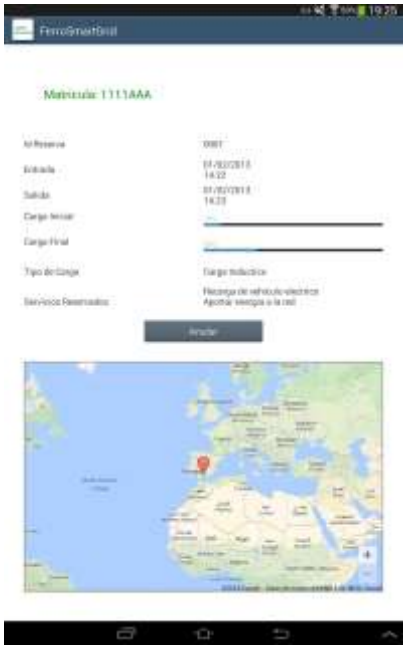


Figura 53.winDetalleReservas

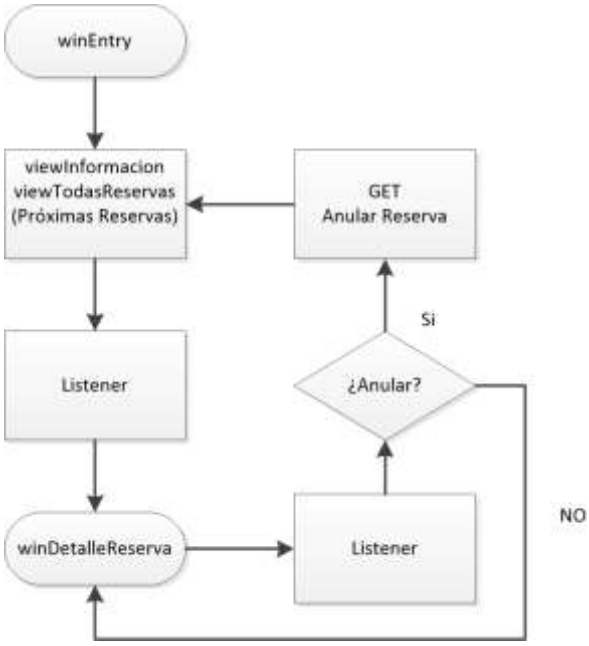


Figura 54.Diagrama de flujo winDetalleReserva

En el caso de estar en *Reservas Usadas* si se eligiera una reserva el usuario podría ver el recibo, para este caso no es necesario hacer ninguna petición al servidor ya que los datos se recibieron cuando se realizó la petición para ver las reservas usadas. En cambio sí es el caso de ***Próximas Reservas*** y se anula una reserva se hará una petición de tipo POST al servidor.

5.3.2.3 viewEstacionesRecarga

Para la creación de esta vista se ha creado una vista del mapa con la librería ***ti.Map***, esta librería la proporciona Appcelerator además de incluir una vista de Google Maps en la aplicación con ***Map.createView***.

En esta vista se representan los **n** nodos de recarga en el mapa que están a **x** kilómetros, la distancia en kilómetros se puede indicar en la vista de preferencias, de la posición del usuario. La aplicación usa la API nativa del GPS para ubicar al usuario.

Cada nodo se marca en el mapa creando un objeto ***Annotation***, como se ve en el código de la figura 50, en este objeto se han definido dos propiedades nuevas:

- ***Servicios***: Almacena los servicios que tendrá ese nodo.
- ***IdNodo***: Almacena el identificador de nodo.

```
function creaPunto
(latitud,longitud,servicios,idnodo){
      var puntoIdNodo = Map.createAnnotation({
               latitude:latitud,
               longitude:longitud,
            pincolor:Map.ANNOTATION_BLUE,
               Servicio:servicios,
               IdNodo: idnodo
      });

      …
```

Figura 55.Código para crear el punto donde se encuentra el nodo

Cuando se pulse sobre un nodo en el mapa aparecerá un cuadro de diálogo mostrando los servicios que ofrece ese nodo. Además se ha incluido la posibilidad de que el usuario seleccione un nodo, seleccione los servicios que desea y vaya a la vista de Reservas para realizar una reserva nueva.

Figura 56.Vista del mapa de Estaciones de Reserva

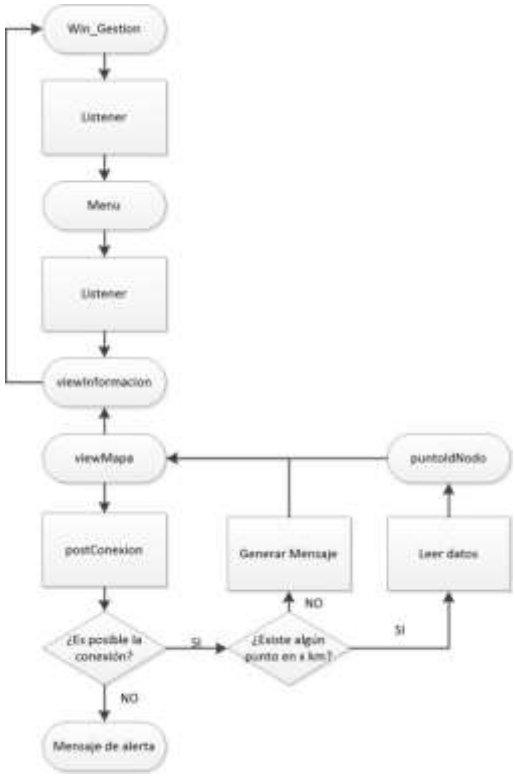


Figura 57.Diagrama de flujo de viewEstacionesRecarga

5.3.2.4 viewRealizaReserva

A esta vista se puede acceder desde el menú de opciones o desde ***viewEstaciones***. Esta vista tiene por objetivo definir los parámetros de recarga durante el tiempo de estancia en el nodo.

El usuario puede elegir entre dos métodos para realizar la reserva de recarga:

- Carga a recargar: Es la carga que desea el usuario recargar durante la estancia en el nodo.
- Precio: Es lo que el usuario quiere recargar durante la reserva, pero definido por unidades monetarias. El precio es la opción que siempre se ha utilizado en el repostaje de combustible, por ello se ha decido que es una opción para mantenerla

Al usuario se le permite también elegir las fechas de llegada y salida, como se ve en la figura 58 y figura 59.

Los últimos tres campos son:

- Tipo de recarga: Se tienen tres posibilidades diferentes para realizar la recarga.
    - Carga Inductiva
    - Carga Capacitiva
    - Recarga Rápida
- El usuario tiene la opción de pagar al retirar el vehículo (Pago Insitu) o se le pasará el recibo a su cuenta personal.

- Por último, en el caso de que se haya seleccionado la opción Realizar Reserva en el menú, el usuario seleccionará uno de los dos métodos para buscar un punto de recarga, figura 58.
    - Por ciudad.
    - Por cercanía.

Figura 58.Vista Realiza Reserva sin nodo especificado

Figura 59.Vista Realiza Reserva

Cuando se accede la búsqueda del punto de recarga, la aplicación crea una nueva ventana, ***winCiudadRecarga***, en esta nueva ventana se crea un mapa, los nodos que se muestran en el mapa serán los nodos disponibles para los parámetros configurados en la reserva.

Los puntos de recarga se representarán en el mapa, y al igual que ocurría en la vista de las Estaciones de Recarga, explicado en el apartado "*5.3.2.3 viewEstacionesRecarga"*, se podrá seleccionar el nodo que al usuario más le interese.

Una vez definidos todos los parámetros de reserva se procederá a Generar Reserva. Se creará una nueva ventana, ***winRealizaReserva***, dónde se mostrarán todos los datos de la reserva, figura 60.

Una vez que el usuario compruebe todos los datos de la reserva que quiere realizar se realiza la reserva, envía una petición POST al servidor SRG para finalizar la reserva.

La respuesta que envía el servidor puede ser afirmativa o negativa:

- Afirmativo: Se recibirá un JSON con el identificador de reserva y la aplicación redirecciona al usuario a la ventana de próximas reservas.
- Negativa: Aparece un mensaje de error con el motivo por el cuál no se puede realizar la reserva.

Figura 61.Ventana de resumen de la reserva

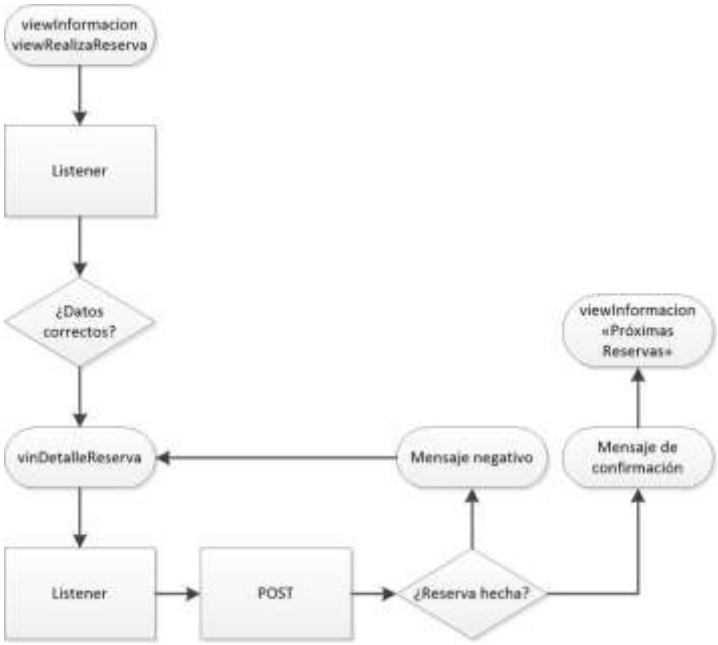


Figura 60.Diagrama de flujo de RealizaReserva

5.3.2.5 *viewPreferencias*

Esta vista muestra al usuario las opciones que puede modificar:

Matrícula del usuario: Da la opción de borrar la matrícula si el usuario decidió que se recordara al entrar en la aplicación.

- Capacidad de la batería (kW): Permite al usuario definir la capacidad de la batería. Guardando el valor en una variable global.
- Consumo de energía (kW/100km): Se ha incluido este campo para futuros cambios en la aplicación. El usuario podrá definir el consumo medio de su vehículo.
- Qué puntos de recarga mostrar: Esta opción se utiliza para definir la distancia desde la posición del usuario a los nodos en la vista de Estaciones de Recarga, muestra los n nodos en k kilómetros.

Al igual que el parámetro de Consumo de energía se pueden definir para posibles actualizaciones de la aplicación:

- Qué tipo de recarga quiere.
- Qué tipo de servicio quiere.

Estas dos opciones podrían servir para que el usuario tuviera una lista de reservas favoritas.

5.4 Dificultades en el desarrollo de la aplicación

Durante el camino del desarrollo de la aplicación se han encontrado dificultades en diversos puntos.

A continuación se exponen los puntos más problemáticos y como se han ido resolviendo.

5.4.1 Paso de parámetros

Al usar Javascript uno de los principales problemas que se encuentran es la gestión adecuada de paso de parámetros, por ejemplo, se tiene un objeto y quisiera pasarlo a una función y modificarlo, como se hace en Java, en Javascript no es posible.

En Javascript los argumentos siempre se pasan por valor y el resultado de la ***function*** se devuelve con ***return***, figura 62.

```
//Funcion para representar los valores en dos columnas
function fila_column(args){
        var row = Ti.UI.createTableViewRow();
        //Vista columna consulta
        var label1 = Ti.UI.createLabel({
                text: args.nombre,
                width: '45%',
        left: args.left1,
                color: args.color1,
        height: 'auto',
        textAlign: Ti.UI.TEXT_ALIGNMENT_LEFT,
        font: { fontSize:args.fontSize },
        });
        //Vista columna datos
        var label2 = Ti.UI.createLabel({
                text: args.valor,
                color: args.color2,
        height: 'auto',
        textAlign: Ti.UI.TEXT_ALIGNMENT_LEFT,
        font: { fontSize:args.fontSize },
                left: args.left2,
        });
        row.add(label1);
        row.add(label2);

        return row;
};
```

Figura 62.Función de Javascript

En un principio se empezó a usar ***return*** para devolver los datos tanto para la creación de las filas de las tablas de datos, como el código que se muestra en la siguiente figura, figura 63, como para cuando se realizaba una petición al servidor. El problema es que Javascript es un lenguaje interpretado en tiempo de ejecución, por ejemplo, en el caso de la conexión con el servidor, los datos eran devueltos en un intervalo de tiempo, si se usa la sentencia ***return*** devuelve la variable sin esperar la

respuesta del servidor por lo que hay que buscar otro camino.

Para solucionar el problema se buscó la forma de devolver los datos de otra forma. La solución ha sido el uso de ***callback***, es decir, una "llamada de vuelta".

Es un concepto sencillo, se llama a una función y en los argumentos se le pasa una función (un ***callback***) o más. La función que se le pasa como argumento espera a que se ejecute el ***callback*** en la función principal. Si por ejemplo el ***callback*** se encuentra en un *if…else* puede que no se ejecute la llamada a la función secundaria.

Un ***callback*** no significa que cuando termine de ejecutar la función se llama a la función que se ha pasado por parámetros. La idea es que cuando se produce un evento se llama a la función que se pasó por parámetro a la función principal.

Los ***callbacks*** no son asincrónicos, es decir, tras ejecutarse la función referenciada continúa por la función principal, es decir, la que ejecuta el ***callback***.

```
//Función que permite marcar el pago en el lugar del estacionamiento.
function funcionPago(callback){
        var rowPago = Ti.UI.createTableViewRow();
        //Vista columna consulta
        var labelTitulo = Ti.UI.createLabel({
                text: 'Pago Insitu: ',
                color: 'black',
                left: '5%',
                font: { fontSize:18 },
        });
        //Vista columna datos
      …

        //Se devuelve el valor elegido.
        basicSwitch.addEventListener('change', function(e) {
                callback(e.value);
        });

      …

      });
```

Figura 63.Función que ejecuta un callback

El ***callback*** se ha usado para la recuperación de los datos de las respuestas de servidor, debido a que estos datos dependen del tiempo de respuesta del servidor.

5.4.2 Adaptación a pantallas

La adaptación de la aplicación a diferentes tamaños de pantalla es un problema real para aplicaciones nativas o como es el caso de esta aplicación, una aplicación híbrida.

Appcelerator da como opción usar un posicionamiento relativo según un tanto por ciento, figura 64.

```
var basicSwitch = Ti.UI.createSwitch({
  titleOn:'Activado',
  titleOff:'Desactivado',
  value:false,
  left: '70%',
  width: '30%',
  height: '90%',
  borderRadius: 5,
  style: Titanium.UI.Android.SWITCH_STYLE_CHECKBOX,
});
```

Figura 64.Propiedades de estilo

En un principio se usaron otros métodos, como es el posicionamiento según los píxeles pero para el diseño de la aplicación se veía un gran inconveniente debido a las diferentes pantallas existentes en el mercado actual, por lo que se ha optado por el posicionamiento relativo según un tanto por ciento.

Al usar porcentajes para el posicionamiento se consiguió que la aplicación se pueda adaptar para varios tipos de pantalla aunque no en pantallas más pequeñas el posicionamiento de los elementos se quede con unos márgenes reducidos.

5.4.3 Sobrecarga en memoria

Uno de los problemas que se encuentran al usar aplicaciones dinámicas es que para presentar información de diferente índole se tiene que cambiar entre diferentes ventanas.

Puede repercutir en la sobrecarga del sistema. Para conseguir una menor carga se usan eventos para ejecutar el cierre de ventanas cuando pasamos de una a otra, figura 65, en cambio para las vistas se usa un método que proporciona Titanium para las ventanas, ***remove***, figura 65.

```
tableView.addEventListener('click',function(e){
        win_gestion.remove(menuView);
        win_gestion.remove(viewInformacion);
       win_gestion.remove(sombra);

       …

win_gestion.open();
//Se cierrra la ventana de entrada
Titanium.App.fireEvent("cerrarWinEntry");
}
```

Figura 65.Código para cerrar una ventana

6 CONCLUSIONES Y LÍNEAS FUTURAS

6.1 Conclusiones

En la actualidad la conectividad se ha hecho imprescindible, y ha crecido la demanda del desarrollo de aplicaciones para dispositivos móviles.

Con la aplicación para gestionar las reservas en FerroSmartGird se ha intentado demostrar que no es necesario el aprendizaje de los lenguajes nativos para crear aplicaciones multiplataforma, dando a conocer en primera instancia las opciones actuales que se pueden encontrar y en una segunda instancia mostrar una aplicación desarrollada en una de las opciones disponibles.

Las ventajas y alternativas que ofrece Appcelerator a la hora de abordar el diseño y programación de aplicaciones multiplataforma son muy amplias. La comunidad que apoya el proyecto es extensa y facilita la resolución de dudas y la participación activa de los desarrolladores.

No obstante en Appcelerator sigue existiendo una problemática presente en Android desde hace años. La gran variedad de dispositivos y los diferentes tamaños de pantallas hacen que la tarea de adaptación sea laboriosa para el desarrollador. Al comenzar el desarrollo de la aplicación se pensó que este problema se solventaría con la posición relativa pero al probar la aplicación en dispositivos con diferentes pantallas se ha visto que el problema persiste.

Aun así Appcelerator sigue siendo una opción a tener en cuenta para el desarrollo de aplicaciones multiplataforma por la independencia que da respecto a los lenguajes de programación nativos.

Appcelererator ofrece acceso a la mayoría de las APIs de las diferentes plataformas, consiguiendo acercarse a las aplicaciones nativas.

Tras hacer un uso intenso de la aplicación se ha comprobado que para un usuario el resultado no difiere mucho de una aplicación nativa, consiguiendo uno de los objetivos primordiales en el desarrollo de aplicaciones basado en entornos multiplataforma.

En cualquier caso la finalización de este proyecto, más que representar un documento cerrado, abre nuevas puertas a la investigación de otras alternativas para el desarrollo multiplataforma, pudiendo ser útil a un desarrollador que desee dar una solución única al problema de la variedad de dispositivos.

6.2 Líneas futuras

La aplicación que se ha desarrollado en este proyecto ha servido como IHM –Interfaz Hombre Máquina- para el servidor de gestión reservas del proyecto FerroSmartGrid.

No obstante la aplicación desarrollada no se tiene que limitar a este proyecto podría servir para proyectos del mismo tipo o proyectos en dónde se quisiera realizar una aplicación para reserva de aparcamiento sin necesidad de ser de vehículos eléctricos, se podría reutilizar parte del código.

Esta aplicación podría extenderse con las siguientes funcionalidades:

- NFC: Una idea que se pensó al comenzar el proyecto era usar la tecnología NFC –*Near Field Communication*- para que cuando el usuario llegase al aparcamiento pudiera utilizar su reserva sin necesidad de introducir ningún código.
- Se podría añadir la posibilidad de crear reservas múltiples, es decir, crear un plan semanal, quincenal, mensual, anual… de reservas con la elección de días y horas necesarias , por ejemplo, un usuario que vive en un pueblo cercano a Sevilla y trabaja en Córdoba y quisiera reservar toda la semana una plaza en el nodo.
- También cabe la posibilidad de añadir reservas rápidas definiendo los parámetros principales en la vista de preferencias.
- Tener reservas favoritas, por ejemplo, si un usuario toma un tren pero no es de forma regular, puede almacenar ese nodo y el tipo de recarga que normalmente realiza.
- Un ítem del menú también podría ser Promociones u Ofertas, por ejemplo, el usuario al estar abonado al servicio podría recibir bonos de descuento u ofertas en un determinado nodo.

Esta aplicación puede servir de ejemplo para desarrollar aplicaciones que hagan conexiones a un servidor a través de REST y con datos JSON.

7 BIBLIOGRAFÍA

1. Libros Web AJAX. [En línea] http://librosweb.es/ajax/.

2. **Leonard Richardson, Sam Ruby.** *RESTful Web Services.* s.l. : O'REILLY, 2007.

3. **Anderson, John.** *Appcelerator Titanium: Up and Running.* s.l. : O'REILLY, 2013.

4. **Cope, Darren.** *Appcelerator Titanium Application Development by Example Beginner's Guide.* s.l. : PACKT Publishing, 2013.

5. **Inc., Adobe Systems.** PhoneGap. [En línea] phonegap.com.

6. **Spain, PhoneGap.** PhoneGap Spain. [En línea] www.phonegapspain.com.

7. **Foundation, The Apacahe Software.** Apache Cordova. [En línea] cordova.apache.org.

8. **Founation, The Apache Cordova.** Documentación Apache Cordoba. [En línea] cordova.apache.org/docs/es/3.1.0/index.html.

9. **Joynet, Inc.** Node js. [En línea] noejs.org.

10. **Inc, Adobe Systems.** Build in Phonegap. [En línea] build.phonegap.com.

11. **Manuel Kiessling, Herman A. Junge.** *El libro para Principiantes en Node.js.*

12. **Zárate, Israel Ortñiz de.** Una docena de... [En línea] 31 de Enero de 2013. http://unadocenade.com/una-docena-de-razones-para-utilizar-phonegap-en-el-desarrollo-de-aplicaciones-par-dispositivos-moviles/.

13. **Solis, Carlos.** Revolucion Mobi. [En línea] 5 de Febrero de 2014. revolucion.mobi/category/phonegap/.

14. **Inc., Xamarin.** Xamarin. [En línea] 2014. xamarin.com.

15. **Inc., Adobe.** Creación de aplicaciones de Adobe Air. [En línea] 2013. http://help.adobe.com/es_ES/air/build/air_buildingapps.pdf.

16. **Roberto Lerusalumschy, Waldemar Celes, Luiz Henrique de Figueiredo.** Lua language. [En línea] www.lua.org.

17. **eMobc.** eMobc. [En línea] www.emobc.com.

18. **Yglesias, Pablo.** Entrevista a Alejandro Sánchez Acosta, CEO de eMobc. [En línea] 10 de Diciembre de 2012. http://www.pabloyglesias.com/entrevista-a-alejandro-sanchez-acosta-ceo-de-emobc/.

19. **libre, El andoide.** Xamarin, la API para crear aplicaciones multiplataforma en C#/.NET. [En línea] http://www.elandroidelibre.com/2014/05/xamarin-la-api-para-crear-aplicaciones-multiplataforma-en-c-net.html?utm_content=buffer0c038&utm_medium=social&utm_source=twitter.com&utm_campaign=buffer.

20. **Whinnery, Kevin.** Comparing Titanium and Phonegap. [En línea] 2012. http://kevinwhinnery.com/post/22764624253/comparing-titanium-and-phonegap.

21. **Libre, Guadalux Fundación Software.** Titanium Appcelerator Vs. PhoneGap. [En línea] http://guadalux.org/appcelerator-vs-phonegap/.

22. **Medida, Alvantia Soluciones a.** PhoneGap y Titanium, alternativas para el desarrollo móvil multiplataforma. [En línea] 31 de Enero de 2013. http://www.alvantia.es/blog/phonegap-y-titanium-alternativas-para-el-desarrollo-movil-multiplataforma/.

23. **Vilches, Alberto.** Desarrollo de aplicaciones móviles multiplataforma: web vs nativas vs multiplataforma. [En línea] http://albertovilches.com/aplicaciones-moviles-web-vs-nativas-vs-multiplataforma.

24. **Doncel, Marco.** Desarrollo de aplicaciones multiplataforma. [En línea] http://www.startcapps.com/blog/desarrollo-de-aplicaciones-multiplataforma/.

25. **Velasquez, Frisley.** Desarrollando Aplicaciones Móviles (Nativas VS Phonegap VS TItanium). [En línea] http://frisley.com/?p=758.

26. **Appcelerator.** Appcelerator. [En línea] http://www.appcelerator.com/platform/appcelerator-platform/.

27. —. Documentación de Titanium. [En línea] http://docs.appcelerator.com/titanium/latest/#!/guide/Quick_Start.

28. —. Wiki Appcelerator. [En línea] https://wiki.appcelerator.org.

29. **Tamas, Dan.** Siete Días con Titanium. [En línea] http://nosoloweb.es/siete-dias-con-titanium-dia-0-introduccion/.

30. **Geek, Ti SL Mundo.** Web Services JSON en Titanium Appcelerator. [En línea] http://ti-sl.blogspot.com.es/2013/07/10-web-services-json-en-titanium.html.

31. **Canarias, Appcelerator.** Appcelerator Canarias. [En línea] http://appceleratorcanarias.wordpress.com/.

32. **Appcelerator.** Appcelerator Blog. [En línea] http://www.appcelerator.com/blog/.

33. —. Appcelerator Developer. [En línea] https://developer.appcelerator.com.

34. **w3shools.** JavaScript Tutorial. [En línea] http://www.w3schools.com/js/default.asp.

35. **JavaScript, Manual de.** Manual de JavaScript. [En línea] http://www.manualdejavascript.com/section/manualdejavascript/.

36. **Crockford, Douglas.** JavaScript. [En línea] http://javascript.crockford.com/.

37. **Eguiluz, Javier.** Introducción a JavaScript. [En línea] http://librosweb.es/javascript/.

38. **JSON.** Introdcucción a JSON. [En línea] http://json.org/json-es.html.

39. **w3schools.** JSON Syntax. [En línea] http://www.w3schools.com/json/json_syntax.asp.

40. **Inc., Google.** Android Developer. [En línea] http://developer.android.com/intl/es/index.html.

ANEXO I: MANUAL DEL USUARIO

1. Introducción

Este anexo es un manual que muestra todas las características que tiene la aplicación al usuario desde el punto de vista de su uso.

No se muestra la parte de instalación, una vez en explotación se realizaría del modo habitual, a través de las tiendas de aplicaciones de los diferentes SO móviles.

2. Estructura

2.1. Interfaz de autenticación

Al iniciar la aplicación aparece la pantalla donde debe registrarse el usuario. Para ello primero deberá elegir la forma de autenticación:

- Id. Usuario: Es la identificación de usuario que el cliente tiene.
- Matrícula: Matrícula del vehículo que se tiene registrado en el sistema.

También el usuario tiene la posibilidad de elegir recordar una matrícula o un usuario para que esta pantalla no aparezca la próxima vez que entre, figura 66, si desea el usuario en un futuro dejar de recordar la matrícula se podrá hacer desde el panel de preferencias, que se explica en el apartado *"8.2.4.Interfaz de Preferencias"*.

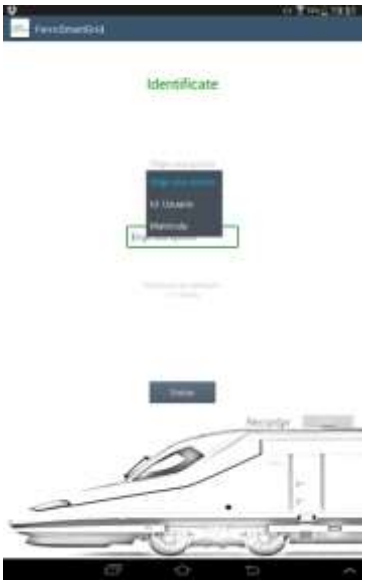


Figura 66.Intergaz de autenticación

Figura 67.Capacidad de la batería

Una vez rellenado el campo de registro el usuario pulsará el botón entrar.

Si es la primera vez que entra en la aplicación le pedirá la capacidad de la batería del vehículo, que podrá modificar en un futuro desde el panel de preferencias, que se explica en el apartado *"8.2.4.Interfaz de Preferencias"*. La carga se debe indicar en kWh.

2.2. Interfaz Principal

Una vez autenticado el usuario, se mostrará una interfaz donde se presentarán los datos de la siguiente reserva que tenga el usuario o un mensaje de ausencia de reservas en caso de que no existan próximas reservas.

En la figura se ven los siguientes campos de información sobre la reserva:

- Matrícula: La matrícula del vehículo.
- Id. Reserva: Identificación de la Reserva
- Entrada: Fecha y hora en la que se espera que llegue el usuario.
- Salida: Fecha y hora que el usuario retire el vehículo.
- Carga Inicial: Carga con la se espera que el vehículo llegue al nodo.
- Carga final: Carga tras la estancia en el nodo.
- Mapa: Localización del nodo en un mapa.

Figura 68.Ventana de bienvenida

2.3. Menú

Para acceder al menú se deberá pulsar en la esquina superior izquierda que hará que se abra el menú. Para cerrarlo se debe pulsar también en la esquina superior izquierda. . El menú se puede ver la matrícula del vehículo y las opciones disponibles:

- Estaciones de Recarga: Nodos que se encuentran a una distancia X, esta distancia se definirá en el

panel de preferencias, apartado *"8.2.4.Interfaz de Preferencias".*

- Realizar Reservas: Interfaz que permite definir los campos para crear una nueva reserva.
- Próximas Reservas: Las reservas que existen para la matrícula indicada y que aún no están usadas.
- Reservas Usadas: Las reservas que existen para la matrícula indica que han sido utilizadas.
- Reservas Anuladas: Las reservas para la matrícula indicada que se han anulado antes de ser usadas.
- Preferencias: Interfaz de configuración.

2.4. Interfaz de Preferencias

Desde la interfaz de usuario se definen parámetros del vehículo y parámetros usados para realizar reservas:

- Matrícula del usuario: Olvida la matrícula que la aplicación tiene almacenada. Si no se ha recordado ninguna aparece el mensaje: "El usuario no ha guardado ninguna matrícula".
- Capacidad de la batería: Se define la capacidad de la batería en kWh. El usuario podrá cambiarlo siempre que desee desde esta opción.
- Consumo de energía: El usuario podrá definir el consumo que posee su vehículo. Esta medida se indica en KWh, Kilowatios hora.
- Qué puntos de recarga mostrar: Define la distancia desde la posición del usuario hasta el nodo más lejano, en kilómetros, que se mostrará en la interfaz Estaciones de Recarga, se explica en el apartado *"8.2.5.Estaciones de Recarga".*

Figura 69.Interfaz de Preferencias

2.5. Interfaz Estaciones de Recarga

Esta interfaz presenta un mapa con los nodos que se encuentran a X kilómetros, definida en la interfaz de preferencias, vista en el apartado *"8.2.4.Interfaz de Preferencias",* respecto de la posición GPS del usuario.

Desde esta interfaz se puede seleccionar el nodo deseado, aparecerá un cuadro emergente que indica los servicios que se ofrecen en el nodo seleccionado, figura.

Figura 70.Interfaz de Estaciones de Recarga

Figura 71.Servicios del nodo

Desde esta interfaz también se podrán seleccionar los servicios que desee el usuario y pulsar el botón de realizar reserva, lo que llevará al usuario a la interfaz de Realiza Reserva, que se explica en el apartado *"8.2.6.Interfaz Realiza Reservas"*.

2.6. Interfaz Realiza Reservas

- Carga al llegar: Es un parámetro opcional por si el usuario desea indicar la carga con la que.se estima de que llegue el vehículo.
- Carga a recargar: Es la carga, en tanto por ciento, que el usuario desee recargar, esta carga es neta, es decir, la carga final será la carga al llegar más la carga a recargar. Este parámetro es obligatorio si no se define el precio.
- Precio: Es el gasto que desea realizar el usuario, es necesario sino se define una carga a recargar. Actualmente solo permite euros.
- Fecha de llegada: La fecha estimada de entrada en el nodo.
- Fecha de salida: La fecha estimada en la que el usuario pretende abandonar el nodo.

- Tipo de recarga: Es una lista de los tipos de carga disponibles que existen.
    - Carga Inductiva.
    - Carga Capacitiva.
    - Recarga Rápida.
- Pago Insitu: Si se selecciona el usuario deberá pagar al retirar el vehículo y no se cargará en su cuenta de abonado.

Figura 72.Interfaz realiza reserva en el nodo seleccionado

Figura 73.Interfaz realiza reserva, sin nodo seleccionado

Si el usuario accede a esta interfaz desde el menú tendrá una opción más, figura 73:

- Buscar punto de recarga: Este campo presenta dos opciones:
    - Por ciudad: El usuario escribe una ciudad y muestra los puntos de recarga que existen en esa ciudad.
    - Por cercanía: El usuario indica que desea ver todos los nodos que existen desde su posición hasta X kilómetros, indicando el número de kilómetros en un cuadro emergente.

Al igual que en la Interfaz de Estaciones de Recarga se podrá seleccionar un nodo, ver los servicios y elegir los que el usuario necesite. Una vez que se acabe se vuelve a la interfaz de Realiza Reserva.

Una vez elegido el nodo el usuario podrá verificar su reserva antes de realizarla, figura 74.

Figura 74.Interfaz verificación de reserva

Figura 75.Mensaje de reserva realizada correctamente

Para finalizar la reserva se pulsará el botón “Reservar”. Y el usuario recibirá:

- Id de Reserva, como se ve en la figura 75.
- Respuesta negativa, si la reserva no ha podido llevarse a cabo.

2.7. Interfaz de Reservas

El usuario puede ver las reservas que ya ha realizado. En el menú se pueden encontrar tres opciones, mencionadas anteriormente.

- Próximas Reservas.
- Reservas Usadas.
- Reservas Anuladas.

Figura 76.Lista Próximas Reservas

Figura 77.Lista Reservas Usadas

Figura 78.Lista Reservas Anuladas

En la interfaz de reservas se muestran las reservas en una lista, figura 76,77 y 78 con los siguientes datos:

- Id Reserva: El identificador de reserva.
- Fecha de entrada.
- Fecha de salida.

El usuario puede obtener más información sobre ellas pulsando en cada una de ellas.

Cuando el usuario pulsa sobre las reservas se le muestra una interfaz mostrando los datos de la reserva seleccionada de una manera más detallada.

- Si se pulsa en una reserva de Próximas Reservas, mostrará los detalles de la reserva y la posibilidad de anularla, figura 80.
- Si se pulsa en una reserva de Reservas Usadas, figura 81, se muestran los detalles de la reserva y la posibilidad de ver el recibo, figura 82.
- Si se pulsa en una reserva anulada, figura 79, se muestran los detalles de la reserva anulada.

Figura 79.Interfaz Detalle Reservas Anuladas

Figura 80.Interfaz Detalle Próxima Reserva

Figura 81.Interfaz Detalle Reserva Usada

Figura 82.Interfaz Reserva Usada Recibo

ANEXO II: MANUAL DEL PROGRAMADOR

Con este anexo se quiere dar a conocer como está organizado el código, para permitir a otros desarrolladores que en un futuro puedan realizar modificaciones y extensiones.

Todo el código se adjunta en un CD.

1. Descarga e instalación de Titanium

Para poder ejecutar Titanium Studio, el sistema debe tener:

- Sistema Operativo: Una versión reciente de Windows, OS X o Ubuntu.
- Memoria: 2 Gb de RAM (Memoria disponible, no memoria total)
- Java Runtime: Oracle JDK, no es posible sustituirlo por otra marca.

Para Windows, la versión de 32 bit de Java JDK sirve para que funcione Titanium de 32 bit y de 64bit.

Para descargarse Titanium Studio antes se tiene que crear una cuenta de Appcelerator, esta cuenta es gratuita y nos dará acceso a una tienda de paquetes, gratuitos y de pago, con diferentes funciones ya desarrolladas.

Una vez registrado simplemente habrá que descargarse la versión de Titanium Studio apropiada para el sistema operativo que se vaya a utilizar, figura 83.

Figura 83.Elección de SO para descargar Titanium Studio

En el caso de este proyecto se ha descargado la versión de Windows, una vez finalizada la descarga simplemente se deberá ejecutar el instalador y seguir las instrucciones. El instalador de Windows además descarga e instala Java runtime, figura 84.

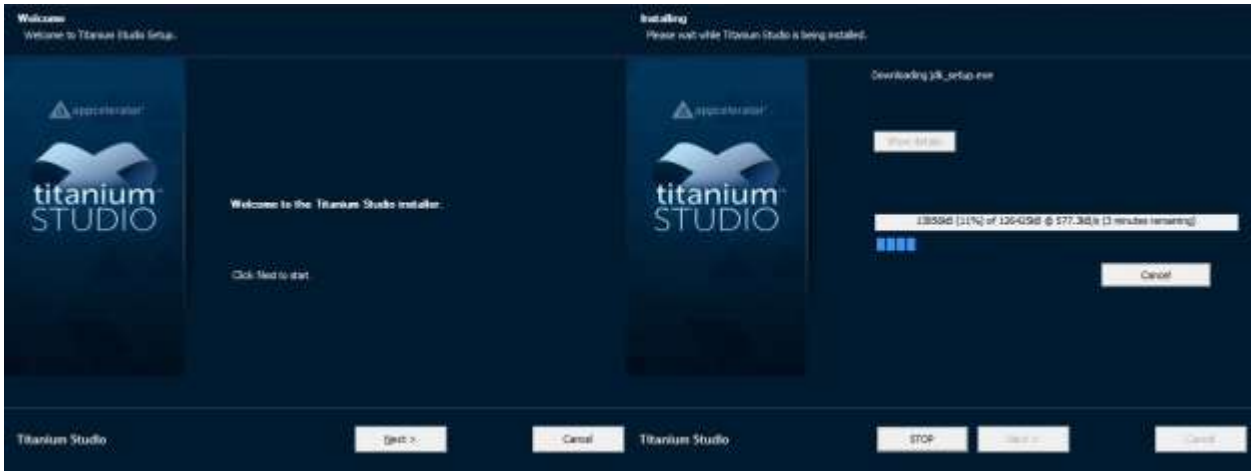


Figura 84.Instalación de Titanium Studio

2. Primera ejecución de Titanium Studio

Una vez terminada la instalación se entra en Titanium Studio, la primera vez que se inicia Titanium pedirá que se defina un espacio de trabajo, dando la opción de dejarlo como predeterminado o preguntar cada vez que se vuelva a iniciar el espacio de trabajo.

Una vez que se elige el espacio de trabajo se solicitan el usuario y la contraseña con la que anteriormente se hizo el registro en Appcelerator.

Figura 85.Registro de usuario en Titanium Studio

3. Instalación de SDKs desde Titanium Studio

Para instalarlo directamente desde Titanium, la primera vez que se inicia se comprueba las plataformas que permite el propio sistema donde se está trabajando y aparece un diálogo de configuración.

El diálogo que aparece indica qué plataformas están instaladas y cuáles no. Aunque en un primer momento no se desee configurar siempre al iniciar Titanium aparecerá una pantalla principal dónde se podrán configurar nuevos SDKs y comprobar los ya instalados, figura 25.

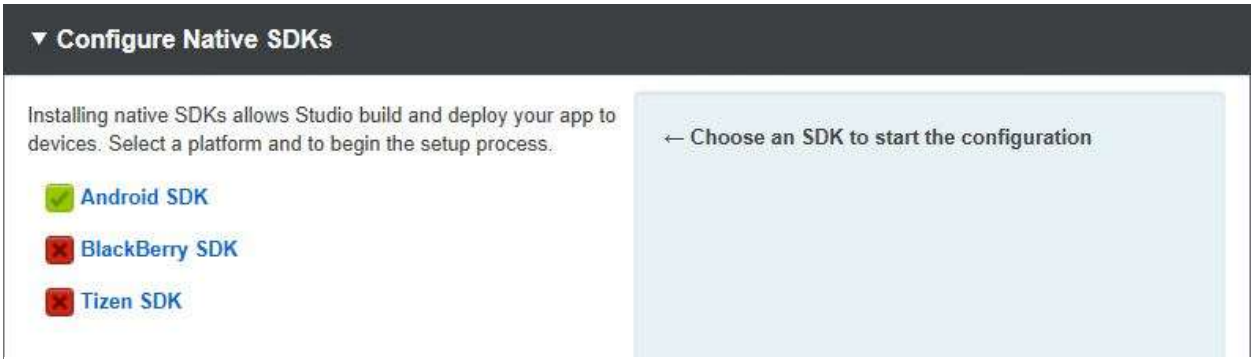


Figura 86.Configuración de SDKs Nativos en Titanium Studio

Las configuraciones de cada plataforma son transparentes, Titanium Studio descarga el software y automáticamente lo instala y lo configura.

4. Importar un proyecto

Para importar un proyecto hay que ir a ***File/Import…***

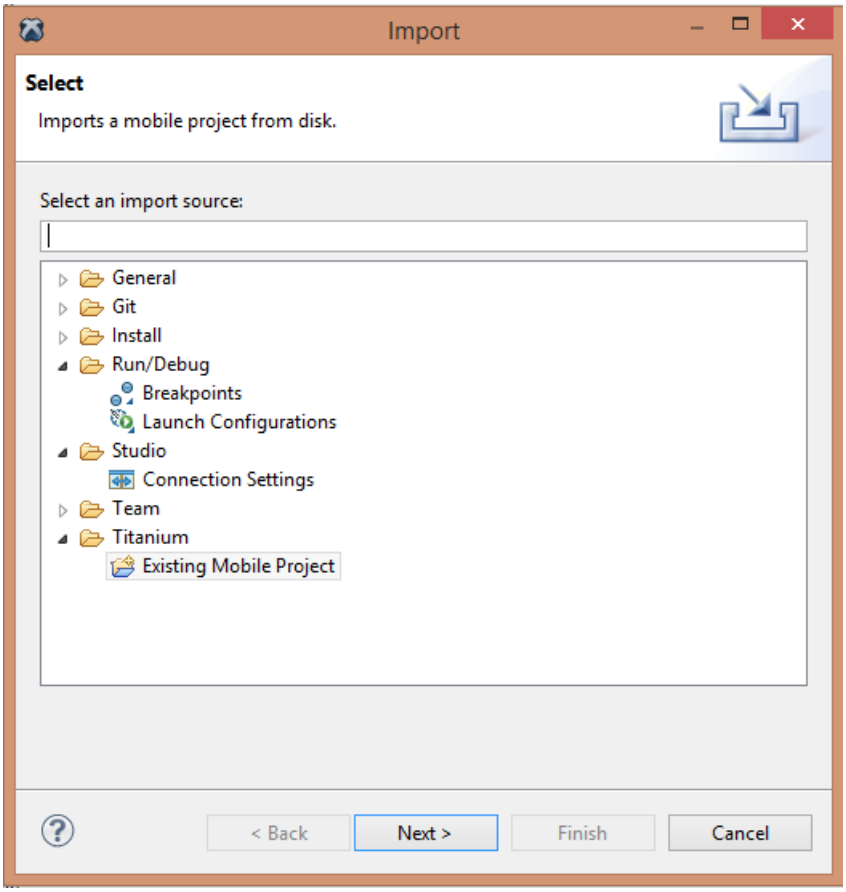


Figura 87.Importar el proyecto

- Se elige ***Titanium/Existing Mobile Project*** si se quiere exportar la carpeta directamente
- O ***General/Archive File*** si se ha exportado el proyecto a un archivo previamente.
- A continuación se buscará la ruta y se selecciona el archivo.

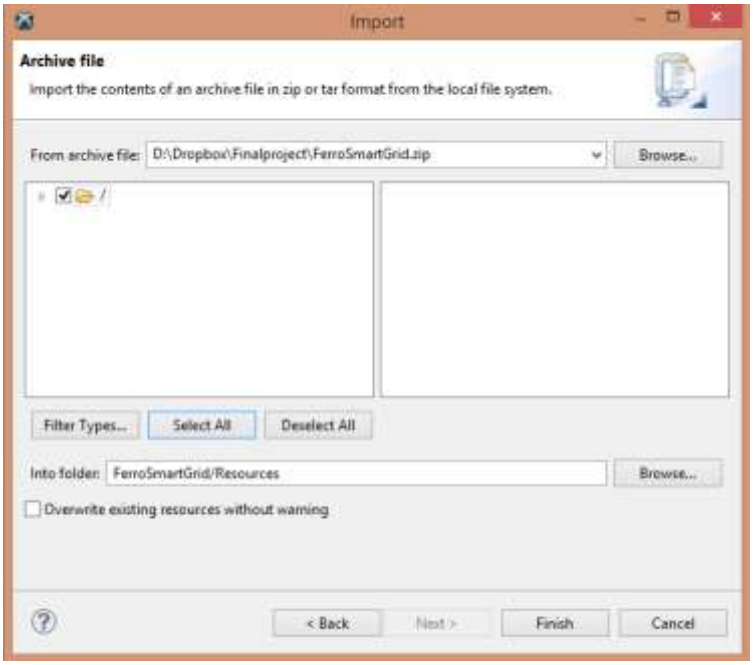


Figura 88.Importar proyecto guardado en un archivo

A continuación se pulsará finalizar y aparecerá la carpeta del proyecto, figura 89.

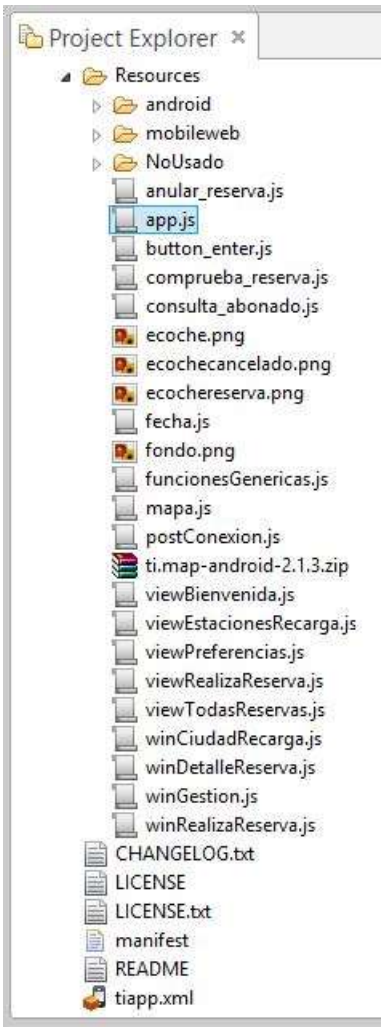


Figura 89.Directorio del proyecto

En el directorio se encuentra el archivo de configuración, llamado ***tiapp.xml***, que podrá visualizarse de dos formas, figura 90:

- En formato xml
- En formato gráfico

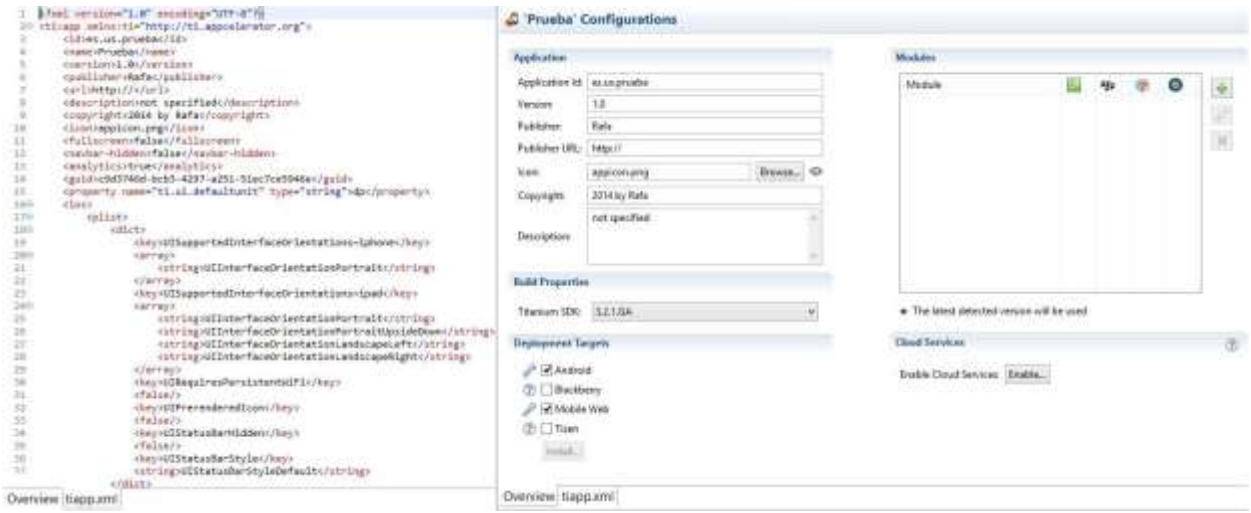


Figura 90.Archivo de configuración: tiapp.xml

Este archivo de configuración permite realizar cambios en las plataformas de desarrollo, introducir módulos para usar APIs del sistema nativo, o configurar temas visuales. También permite cambiar parámetros para la publicación de la aplicación final.

Además del fichero de configuración se crean un archivo para redactar la licencia de uso de la aplicación y el archivo principal de la aplicación llamado ***app.js***.

En el directorio del proyecto el desarrollador se encuentra:

- ***LICENSE*** y ***LICENSE.txt***.
- El archivo ***manifest*** donde se encontrarán todos los datos del desarrollador de la aplicación.
- Se encuentran todos los archivos de código.
- Una carpeta por cada plataforma para la que se desarrolla la aplicación.
- Se incluye el módulo ***ti.map-andrid-2.1.3.zip*** que incluye las librerías necesarias para añadir mapas de Google Maps a la aplicación.
- Las imágenes qué se pueden usadas en la aplicación se encuentra en el directorio ***Resources***.

5. Estructura de la aplicación

La aplicación se divide en archivos para dividir el problema y que sea más fácil su comprensión, a continuación se presenta un esquema de cómo se organizan estos archivos.

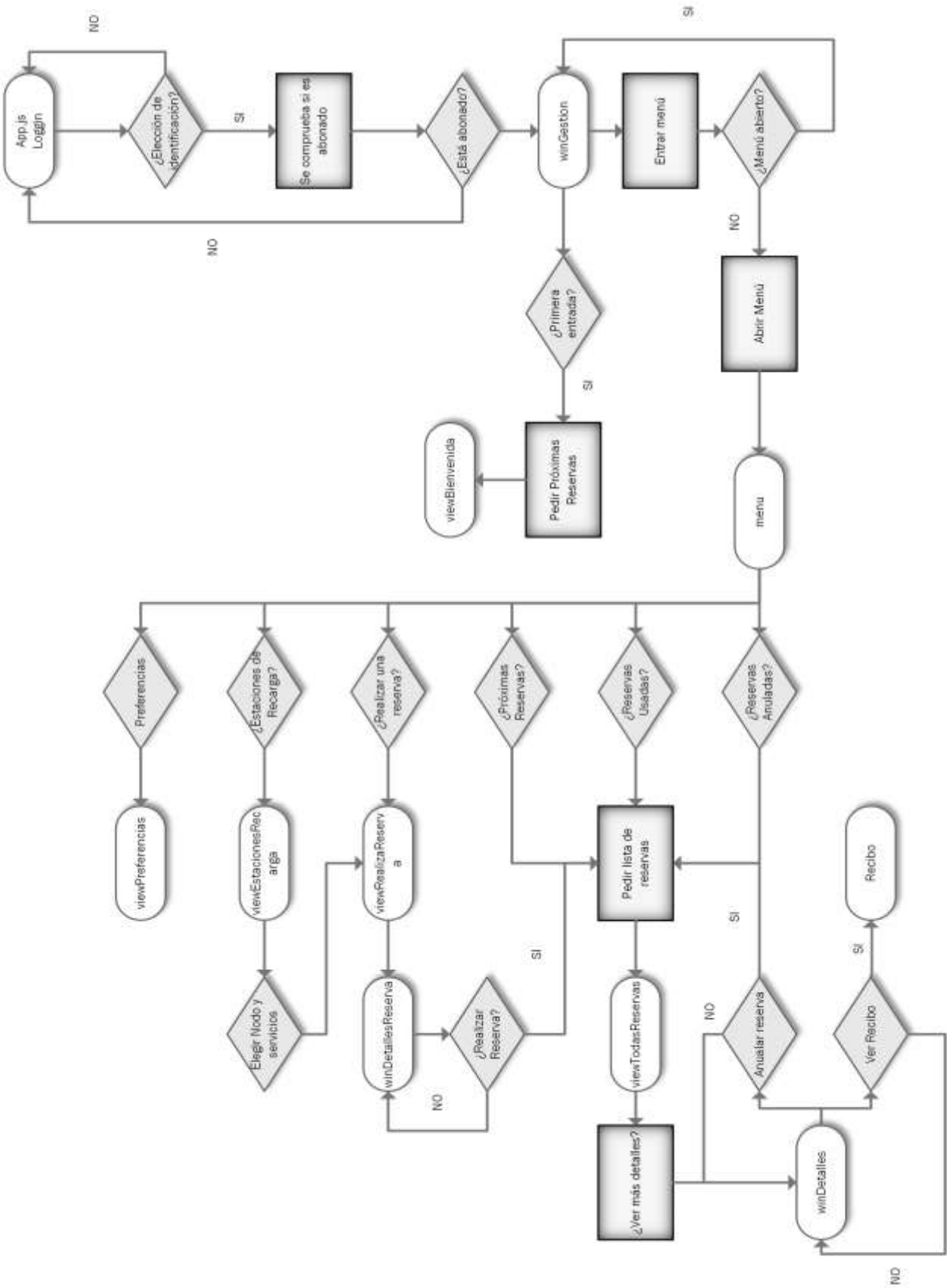


Figura 91.Diagrama de flujo de la aplicación

En la figura 91 se ve como el archivo principal es app.js, este archivo tiene que estar presente siempre que se cree una aplicación en Titanium, es el archivo que comienza a leer la máquina de Javascript cuando se ejecuta la aplicación.

6. APIs de Titanium.

6.1. Elementos básicos

El diseño de la interfaz de utilizan los siguientes elementos:

- Ventana: Es la interfaz a la que se le añaden todos los elementos.

```
//Ventana de bienvenida donde el usuario se registra
var win_entry = Titanium.UI.createWindow({
        tittle:'Win_entry',
        backgroundImage: '/fondo.png'
});
```

Figura 92.Código para crear una ventana

- Vistas: Es un espacio vacío para contener los elementos básicos que se han visto anteriormente en el apartado "*5.3.Diseño de la Aplicación*".

  Por ejemplo, cuando se llama al fichero winGestion.js se crea la ventana principal de la aplicación, en esta ventana se cargarán diferentes vistas según la opción que el usuario desee visualizar, para definir una vista se tiene que usar el objeto de la librería ***Titanium Ti.UI.View***.

```
// --- MENU ---
//Vista del menú en el lado izquierdo
var menuView = Ti.UI.createView({
        top:0,
        left:'-60%',
        width:'60%',
        backgroundColor: 'lightgray',
        abierto: false
});
```

Figura 93.Código para crear una vista

- Cuadro de texto: En este cuadro de texto se podrá incluir el texto que el desarrollador desee. Para definirlo se usa la biblioteca ***Titanium.UI.createLabel***, como se ve en la figura 94.

```
var label_id = Titanium.UI.createLabel({
        color: '#088A08',
        font: { fontSize:32 },
        text: 'Identifícate',
        textAlign: Ti.UI.TEXT_ALIGNMENT_CENTER,
        bottom: '90%',
        width: Ti.UI.SIZE, height: Ti.UI.SIZE
});
```

Figura 94.Código para crear un cuadro de texto

- Campo de texto: El campo de texto permite al usuario escribir en ese cuadro un texto. Además de las propiedades que se han mencionado para el cuadro de texto. Se puede almacenar en el campo ***value*** el valor que el usuario le da. Para crear este cuadro se usa el código de la figura 95.

```
//Cuadro de texto donde se introduce la identificación
var box_user = Titanium.UI.createTextField({
        value: "",
        borderStyle: Ti.UI.INPUT_BORDERSTYLE_ROUNDED,
        color: '#088A08',
        backgroundColor: 'white',
        borderColor:'#088A08',
        borderRadius: 5,
        borderWidth: 3,
        bottom: '60%',
        width: 250,
        height: 50
});
```

Figura 95.Código para crear un campo de texto

Estos elementos básicos están presentes a lo largo del código.

- Al finalizar de realizar el diseño hay que añadir todo a la interfaz, la ventana, para que aparezca en la interfaz. Para añadir los elementos a la ventana se usa el método ***add.***

```
//Incluimos los objetos en la ventana de entrada
win_entry.add(label_id);
win_entry.add(pickerIdentificacion);
win_entry.add(box_user);
win_entry.add(ejemploId);
win_entry.add(labelRecordar);
win_entry.add(basicSwitch);
win_entry.add(button_enter);
```

Figura 96.Método para añadir elementos a una vista o ventana

- Tabla: Se usa para crear una lista de elementos. Un ejemplo de tabla se puede ver en el fichero de ***RealizaReserva.js*** dónde para mostrar los campos que se tienen que rellenar para realizar la reserva se usa tabla, figura 97.

```
var tablaReserva = Ti.UI.createTableView({
        minRowHeight: 60,
        separatorColor: 'gray',
});
```

Figura 97.Código para crear una tabla

- Para añadir elementos se usa el evento ***add*** explicado en el punto anterior. Las filas se crean con el objeto ***TableViewRow***, como se ve en la figura 98.

```
var rowCarga = Ti.UI.createTableViewRow();
```

Figura 98.Código para crear una fila

- Titanium permite la creación de objetos que se encuentran en otros ficheros, por ejemplo una vista que está compuesta de elementos básicos y que se quiere añadir a una ventana. Para poder crear estos objetos será necesario definir un objeto y asociarle el fichero que necesita para crear dicho objeto.

```
// --- FICHEROS REQUERIDOS ---
viewBienvenida = require('./viewBienvenida');
...
//Creación de una vista se inicia al entrar la primera vez
            viewInformacion = new
viewBienvenida(args.matricula,args.id_usuario,args.uri,fechaHoy,"311220202359",false,true,function(mapa){
...
```

Figura 99.Creación de un objeto

6.2. Eventos

Además de los elementos básicos existen elementos con los que se puede interactuar en la aplicación.

- ***Switch***: Es un interruptor que permite activar o desactivar una casilla, además del ON/OFF de la interfaz de autenticación, un switch puede ser una casilla de verificación. Para crearlo se ha usado de nuevo la biblioteca ***Titanium.UI***, y en este caso se le ha añadido ***createSwitch*** como se ve en la figura.

```
//Switch recordar, existe la opción de recordar la matrícula
introducida
var basicSwitch = Ti.UI.createSwitch({
        titleOn:'ON',
        titleOff:'OFF',
        value:false,
        right: '2%',
        width: '15%',
        height: 35,
        bottom: '25%',
        borderRadius: 5,
        style: Titanium.UI.Android.SWITCH_STYLE_TOGGLEBUTTON,
});
```

Figura 100.Definición de Switch en Titanium

- ***Botón***: La biblioteca ***Titanium.UI***, permite crear botones, ***createButton***.

```
//Acciones al pulsar el botón de entrada
button_enter.addEventListener('click',function(){
        var input_number = Ti.UI.createTextField();
        var dialogCarga = Ti.UI.createOptionDialog();
...
```

Figura 101.Evento al hacer click en un botón

Ambos elementos van asociados a un evento que es lanzado cuando existe una interacción con el elemento. El lanzador del evento se añade como se ve en la figura 101.

En este caso se le añade un ***listener*** al botón de entrada cuando se pulse sobre él. En el caso del switch el evento será ***change*** cuando se conmute de ON a OFF. Cuando este evento ocurre se ejecuta la función que lleva asociada.

- ***Slider***: Es una barra deslizante dónde se puede elegir un valor. Este valor se almacena en una propiedad de objeto slider, ***value***.

```
var slider = Titanium.UI.createSlider({
        min: 0,
        max: 100,
        left: '50%',
        width: '45%',
        value: 0
});
```

Figura 102.Definición de un slider

Además de los eventos asociados a un elemento existen otros eventos que podrán ser llamados durante la ejecución de la aplicación a través de su nombre. Para crear un evento de este tipo se usa la biblioteca ***Titanium.App.addEventListener*** como se ve en la figura 103.

```
//Evento para cerrar la ventana de registro.
Ti.App.addEventListener('cerrarWinEntry',function(){
        win_entry.close();
});
```

Figura 103.Definición de un evento nuevo

Esta función se ejecutará cuando se llame al evento ***cerrarWinEntry***, usando el siguiente código.

```
//Se cierrra la ventana de entrada
Titanium.App.fireEvent("cerrarWinEntry");
```

Figura 104.Llamada a un evento

6.3. Paso de parámetros

Cuando se programa en Javascript suele ser confuso, o en algunos casos poco natural, la manera en que maneja el paso de parámetros a las funciones. Hay que tener en cuenta algunos detalles:

Veamos un par de detalles:

- Sólo existen funciones, ***function***
- La lista de parámetros es solo una lista de nombres de variables separadas por comas, no es necesario indicar el tipo.
- No existe paso de parámetros por referencia, solo por valor, así que no se pueden hacer procedimientos que modifiquen varias variables.
- La función puede devolver un único valor, usando la sentencia ***return***.

En Javascript se pueden utilizar ***callback*** o traducido una "devolución de llamada". Es simple, al invocar una función se envía como parámetro la referencia a una función (un ***callback***) esperando que la función invocada se encargue de ejecutar la función ***callback*** cuando lo considere necesario.

Se podrían anidar funciones que se llamasen a través de ***callback***, pero un gran número de funciones anidadas aumentaría la dificultad.

Por último y no menos importante, los callbacks no son asincrónicos, es decir, cuando se ejecuta un ***callback*** bloquea la función referenciada. Cuando la función referenciada acaba se continua con la ejecución de la función principal.

7. APIs de la aplicación

7.1. win_Entry

La interfaz de autenticación se compone de una ventana, ***win_entry***, que tiene como propiedad un fondo que en este caso es una imagen.

Además tiene los siguientes elementos:

- ***Label_id*** y ***ejemploId***: Son cuadros de texto que muestran un mensaje en la ventana de autenticación.
- ***Box_user***: Es un campo de texto donde se introduce la autenticación:
  - ***Matrícula.***
  - ***Identificador de usuario.***
- El usuario tiene la posibilidad de decir a la aplicación que recuerde la matrícula para ello se ha creado en la ventana de autenticación:
  - ***labelRecordar***: Es una etiqueta que indica al usuario que el switch asociado sirve para recordar la matrícula.
  - ***basicSwitch***: Es un switch ON/OFF, que lanza un evento para almacenar la matrícula en una variable global de la aplicación. Si el usuario desea que la aplicación deje de recordar la identificación podrá cambiarlo en el panel de Preferencias.

Cuando se lanza el evento del botón de entrada se llama al fichero ***button_enter.js***, en este fichero se comprueba si el campo de autentificación es una matrícula o es un usuario usando una sentencia *if…else*, en el caso de que sea una matrícula se realiza una validación para ver si el usuario la ha escrito correctamente. A continuación se hace una consulta al servidor llamando al fichero ***consulta_abonado.js***, este fichero contiene una consulta GET, figura 105.

```
function consulta_abonado(matricula,uri, callback){

        //Definición de una conexión para peticiones http
        var connectConsulta =  Titanium.Network.createHTTPClient({
        // Llamamos a la funcion cuando la respuesta se ha producido
                onload : function(e) {
                //Llamo al evento para que analice los datos recibidos
                Titanium.API.info(this.responseText);
                var idAbonado = JSON.parse(this.responseText);
                //Devuelve el valor para comprobar si se ha anulado correctamente
                callback(idAbonado);
        },
...
```

Figura 105.Función consulta_abonado

A la función consulta_abonado.js se le pasa una función como parámetro, un ***callback***, cuando en ***consulta_abonado*** se ejecute la sentencia ***callback*** se llama a la función secundaría que está como parámetro en la llamada a consulta abonado, como se ve en la figura 105.

Si el usuario está registrado, es decir, el servidor ha respondido con el identificador de usuario se llama al fichero ***winGestion.js***, que es el fichero que crea la interfaz principal de la aplicación.

7.2. winGestion

Al igual que en en ***win_Entry*** se crea la ventana ***winGestion***.

La primera vez que entra en esta interfaz se crea una vista, ***viewBienvenida***, para mostrar los datos de la próxima reserva que tiene un usuario.

Para crear ***viewBienvenida*** se crea un nuevo objeto de tipo ***viewBienvenida***. Primero se realiza una consulta POST al servidor para las próximas reservas y a continuación se crea la vista con elementos básicos para colocar la matrícula y una tabla para presentar los datos de la consulta. En caso de que el usuario no tuviera ninguna reserva saldría un mensaje en la interfaz indicándolo.

Para el menú se creará una tabla con los diferentes ítems a los que se puede acceder. Para acceder a él se creará un ***listener*** para que cuando se haga ***click*** en la esquina superior derecha se desplace y aparezca, como se muestra en el código de la figura.

Además se usa una variable ***menuView*** con la propiedad abierto, es un booleano, que permite a la aplicación saber si el menú está abierto o cerrado.

```
Ti.App.addEventListener("pulsaIcon",function(){
        if (menuView.abierto == false) {
                menuView.animate({
                        left:0,
                        duration:300,
                curve:Ti.UI.ANIMATION_CURVE_EASE_IN_OUT
                });
                sombra.animate({
                opacity: 0.5,
                        left: 0,
                duration:0,
                curve:Ti.UI.ANIMATION_CURVE_EASE_IN_OUT
                });
                menuView.abierto = true;
      } else {
                menuView.animate({
                        left:'-60%',
                        duration:300,
                        curve:Ti.UI.ANIMATION_CURVE_EASE_IN_OUT
                });
                sombra.animate({
                        opacity: 0,
                        left: '-100%',
                        duration: 0,
                        curve:Ti.UI.ANIMATION_CURVE_EASE_IN_OUT
                });
                menuView.abierto = false;
        };
});
```

Figura 106.Evento para abrir y cerrar el menú

Para cargar la vista adecuada se usará un ***switch***. En este ***switch*** se evaluará el valor que se ha pulsado de la tabla del menú. Los casos del ***switch*** son:

- ***Reservas Usadas***: En este caso se crea una nueva vista que llamará al fichero ***viewTodasReservas.js*** para crearla y se le pasarán los parámetros necesarios para hacer una consulta al servidor como se explicó en el aparatado "*5.2.5 Petición POST* ", la sentencia de llamada a la función es la de la figura.

```
viewInformacion = new
viewTodasReservas(args.matricula,args.id_usuario,args.uri,"010119990000","311220992359",
true,true,e.rowData.title);
```

Figura 107.Paso de parámetros para la vista de Reservas Usadas

Los parámetros que se le pasan son:

  - Matrícula.
  - Identificación de usuario.
  - URIs.
  - Fecha de llegada.
  - Fecha de salida.
  - Reservas usadas, que será *true* en casa de que se haya usado y *false* en caso de que la reserva no haya sido usada, el sexto parámetro de la función de la figura 107.
  - Reservas válidas, que será *true* en caso de que no se haya anulado una reserva y *false* en

caso de que se haya anulado una reserva, el séptimo parámetro de la función de la figura 107.

- ***Reservas Anuladas***: En este caso ocurre lo mismo que en el caso de Reservas Usadas, para crear la vista se llama a ***viewTodasReservas.js***, lo único que cambian son los parámetros que se le pasan para realizar la petición al servidor. Se pondrán a *false*, los campos usadas y válidas.

```
viewInformacion = new
viewTodasReservas(args.matricula,args.id_usuario,args.uri,"010119990000","311220992359",
false,false,e.rowData.title);
```

Figura 108.Paso de parámetros para la vista de Reservas Anuladas

- ***Próximas Reservas***: Este caso es igual que los otros dos anteriores cambiando los parámetros de petición al servidor. Se pondrán a *false*, el campo usadas y *true* el campo válidas.

```
viewInformacion = new
viewTodasReservas(args.matricula,args.id_usuario,args.uri,"010119990000","311220992359",
false,true:,e.rowData.title);
```

Figura 109.Paso de parámetros para la vista de Próximas Reservas

- ***Estaciones de recarga***: Se crea una nueva vista, en este caso la vista será un mapa, usando la función ***viewEstacionesRecarga***.

```
viewInformacion = new viewEstacionesRecarga(args.matricula,fechaHoy,args.uri);
```

Figura 110.Paso de parámetros para la vista de Estaciones de Recarga

En este caso se le pasan los parámetros:

  - Matrícula.
  - La fecha actual.
  - URIs.

- ***Realizar una Reserva***: En este caso se crea una nueva vista usando la función ***viewRealizaReserva***, para crear la vista que permita al usuario definir los parámetros de su reserva.

```
viewInformacion = new
viewRealizaReserva(args.matricula,args.uri,"nodoDesconocido",{"Codigo":"S00",
"Descripcion":""},"");
```

Figura 111. Paso de parámetros para la vista de Realiza una Reserva

Para esta vista se le pasan los siguientes parámetros:

  - Matrícula.
  - URIs.
  - El número de nodo, se usa este parámetro cuando se crea la vista desde el mapa de Estaciones de Recarga, como se comentó en el apartado "***viewEstacionesRecarga***". Si no es el caso se usará ***nodoDesconocido.***

- El identificador del nodo y los servicios que ofrece el nodo se pasan vacíos en el caso de que el usuario realice directamente una reserva.

- ***Preferencias***: Esta vista carga el menú de opciones.

```
viewInformacion = new viewPreferencias();
```

Figura 112.Llamada a la función que crea la vista de Preferencias

En este caso no se le pasa ningún parámetro.

7.3. viewBienvenida

En ***viewBienvenida*** se crea una vista, al entrar tras la ventana de autenticación se entra en la ventana de gestión y se carga la vista de bienvenida, esta vista contiene los elementos de la última reserva que contendrán los siguientes elementos:

- ***titulo_seccion***: Es un cuadro de texto para mostrar la eitqueta: *"Bienvenido"*, y los demás parámetros son para el estilo, figura 113, las propiedades que se usan para el estilo se explican en el apartado *"9. Diseño de la aplicación"*.

```
//Titulo de la sección
var titulo_sesion = Titanium.UI.createLabel({
        color: '#088A08',
        font: { fontSize:32},
        text: "Bienvenido",
        left: "10%",
        textAlign: Ti.UI.TEXT_ALIGNMENT_LEFT,
        bottom: '95%',
        width: Ti.UI.SIZE, height: Ti.UI.SIZE
});
```

Figura 113.Definición de Título de la sección

- ***matriculaLabel***: Es un cuadro de texto que muestra la matrícula del usuario.

- ***table:*** Es la tabla que contendrán todas las filas con la información que se recibe en la respuesta del servidor SRG.

```
//Tabla para presentar los datos
var table = Ti.UI.createTableView({
        separatorColor: "transparent",
        left: '10%',
        top: '15%',
        minRowHeight: 50,
        touchEnabled: false,
});
```

Figura 114.Tabla para representar los datos de la reserva

- Se crea un nuevo objeto consulta para pedir los datos de la próxima reserva. Para ello se crea un nuevo objeto ***postConexion***. Se le pasa el objeto ***datosConsultaReserva*** que contiene los parámetros que se le envían al servidor.

```
consulta = new
postConexion(datosConsultaReserva,uri.uri_base,uri.infoReserva,
function(tabledata){
```

Figura 115.Llamada a la función postConexion

Cuando se llama a la función secundaria, ***callback***, se comprueba si se han recibido próximas reservas.

Si el usuario tiene reservas se recorren los campos de la reserva y se introducen en la tabla. Para introducir los datos en la tabla se usa la función ***fila_columnn,*** esta función está definida en ***funcionesGenericas.js***, construye dos columnas con los argumentos que se le pasen. La llamada a la función se ve en la figura 116.

```
var tituloTable = fila_column({nombre: "Próxima reserva:",
fontSize: 24,color1: 'darkgray',color2: 'darkgray',left1:
'5%',left2: '50%'});
tbl_data.push(tituloTable);
```

Figura 116.Llamada a la función fila_column

Para incluir funciones, cómo la de ***fila_column***, se incluye el nombre del fichero con la sentencia, ***Titanium.include([Nombre del archivo que contiene la función])***.

Una vez que tenemos una fila, se introduce en la tabla con la sentencia ***push.***

Para terminar se incluirá la posición del nodo en el mapa, para ello se crea el objeto mapa, incluido en el archivo ***mapa.js***. Se le pasan las coordenadas del nodo y crea una chincheta en la posición que se encuentra el nodo.

```
objmapa = new
mapa('40%','90%','5%','5%',tabledata[0].Latitud,tabledata[0].Longitud,tabledat
a[0].Ciudad,tabledata[0].Lugar,function(pMap){
        mapaIdNodo = pMap;
});
```

Figura 117.Objeto mapa

7.4. viewEstacionesRecarga

Esta vista se carga cuando se llama desde el menú. En esta vista se crea un objeto llamado ***datosConsultaPlazas*** que almacena una petición de plazas libres en los nodos cercanos que tengan un umbral, radio en km con centro la posición del smartphone, para hacer una petición POST al servidor como en el apartado *"7.5 viewBienvenida".*

En esta vista se crea un mapa, para ello se ha utilizado el modulo ***ti.map-android-2.1.3.zip***, este módulo hay que descargárselo del *Marketplace* de Appcelerator, de la siguiente URL:

https://marketplace.appcelerator.com/home

Además para poder utilizarlo se necesita registrar la aplicación en Google, para ello hay que seguir la siguiente guía, se encuentra en inglés en la página de documentación de Titanium:

http://docs.appcelerator.com/titanium/latest/#!/guide/Google_Maps_v2_for_Android

Los pasos a seguir son:

- Crear un proyecto en Google API y activar Google Map ANdorid API v2.
- Obtener y añadir una llave de Google Maps API para el proyecto. Para generar la clave se necesita:
    - El ID de la aplicación que se encuentra en el fichero de ***tiapp.xml***.
    - Y una huella certificada SHA-1. Para obtener esta huella se usará la herramienta de JDK para obtener cualquier certificado de depuración. La clave está almacenada el directorio raíz del Titanium SDK en:

mobilesdk/<platform>/<sdk_version>/android/dev_keystore

Una vez realizado el proceso de activación para usar los mapas, la definición de la vista del mapa se realiza como se ve en la figura 118.

```
//Se crea una vista del mapa
var viewMapa = Map.createView({
        mapType:Map.NORMAL_TYPE,
        animate: true,
        userLocation: true,
        height: '100%',
        width: '100%'
});
```

Figura 118.Visa mapa

Para crear un punto, se tiene que usar el método ***createAnnotation*** que proporciona la librería de los mapas, esta anotación se usa para posicionar los nodos en el mapa, que se reciben de hacer una petición POST de solicitud de plazas al servidor. Este punto se marcará con una chincheta en el mapa, además se le ha añadido dos propiedades:

- Servicios: Almacena los servicios que tiene el nodo.
- IdNodo: Identificador del nodo.

Se puede pulsar la chincheta que sitúa a cada nodo y aparece un cuadro que contiene los servicios. Para conseguir esta acción se creado la función ***includeServicios***, que crea una tabla, cada fila es un servicio. También se ha creado un switch, en este caso es un ***check***, asociado a un evento. Y por último un botón que dirige al usuario a la interfaz para realizar una reserva, este botón se crea con la función ***botonReserva*** y se añade a la tabla que muestra los servicios del nodo.

Además se ha añadido la función ***posiciónLocal***, código de la figura 119. Esta función añade la posición del usuario al mapa.

```
// ----GEOLOCALIZACIÓN----
function posicionLocal(callback){
	Titanium.Geolocation.accuracy =
Titanium.Geolocation.ACCURACY_BEST;
	// Distancia de error. Este valor es en metros
	Titanium.Geolocation.distanceFilter = 10;
	// Se obtiene la posción actual

	Titanium.Geolocation.getCurrentPosition(function(e){
		if (e.error) {
			alert('No se puede obtener la posición
actual del usuario');
			return;
		} else {

		callback(e.coords.longitude,e.coords.latitude);
		};
	});
};
```

Figura 119.Posición del usuario

7.5. viewPreferencias

Para crear esta vista se ha utilizado una tabla y la función ***definirOpcion***, que define las filas de las tablas y se usa para los parámetros que se pueden configurar.

```
//Función para definir las opciones
function definirOpcion (titulo,observacion,dato){
	var row = Ti.UI.createTableViewRow({
		hasChild: true,
		opcion: dato
	});
```

Figura 120.Función para definir cada opción

Una propiedad que contiene el objeto ***row***, fila es ***hasChild: true***, es decir, esta propiedad indica que si se pulsa sobre la fila, o se desliza, o se realiza una acción sobre la fila ejecutará un evento. Esta opción también se encuentra en el menú.

Cuando se pulsa sobre una fila se abrirá un cuadro de diálogo, para crearlo se ha usado el código de la figura 121.

```
var dialogOpcion = Ti.UI.createOptionDialog();
...
	dialogOpcion.androidView = output_text;
	dialogOpcion.buttonNames = ['Eliminar','Aceptar'];
	...                              dialogOpcion.show();

	dialogOpcion.addEventListener('click', function(e) {
		Ti.API.info(e.index);
		...
```

Figura 121.Creación de un diálogo

Según la fila que se haya pulsado aparece un mensaje.

7.6. viewTodasReservas

Esta vista al igual que la vista de preferencias crea una tabla en la ventana. Esta vista es usada por tres ítems del menú:

- Próximas Reservas.
- Reservas Usadas.
- Reservas Anuladas.

Cada fila de la tabla representa una reserva y para crear una fila se llama a la función ***filaConsulta***, código de la figura 123.

Cada reserva presenta los siguientes campos:

- Una imagen, que depende de la opción que ha elegido el usuario, verde próximas reservas, azul reservas usadas y rojo reservas anuladas. Se usa la función ***createImageView***, que se ve en el código de la figura 123.
- El identificador de reserva.
- Fecha de entrada.
- Fecha salida.

Para la consulta se usa una petición POST con la estructura de datos de la figura 122.

```
//Formato datos de envio
var datosConsultaReserva = {
        IdUsuario: id_usuario,
        Matricula: matricula,
        FechaDesde: fechaInicio,
        FechaHasta: fechaFin,
        Usada: Usada,
        Valida: Valida,
};
```

Figura 122.Formato de datos de consulta de reserva

A cada fila tiene asociado un evento activo, ***hasChild:true***, para abrir una ventana nueva y mostrar la información de la reserva más detallada. Para crear esta ventana se ha usado ***winDetalleReserva.js***. Esta ventana se ha construido con la misma estructura que para la ***viewBienvenida.***

A esta ventana se le ha añadido un botón en sólo dos casos:

- Botón de anular reserva en próximas reservas: Hace una petición GET al servidor SRG para anular la reserva.
- Botón de recibo para ver los recibos de una manera detalla. Se muestran en un diálogo que se muestra al hacer click sobre el botón.

```
//Función que crea la vista de una reserva
function filaConsulta(IdReserva,fechaEntrada,fechaSalida,infoDatos,imagen){
	var row = Ti.UI.createTableViewRow({
		hasChild:true,
		datos: infoDatos,
	});
	var viewCoche = Ti.UI.createImageView({
		image: imagen,
		width: '18%',
		height: '90%',
		left: '5%'
	});
	var labelIdReserva = Ti.UI.createLabel({
		text: ">>" + " Id Reserva: ",
		color: 'gray',
		font: { fontSize:20},
		left: '30%',
		bottom: '70%'
	});
	var labelDatoIdReserva = Ti.UI.createLabel({
		text: IdReserva,
		color: 'black',
		font: { fontSize:20},
		left: '50%',
		bottom: '70%'
	});
	var labelFechaEntrada = Ti.UI.createLabel({
		text: "Fecha de Entrada: ",
		color: 'gray',
		font: { fontSize:14},
		left: '30%',
		bottom: '40%',
	});
	var labelDatoFechaEntrada = Ti.UI.createLabel({
		text: fechaEntrada,
		color: 'gray',
		font: { fontSize:14},
		left: '55%',
		bottom: '40%',
	});
	var labelFechaSalida = Ti.UI.createLabel({
		text: "Fecha de Salida: ",
		color: 'gray',
		font: { fontSize:14},
		left: '30%',
		bottom: '10%'
	});
	var labelDatoFechaSalida = Ti.UI.createLabel({
		text: fechaSalida,
		color: 'gray',
		font: { fontSize:14},
		left: '55%',
		bottom: '10%'
	});
	//Se incluyen todos los datos a la fila de la tabla
	row.add(viewCoche);
	row.add(labelIdReserva);
	row.add(labelDatoIdReserva);
	row.add(labelFechaEntrada);
	row.add(labelDatoFechaEntrada);
	row.add(labelFechaSalida);
	row.add(labelDatoFechaSalida);
	return row;
};
```

Figura 123.Código para creación de una fila de la tabla de consultas

7.7. viewRealizaReserva

Esta vista también crea una tabla. En esta tabla el usuario podrá definir los parámetros de su reserva.

Al igual que en las otras vistas se crea un objeto para almacenar los datos de la petición POST al servidor SRG, figura 124.

```
var datosReserva = {
        IdNodo: IdNodo,
        Matricula: matricula,
        TipoCarga: {"Codigo": "0", "Descripcion": ""},
        CargaLlegada: 0,
        CargaRecarga: 0,
        Precio: 0,
        FechaLlegada: "ddmmyyyyhhmm",
        FechaSalida: "ddmmyyyyhhmm",
        Insitu: false,
        ServiciosSolicitados: [],
};
```

Figura 124.Objeto para el envío de datos de la Reserva

Se crea la función ***definirBarra***, esta función crea una descripción del campo y una barra que se puede deslizar para definir:

- El precio.
- La carga de llegada.
- La carga a recargar.

Cada barra llama a un evento cada vez que se modifica para almacenar el valor en los campos correspondientes del objeto ***datosReserva.***

```
//Función para elegir la hora que se quiere realizar
la reserva
function definirDiaHora(callback){
        var fechainicial;
        var rowDia = Ti.UI.createTableViewRow();
        var pickerDia = Ti.UI.createPicker({
                left:'5%',
                type: Ti.UI.PICKER_TYPE_DATE,
        });
        var pickerHora = Ti.UI.createPicker({
                left: '50%',
                type: Ti.UI.PICKER_TYPE_TIME,
                format24: true,
        });

        pickerHora.selectionIndicator = true;
        pickerDia.selectionIndicator = true;
        rowDia.add(pickerHora);
        rowDia.add(pickerDia);
        //Se crea un evento que devuelve la hora y el
día elegido por el usuario cuando se produce un cambio

        rowDia.addEventListener('change',function(e){
                var fecha;
```

Figura 125.Código de la función definirDiaHora

Para la fecha y la hora se crea un objeto que es del tipo de la fecha y otro objeto el tiempo. Como se ve en la figura 125.

Se crea un evento en la fila de definición de las fechas para que cuando se modifique algún valor quede capturado, el evento se ve en la figura 125.

Para definir el tipo de carga se ha creado una lista. Primero se debe crear una matriz de datos, como se ve en la figura 126. A continuación se añaden los datos al objeto y se crea un evento para que cuando se modifique el valor devuelva el valor nuevo, este cambio se realiza con un ***callback.*** Para definir el tipo de carga se usa la función ***definirServ.***

```
//Función para definir el servicio que desea el usuario
function definirServ(titulo,callback){
	var rowCarga = Ti.UI.createTableViewRow();
	var labelTitulo = Ti.UI.createLabel({
		text: titulo,
		width: '45%',
		left: '5%',
		color: 'black',
		height: 'auto',
		textAlign: Ti.UI.TEXT_ALIGNMENT_LEFT,
		font: { fontSize:18 },
	});
	var pickerCarga = Ti.UI.createPicker({
		left: '50%',
		backgroundColor: 'white',
		color: 'black',
	});
	var data = [];
			data[0]=Ti.UI.createPickerRow({title:'Elige una opción',codigo:"TP00"});
	data[1]=Ti.UI.createPickerRow({title:'Carga Inductiva',codigo:"TP01"});
	data[2]=Ti.UI.createPickerRow({title:'Carga Capacitiva',codigo:"TP02"});
	data[3]=Ti.UI.createPickerRow({title:'Recarga Rápida',codigo:"TP03"});

	pickerCarga.add(data);
	pickerCarga.selectionIndicator = true;
		pickerCarga.addEventListener('change',function(e){
		callback(e.row);
	});

	rowCarga.add(labelTitulo);
	rowCarga.add(pickerCarga);
	return rowCarga;
};
```

Figura 126.Código función definirServ

Para la opción de ***Pago Insitu*** se usa un switch como en el caso de los servicios de ***viewEstacionesRecarga***.

Una vez creados todos los campos se incluyen en la tabla con el método ***push***.

Para terminar se ha creado un botón con un evento que crea el objeto comprueba y mira si todos los datos están correctos.

Si lo están se abre una nueva ventana, ***winReserva.open***. Esta ventana es una tabla basada en la estructura de ***winDetallesReserva***.

Se crea una tabla y se muestra un resumen de la reserva del usuario. Para realizar la reserva se pulsa el botón que llama a un evento que hace una petición POST al servidor con los datos almacenados.

8. Estructura de datos de la aplicación

A lo largo de la aplicación se han definido objetos que permiten una mayor independencia de código.

- Un objeto que contiene todas las URIs del servidor SRG.

```
var uri = {
        uri_base: 'http://subsistemareservaguiado.appspot.com/IHM/',
        reserva: 'UserReserve',
        infoReserva: 'UserInfoReserve',
        plazas: 'RequestPlaces',
        anular: 'NullReserve/',
        usuario: 'suscriber/',
};
```

Figura 127.Objeto que contiene las URIs

- Para el envío de datos al servidor se ha creado un objeto que tenga como propiedades los datos que se envían al servidor.

```
var datosConsultaReserva = {
        IdUsuario: id_usuario,
        Matricula: matricula,
        FechaDesde: fechaInicio,
        FechaHasta: fechaFin,
        Usada: Usada,
        Valida: Valida,
};
```

Figura 128.Estructura de datosConsultaReserva

Para acceder a cada dato se usa la notación por puntos. Al contrario de lo que sucede en otros lenguajes orientados a objetos, como por ejemplo Java, para asignar el valor de una propiedad no es necesario que la clase tenga definida previamente esa propiedad, aunque para realizar un código más limpio en la aplicación se ha decido definir todas las propiedades antes de utilizarlas.

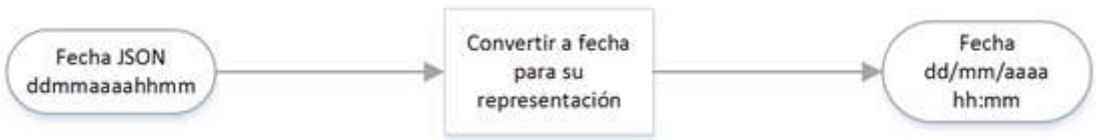


Figura 129.Conversión fecha JSON-Formato Estándar-JSON

```
function fecha(fechaJSON,callback){
        var fecha;
        fecha = fechaJSON.substring(0,2) + "/" +
                        fechaJSON.substring(2,4) + "/" +
                        fechaJSON.substring(4,8) + "\n" +
                        fechaJSON.substring(8,10) + ":" +
                        fechaJSON.substring(10,12);
        callback(fecha);
};
//Se exporta el resultado
module.exports = fecha;
```

Figura 130.Código para la conversión fecha JSON-Formato Estándar-JSON

- Para las fechas se ha implementado una función que convierte la fecha que se obtiene del objeto JSON al formato de fecha dd/mm/aaaa hh:mm, como se ve en la figura 115.

Para cambiar el formato de la fecha se llama a la función fecha como función principal y cuando se modifica se llama a la función secundaría para devolverlo al principal, consiguiendo que la fecha sea la que se ha recibido del servidor.

9. Diseño de la aplicación

Para diseñar las interfaces se han usado las propiedades de estilo que contienen los objetos de Javascript. Principalmente se ha trabajado con porcentajes para poder ajustarlo lo máximo posible a las diferentes pantallas que se encuentran actualmente en el mercado, aunque también se pueden usar píxeles si se usara una única resolución.

Algunas propiedades que se han usado son:

- ***Heigth***: Altura del elemento.
- ***Width***: Anchura del elemento.
- ***Bottom***: Es la posición a la que se colocará desde la parte inferior. Esta propiedad es muy útil para colocar los elementos en la vista o ventana.
- ***Color***: El color del texto.
- ***BackgroundColor***: El color de fondo del elemento, aunque también se puede usar ***BackgroundImage*** para usar una imagen de fondo.
- ***Left*** o ***right***: Es la posición a la que se colocará el elemento desde el margen izquierdo o derecho respectivamente.

Un ejemplo del uso de algunas de estas propiedades se ven en el código siguiente para la creación de un título de sección.

```
//Titulo de la sección
var titulo_sesion = Titanium.UI.createLabel({
        color: '#088A08',
        font: { fontSize:32},
        text: "Bienvenido",
        left: "10%",
        textAlign: Ti.UI.TEXT_ALIGNMENT_LEFT,
        bottom: '95%',
        width: Ti.UI.SIZE, height: Ti.UI.SIZE
});
```

Figura 131.Creación de Título de sección

Existen gran variedad de propiedades para el diseño, y a los elementos se le pueden asociar gran cantidad de eventos. Todas las propiedades, métodos y eventos de los elementos de Titatium se pueden consultar en la documentación, el enlace se encuentra en la bibliografía.

Para pasar la fecha de formato JSON al formato ***dd/mm/yyyyy hh:mm***, se ha usado la función ***fechaJSON*** del archivo de *funcionesGenericas.js*.

```
//Función para escribir la fecha en formato JSON para el servidor
function fechaJSON (args){
        var f_actual = args.fecha;
        var fechaHoy;

        if(f_actual.getDate()>= 10 && (f_actual.getMonth()+1)>= 10){
                fechaHoy = f_actual.getDate()+""+(f_actual.getMonth()+1)+""+f_actual.getFullYear()+"0000";
        } else if(f_actual.getDate()< 10 && (f_actual.getMonth()+1)>= 10){
                fechaHoy = "0"+f_actual.getDate()+""+(f_actual.getMonth()+1)+""+f_actual.getFullYear()+"0000";
        } else if(f_actual.getDate()>= 10 && (f_actual.getMonth()+1)< 10){
                fechaHoy = f_actual.getDate()+"0"+(f_actual.getMonth()+1)+""+f_actual.getFullYear()+"0000";
        } else {
                fechaHoy = "0"+f_actual.getDate()+"0"+(f_actual.getMonth()+1)+""+f_actual.getFullYear()+"0000";
        };

        return fechaHoy;
};
```

Figura 132.Código para convertir el formato de la fecha