ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE
TELECOMUNICACIÓN
UNIVERSIDAD POLITÉCNICA DE CARTAGENA

Proyecto Fin de Carrera

Desarrollo de Apps Multiplataforma: Un caso comparativo

AUTOR: Inmaculada Mercader Sola

DIRECTOR: Pedro Sánchez Palma

SEPTIEMBRE 2014

Autor: Inmaculada Mercader Sola

E-mail del Autor: imercsola@gmail.com

Director: Pedro Sánchez Palma

E-mail del Director: pedro.sanchez@upct.es

Título del PFC: Desarrollo de Apps Multiplataforma: Un caso comparativo

Resumen:
El proyecto consiste en mostrar las diferentes formas en las que se pueden desarrollar apps multiplataforma, y los diferentes tipos de apps multiplataforma que existen.
Para ello se utilizarán los frameworks Appcelerator Titanium, PhoneGap y jQuery Mobile para desarrollar una aplicación ejemplo. Que servirá para realizar una comparativa entre estos frameworks, que se utilizará para que si en un futuro, se desea crear una aplicación multiplataforma, seleccionar el tipo de app multiplataforma y el framework más adecuados.

Titulación: Ingeniería Técnica de Telecomunicación, especialidad Telemática

Departamento: Tecnologías de la Información y las Comunicaciones

Fecha de presentación: Septiembre 2014

Índice

1. Introducción.

1.1. Motivación.

Este proyecto pretende explicar los diferentes enfoques que se pueden utilizar para desarrollar apps multiplataforma. Para ello se explicaran los pros y contras de desarrollar una app desde un punto de vista nativo o un punto de vista multiplataforma**.** Además, se profundizará en las diferentes estrategias que se podrán utilizar para desarrollar apps multiplataforma, es decir, se explicarán las diferencias entre seguir una estrategia híbrida para desarrollar apps híbridas o interpretadas, o seguir una estrategia basada en web para desarrollar web apps. Con este objetivo se realizará una app basada en un caso de estudio previamente presentado, que permitirá realizar un análisis exhaustivo de las diferencias y características entre los diferentes tipos de apps multiplataforma, para así poder realizar una comparativa entre ellos.

Además, este caso de estudio también permitirá presentar una serie de frameworks para el desarrollo de apps multiplataforma:

- Desde el punto de vista de las apps híbridas, si se desarrollan en PhoneGap.
- Desde el punto de vista de las apps interpretadas, si se desarrollan en Appcelerator Titanium.
- Desde el punto de vista de las web apps, si se desarrollan con JQuery Mobile.

De los cuales se realizará un análisis de sus capacidades y características.

1.2. Objetivos del Proyecto.

Los objetivos de este proyecto serán por lo tanto:

- Explicar los pros y contras de los diferentes puntos de vista empleados para desarrollar aplicaciones móviles.
- Realizar un estudio de las diferentes estrategias que se pueden utilizar para desarrollar apps multiplataforma, es decir, realizar un estudio sobre las apps híbridas, apps interpretadas y web apps.
- Presentar un caso de estudio para desarrollar una app, que permita realizar un análisis de las apps híbridas, apps interpretadas y web apps.
- Desarrollar la app mediante los frameworks PhoneGap, Appcelerator Titanium y jQuery Mobile.
- Señalizar las capacidades y características de cada uno de estos frameworks.
- Utilizar la app desarrollada para realizar una comparativa de las capacidades de los diferentes tipos de apps multiplataforma desarrolladas.
- Utilizar esta comparativa para que en el caso futuro de una aplicación, se elija la estrategia y framework más adecuados.

2. El mercado global.

2.1. El mercado global de los dispositivos móviles.

Worldwide Device Shipments by Segment (Thousands of Units)

Device Type	2012	2013	2014
PC (Desk-Based and Notebook)	341,273	303,100	281,568
Ultramobile	9,787	18,598	39,896
Tablet	120,203	184,431	263,229
Mobile Phone	1,746,177	1,810,304	1,905,030
Total	**2,217,440**	**2,316,433**	**2,489,723**

Source: Gartner (October 2013)

Según estudios recientes el mercado de los dispositivos informáticos alcanzará los 2320 millones de unidades en el 2013, lo que supone un incremento del 4,5 % respecto al 2012.

Sin embargo, la venta de ordenadores de sobremesa y los portátiles han disminuido un 11,2 %, mientras que la venta de Tablets y Smartphones han aumentado un 53,4 y un 3,7 % respectivamente.

Estos datos muestran un cambio en la mentalidad del usuario, que al vivir en un entorno cada vez más conectado a las redes móviles, prefiere dispositivos que le permitan tener acceso a ellas desde cualquier lugar. Y por lo tanto, le proporcionen más movilidad, flexibilidad, y que puedan ser adquiridos a un menor coste.

Sin embargo, también se debe tener en cuenta aquellos usuarios que necesitan dispositivos que le permitan realizar su trabajo. Por lo que no es de extrañar que exista la necesidad de combinar la funcionalidad de un PC con la flexibilidad de una Tablet, esta podría ser una de las posibilidades de desarrollo de nuevos dispositivos en el futuro. Pero por el momento según las previsiones lo que único que se puede saber es que el mercado de los dispositivos móviles seguirá creciendo.

2.2. Situación de las Plataformas.

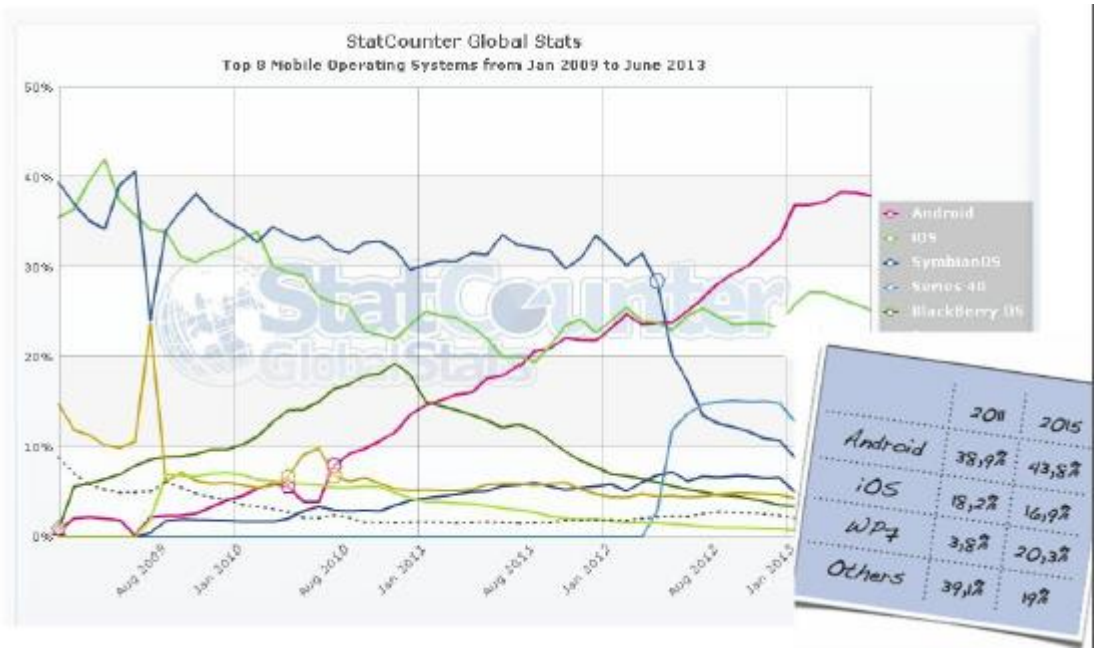


Figura 1: Top de las 8 primeras Plataformas Móviles.

Como se puede observar en la anterior gráfica, con datos del año 2013, la plataforma más utilizada, es Android, con una presencia del 38,9%. Mientras que, en segundo lugar se sitúa iOS con un 18,2%, en tercer lugar Windows Phone con un 3,8%, y el resto, el 39,1%, se lo reparten otras plataformas. Por lo tanto, Android le saca una considerable ventaja al resto de plataformas en la actualidad.

Pero esta situación podría cambiar en el año 2015 debido a la adquisición de Nokia por parte de Microsoft. Algunos analistas apuntan al aumento de la presencia de Windows Phone concretamente hacia un 20,3% en detrimento de iOS que sufriría la disminución a un 16,9% mientras que el resto de sistemas operativos disminuirían su presencia al 19%. Sin embargo, no se prevé que Android se vea perjudicado, ya que seguiría aumentando su presencia hasta un 43,8 %.

Por lo tanto, nadie debe extrañarse del interés de algunas compañías en desarrollar aplicaciones para cada una de estas plataformas, ya que es un mercado emergente, y sólo en 2013 el mercado de las aplicaciones para móviles movió 26 billones de dólares, y fueron descargadas un total de 102 billones de apps. Además se prevé que en 2017 se podría llegar a los 268 billones de descargas, y al beneficio de 77 billones de dólares anuales.

Así pues este es sin duda uno de los mercados con mayores beneficios y garantías en la actualidad, pero también es extremadamente competitivo y hay que pensar en la funcionalidad y la estrategia utilizadas para desarrollar las aplicaciones. Es decir, se debe decidir si se desea crear una app multiplataforma o una app nativa, que herramientas de desarrollo se desea utilizar, y que funciones debe implementar la aplicación. Todas estas cuestiones serán desarrolladas en los siguientes capítulos.

3. Aplicaciones nativas.

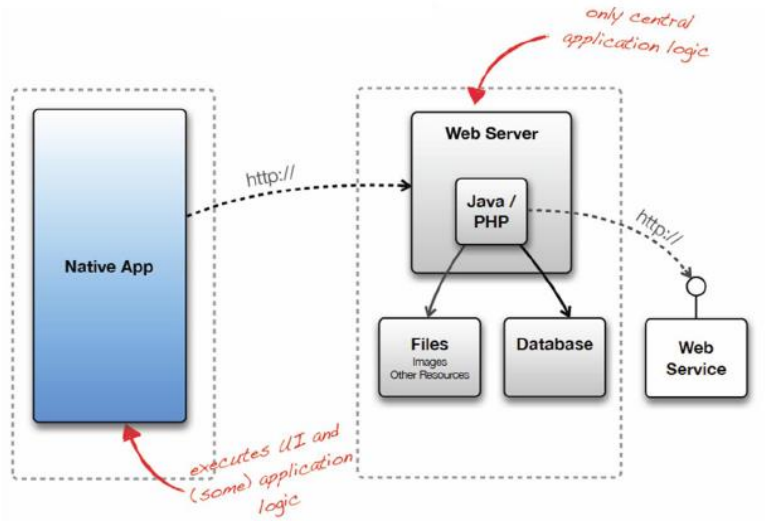


Figura 2: Arquitectura de una App Nativa.

Las aplicaciones nativas son aquellas que han sido implementadas en el leguaje nativo de cada una de las plataformas móviles, es decir, han sido desarrolladas para funcionar en una plataforma específica. Así que si se crea una app para que funcione en Android deberá implementarse su código en Java, y si es para iOS se deberá implementar en Objetive-C.

3.1. Entornos de desarrollo.

Se emplearan los siguientes entornos de desarrollo dependiendo de la plataforma:

Plataforma	IDE	Emulador	Lenguaje
Android	Eclipse	Android SDK	Java
Windows Phone	Microsoft Visual Studio	Windows Phone SDK	C# y .NET
iOS	xCode	iOS SDK	Objetive-C

3.2. Pros y contras.

Pros:

- Este tipo de apps tienen acceso a todos los recursos del dispositivo.
- Los desarrolladores tiene acceso a soporte y material educativo.
- Emplean al máximo las características de los dispositivos, alcanzando el máximo rendimiento posible.
- Pueden ser publicadas en las App stores, permitiendo que los usuarios accedan a su descarga y actualizaciones, y además facilita la monetización.

Contras:

- Se necesitan desarrolladores específicos para cada plataforma, por lo que aumenta el coste de desarrollo.
- Se necesita la autorización de las App stores para publicar o actualizar las apps, y pagar licencias para la publicación, lo que puede suponer un aumento en el tiempo y coste.
- Su código no es reutilizable, no se puede utilizar código para desarrollar apps para distintas plataformas, ya que se utiliza el lenguaje específico de cada una de ellas.

3.3. Cuándo se usaran.

- Si se busca un rendimiento y aspecto nativos.
- Cuando se quiera que estén disponibles en los App Store y Market Place.
- Se desea alcanzar el rendimiento máximo de los dispositivos.

4. Aplicaciones multiplataforma.

Las aplicaciones multiplataforma son aquellas que independientemente de su código son capaces de funcionar en plataformas distintas. Para su desarrollo se suelen emplear tecnologías web, siguiendo 3 estrategias distintas:

- Basada en web.
- Híbrida.
- Con enfoque dirigido por modelos.

A continuación serán explicadas cada una de estas estrategias para desarrollar apps multiplataforma.

4.1. Estrategia Mobile Web.

4.1.1. Server-side web.

Son aquellas apps que han sido construidas mediante tecnologías web, es decir, mediante una combinación de HTML, CSS y JavaScript, conocida como HTML5. Y que son ejecutadas en un navegador web. Por lo tanto, una aplicación web es una versión de una página web que ha sido optimizada para funcionar correctamente en dispositivos móviles. Por lo que deberán ser diseñadas teniendo en cuenta las características de este tipo de dispositivos, como son, entre otras, el tamaño de la pantalla, el zoom, o el reconocimiento táctil.

La arquitectura de este tipo de aplicaciones consistirá en una aplicación que corre en el sitio web móvil y es construida con HTML, CSS y JavaScript. Como en cualquier aplicación web, la representación de los datos se realiza en el lado del cliente, mediante un navegador web, y el lado del servidor se utiliza para ejecutar el UI y la lógica de la aplicación y proporcionar acceso a los datos.

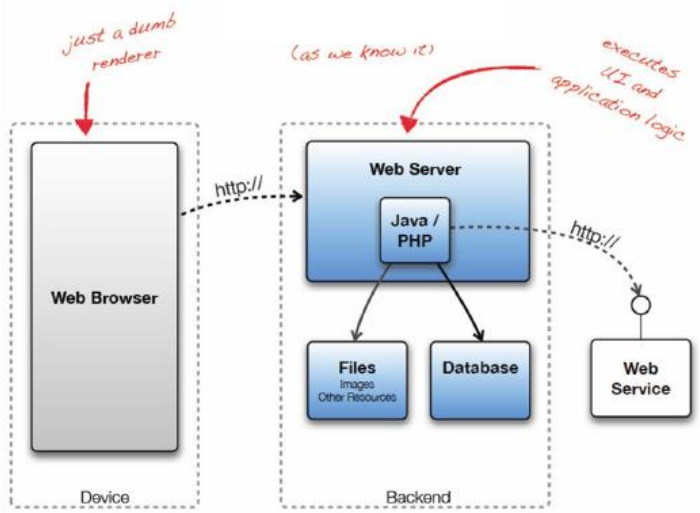


Figura 3: Arquitectura de una App Server-Side Web.

4.1.1.1. Frameworks o herramientas de desarrollo.

Existen en la actualidad un gran número de frameworks para el desarrollo de este tipo de aplicaciones, que han sido capaces de mejorar su diseño y comportamiento, de manera que cada uno de los websites desarrollados se visualicen y comporten de la misma forma en navegadores de escritorio que en navegadores de dispositivos móviles. Como ejemplos, se pueden encontrar:

- Foundation: Desarrollado por Zurb, puede obtenerse en http://foundation.zurb.com.
- Bootstrap: Desarrollado por Twitter, se puede descargar en http://getbootstrap.com.

4.1.1.2. Pros y Contras.

Pros:

- Funcionan en todas las plataformas.
- Siempre están actualizadas, ya que no se necesita pasar por un proceso de validación para ser publicadas.
- Su coste de desarrollo es pequeño, no se necesita pagar por licencias o por el entorno de desarrollo.
- Se utilizan tecnologías web para desarrollarlas, lo que facilita la implementación de código.

Contras:

- El desarrollador tendrá un control escaso de como las aplicaciones se visualizan y actúan en los navegadores.
- Este tipo de aplicaciones tendrán acceso limitado a los recursos de los dispositivos. Tendrán acceso a funcionalidades como el giroscopio, pero no tendrán acceso, por ejemplo, a recursos como la cámara o la agenda de contactos.
- Su rendimiento tendrá relación directa con la capacidad de acceso a la red, el usuario percibirá los fallos de conectividad y las interrupciones, además sólo podrá acceder si está conectada a la red.
- El proceso de monetización se dificultará, ya que este tipo de aplicaciones no podrán distribuirse vía App store y sólo se encontrarán en los navegadores.

4.1.1.3. Cuando se usarán.

- Si se necesita una página web accesible desde todas las plataformas, que sirva para proporcionar un servicio determinado.

4.1.2. Client-Side Web (HTML5).

Esta estrategia se seguirá si se desea desarrollar una aplicación web que tenga el aspecto y comportamiento de una aplicación nativa.

Este tipo de aplicaciones se ejecutarán en un navegador web y utilizarán JavaScript para ejecutar el interfaz de usuario y la lógica de la aplicación. JavaScript se comunicará asíncronamente con el Backend, que contiene sólo la lógica central, lo que elimina la necesidad de recargar la aplicación cada vez que se hace una petición al servidor.

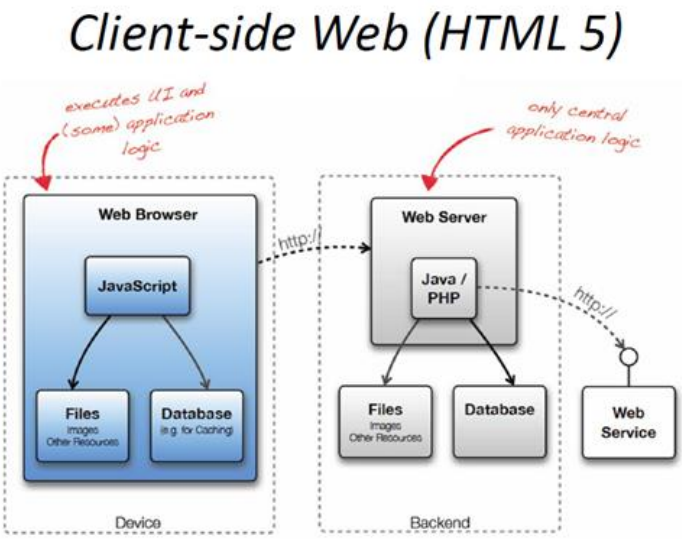


Figura 4: Arquitectura de una App Client-Side Web.

Hay que destacar que existen dos formas distintas para desarrollar este tipo de aplicaciones, la primera centrándose sólo en el interfaz de usuario UI, y la segunda utilizando elementos del UI y englobando la infraestructura MVC (Modelo Vista Controlador).

4.1.2.1. Frameworks o herramientas de desarrollo.

Entre los frameworks que el desarrollador es capaz de emplear para desarrollar este tipo de apps, se encuentran:

- iUI.js: Es un framework libre desarrollado por Joe Hewitt, y que se puede obtener en http://iui-js.org.
- Sencha Touch: Es un framework desarrollado por Sencha, con licencia comercial y que se puede obtener en http://sencha.com/products/touch.
- jQuery Mobile: Es un framework para desarrollo de software libre, desarrollado por JQuery Project y que se puede descargar en http://jquerymobile.com.

Más adelante, se realizará un estudio de las características jQuery Mobile.

4.1.2.2. Pros y Contras.

Pros:

- Son compatibles con todas las plataformas móviles.

- Su coste de desarrollo es pequeño, no se necesita pagar ni por las licencias ni por los frameworks.
- Su curva de aprendizaje es muy rápida, el programador tiene acceso a documentación y a foros que le permitirán aprender a desarrollarlas.
- Siempre estarán actualizadas, no será necesario esperar a su validación por parte de un App store.

Contras:

- No tendrán acceso a todos los recursos de los dispositivos.
- Su rendimiento será muy inferior al de una aplicación nativa, estará limitado por la capacidad del buscador y la calidad de acceso a la red.
- No se podrán encontrar en las App stores, sólo se podrá acceder en los buscadores.

4.1.2.3. ¿Cuándo se usarán?

Se usaran si:

- Se necesita crear un servicio accesible de todas las formas posibles.
- Se necesita crear algo que tenga el aspecto de una aplicación, pero que no necesite tener acceso a todos los recursos de los dispositivos.

4.2. Estrategia Híbrida.

4.2.1. Aplicaciones Híbridas.

Las aplicaciones híbridas son aplicaciones multiplataforma que han sido desarrolladas mediante tecnologías web, y empaquetadas en código nativo. Y que además proporcionan acceso al hardware y resto de recursos de los dispositivos.

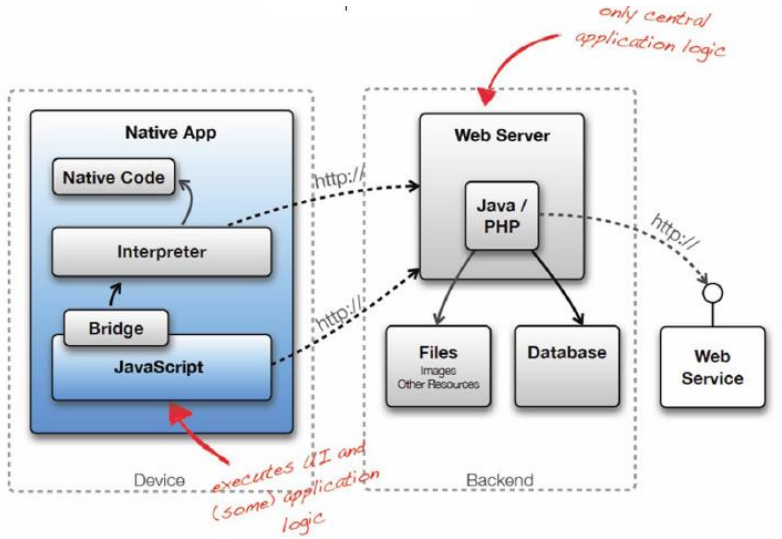


Figura 5: Arquitectura de una App Híbrida.

Por lo tanto, si se observa su arquitectura consiste básicamente, en un empaquetador nativo que contiene una aplicación web-based, y que por lo tanto será mostrada en un navegador

web. El cual ejecutará el UI y la lógica de la aplicación mediante código JavaScript que será interpretado línea a línea a código nativo.

Por lo que para el usuario final, la aplicación híbrida se visualizará como una aplicación nativa.

4.2.1.1. Herramientas de desarrollo o Frameworks.

El framework más famoso de este tipo es PhoneGap, que es libre y de código abierto. Y que en sus inicios fue desarrollado por Nitobi y que más adelante fue comprado por Adobe Systems. Además, se trata de una distribución de Apache Cordova. Puede ser descargado en http://phonegap.com .

4.2.1.2. Pros y Contras.

Pros:

- Este tipo de apps pueden ser usadas en casi todas las plataformas.
- Al ser desarrolladas mediante tecnologías web la curva de aprendizaje es bastante rápida, ya que se tiene acceso a numerosos tutoriales y a soporte técnico.
- Se tiene acceso a todos los recursos, ya sean de hardware como de software, de los dispositivos.
- Su coste de desarrollo es inferior al de una app nativa, ya que no es necesario ni pagar por licencias, ni por el uso de frameworks.
- Este tipo de apps son accesibles mediantes las App stores, lo que facilita su distribución, monetización y actualización.

Contras:

- Su rendimiento es inferior a la de una aplicación nativa en ciertos aspectos. Ya que por un lado, su uso de la memoria es superior, y por otro lado, es muy difícil conseguir el aspecto y comportamiento de una app nativa.

4.2.1.3. Cuando se usarán.

Se usaran si:

- Se quieren desarrollar apps basadas en HTML5, con el objetivo de disminuir el tiempo y los costes de desarrollo.
- Se desea crear un app que tenga acceso a todos los recursos de los dispositivos.

4.2.2. Aplicaciones Interpretadas.

Las aplicaciones interpretadas son aplicaciones cuyo código ha sido escrito en código App Script, el cual ha sido compilado a byte-code en tiempo de diseño, e interpretado en tiempo de ejecución como código nativo para cada una de las plataformas soportadas. Esto quiere decir que el tiempo de ejecución actúa como una capa que abstrae las diferencias entre las diferentes plataformas. Por lo tanto, este tipo de aplicaciones implementan y ejecutan los recursos de los dispositivos, mediante el uso de la capa de abstracción que permite tener acceso a cada uno de los APIs de las plataformas soportadas. Esta arquitectura elimina algunos problemas de rendimiento que las apps híbridas sufren para proporcionar un aspecto y comportamiento nativo, ya que utilizan la mayoría de componentes de UI nativos. Mientras que la lógica de la aplicación es manejada en una plataforma independiente utilizando una serie de comandos en XML u otros lenguajes descriptores.

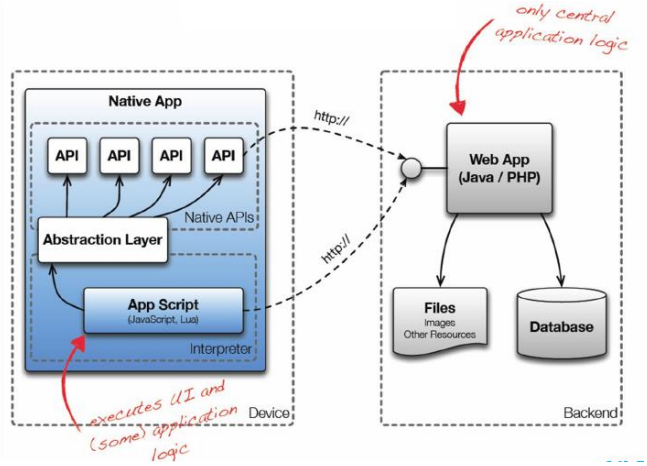


Figura 6: Arquitectura de una App Interpretada.

4.2.2.1. Frameworks o herramientas de desarrollo.

Para poder desarrollar este tipo de apps podrán ser utilizadas las siguientes frameworks:

- Appcelerator Titanium: Es un framework libre desarrollado por Appcelerator Inc. Y que puede ser conseguido en http://appcelerator.com. Este framework será explicado más adelante.
- Rhomobile: Es un framework de código abierto desarrollado por Motorola, y que puede conseguirse en http://rhomobile.com.
- Corona Mobile: Es una framework desarrollado por Corona Labs Inc. y creado por Walter Luh, el cual se puede conseguir en http://www.coronalabs.com.

4.2.2.2. Pros y contras.

Pros:

- Su curva de aprendizaje es rápida, al ser desarrolladas mediante una versión modificada de JavaScript, del que se tiene acceso a numerosos tutoriales.
- Se tiene acceso a todos los recursos de cada una de las plataformas soportadas. Lo que aproximará la experiencia de usuario a una experiencia nativa.

- Su coste de desarrollo será pequeño, ya que se tendrá acceso a herramientas de desarrollo gratuitas, pero se deberán pagar algunas licencias.
- Este tipo de apps pueden estar disponibles en las App stores, lo que facilita su monetización y disponibilidad.
- Su rendimiento será muy parecido a la de una aplicación nativa.

Contras:

- Su código no es 100% reutilizable, y en ocasiones deberá adaptarse parcialmente según la plataforma para la que se ejecute.
- No todas las plataformas relevantes son soportadas por ahora, en concreto Windows Phone.

4.2.2.3. Cuando se usarán.

Se usaran si:

- Se desea desarrollar una aplicación con el mismo código para todas las plataformas, es decir, se desea utilizar el mismo lenguaje.
- Se desea tener acceso a todos los recursos de los dispositivos.
- Se desea obtener una aplicación con un rendimiento, aspecto y funcionamiento similares a la de una app nativa.

4.3. Enfoque dirigido por modelos.

En este tipo de apps el código es escrito en un único lenguaje de programación mientras que un compilador traduce el código a código nativo para cada plataforma. Con esta estrategia un código nativo puede producirse desde un único código base.

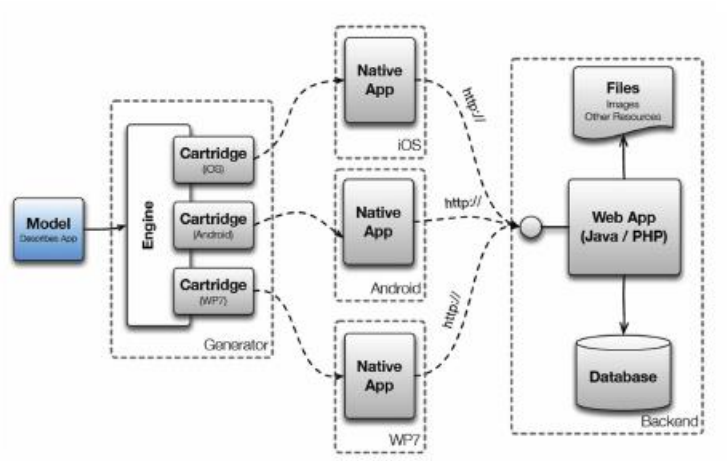


Figura 7: Arquitectura de una App de Compilación Cruzada.

4.3.1. Frameworks o herramientas de desarrollo.

Entre los diferentes tipos de frameworks para el desarrollo de este tipo de apps existen:

- MonoTouch: Es un framework desarrollado por Xamarin, gratuito y de código abierto, que se puede conseguir en http://mono-project.com.
- Applause: Es un framework desarrollado por Applause Project, y que puede ser descargado en http://github.com/applause/applause.

4.3.2. Pros y Contras.

Pros:

- Este tipo de apps permiten tener acceso a todos los recursos hardware y software de los dispositivos, de la misma forma que las aplicaciones nativas.
- Se tiene acceso a documentación y tutoriales.
- Se tiene acceso a soporte gratuito.
- Su rendimiento es muy parecido al de una aplicación nativa.
- Se puede tener acceso a ellas a través de las App stores, lo que facilita su monetización, obtención y mantenimiento.

Contras:

- Se utilizará como lenguaje de programación lenguajes como C# o similares.
- Su coste de desarrollo será considerable, al aumentar el tiempo de desarrollo y tener que pagar algunas licencias.
- No son válidas para todas las plataformas, concretamente para BlackBerry.
- No es un método adecuado para desarrollar apps complejas.

4.3.3. Cuándo se usarán.

Se usarán si:

- Se desea utilizar un único lenguaje de programación para todas las plataformas.
- Se desea acceder a todos los recursos de los dispositivos, de igual manera que una aplicación nativa.
- Se desean apps con un rendimiento, aspecto y comportamiento de una app nativa sin serlo.

5. Frameworks multiplataforma.

Existen diferentes frameworks para desarrollar diferentes tipos de apps multiplataforma. Estos son:

- Appcelerator Titanium: Permite desarrollar apps interpretadas.
- PhoneGap: Permite desarrollar apps híbridas.
- jQuery Mobile: Permite desarrollar web apps.

5.1. Appcelerator Titanium.

Titanium es un framework libre y de código abierto que permite crear apps multiplataforma utilizando tecnologías web, concretamente una versión modificada de JavaScript.

Titanium Mobile SDK permite crear, ejecutar, y empaquetar apps para iOS, Android y BlackBerry, empleando APIs JavaScript. Por lo tanto, estas apps se ejecutan en una máquina JavaScript que invoca APIs nativas. Esto quiere decir que en realidad lo que se hace es crear una app nativa utilizando JavaScript en lugar de Java u Objetive-C.

Titanium es a alto nivel una combinación de:

- Máquinas Titanium SDK: Estas se componen de un conjunto de scripts de Python y soportan máquinas que trabajan con máquinas SDK nativas. Las máquinas Titanium combinan el código JavaScript, un intérprete de JavaScript, y los recursos, en una aplicación binaria que será instalada en un emulador o en un dispositivo.
- Mobile APIs: Es una API basada en JavaScript que permite acceder al UI nativo y a otros componentes de la app.
- Titanium Studio: Es un IDE gratis desarrollado por Titanium basado en Eclipse. Se puede utilizar para escribir, probar y depurar las apps desarrolladas. Tiene integradas plantillas y apps ejemplo para facilitar el aprendizaje y desarrollo de apps. Y permite acceder a actualizaciones de Titanium SDK, y el uso de módulos.
- Módulos: Titanium se construye de una serie de módulos que extienden algunas funciones de la API. Entre estos módulos se pueden encontrar módulos que sirven para conectarse e interactuar con Facebook, o para interactuar con mapas nativos.
- Servicios en nube: Titanium permite acceder a una serie de servicios en nube, como por ejemplo, estadísticas de la frecuencia con la que una app desarrollada es utilizada y en que plataformas.

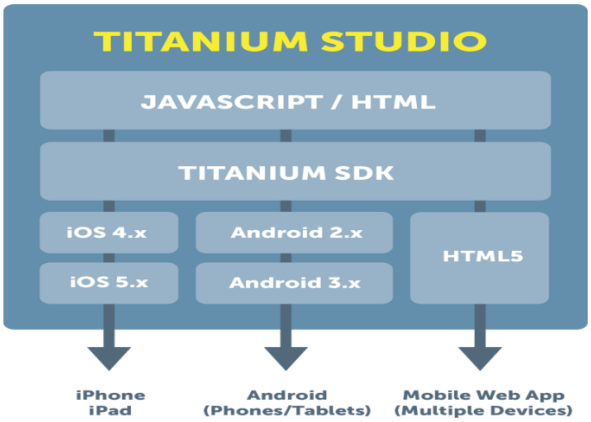


Figura 8: Arquitectura de Titanium Studio.

Según su arquitectura, Titanium funciona como un puente entre el código JavaScript y la plataforma nativa correspondiente. En primer lugar se encuentra la app desarrollada en JavaScript, en segundo lugar Titanium SDK y las APIs, y en último lugar el usuario de la app. Analizando esta arquitectura se llega a la conclusión de que la app desarrollada en JavaScript lo que hace es llamar a las APIs de Titanium que ejecutan acciones tales como abrir ventanas, utilizar la cámara o crear botones. El puente Titanium que forma parte del SDK, conocido como Kroll, lo que hace es traducir esas llamadas a su equivalente nativo, al igual que si los eventos ocurren en el lado nativo los traducen a llamadas JavaScript.

Por lo tanto, lo que hace Titanium es ayudar a los desarrolladores a crear apps nativas mediante el lenguaje JavaScript.

Entre las plataformas soportadas por Titanium se encuentran iOS, Android y BlackBerry. De los cuales se podrán utilizar una serie de características particulares, además de una serie de características comunes. Ya que aunque al desarrollar una app parte del código es reutilizable entre distintas plataformas, para que dicha app funcione en todas ellas, debe sufrir una serie de modificaciones, por lo que el código no es 100% reutilizable.

Entre las características de las plataformas soportadas se encuentran:

- UI Nativo.
- Eventos.
- Información sobre dispositivos: Se puede obtener información sobre los dispositivos como: su dirección IP, su identidad, el nivel de batería, o el nombre de su Plataforma o Sistema Operativo.
- Información sobre las redes conectadas: Se podrá obtener el estado de la conexión y el tipo de red que está conectado el dispositivo. Además se puede obtener, mediante el evento *change,* los cambios de conexión con la red.
- Notificaciones: Se podrán utilizar alertas, diálogos de confirmación y avisos.
- Vibración.
- Gestos: Maneja gestos como agitar.
- Acelerómetro.

- Animaciones: Se podrán añadir animaciones a los elementos del UI, como son animaciones 2D y 3D, animaciones básicas, y transiciones entre las diferentes ventanas, aunque esto último sólo sea soportado por los dispositivos iOS.
- Trabajar con datos locales: Se podrán trabajar con datos locales mediante el uso de: Propiedades, Bases de Datos SQLite, y el FileSystem (Sistema de Archivos).
- Trabajar con datos remotos: Se podrá trabajar con servidores remotos mediante el Protocolo HTTP, con el que se pueden trabajar con datos JSON y XML, y subir y bajar archivos de un servidor remoto.
  O utilizar servicios web SOAP, que es el formato para los datos XML devueltos por el servicio web. Pero su uso no es recomendable, por su coste de datos superior a los datos XML, y la necesidad de traducir el formato XML a JavaScript.
- Servicios de localización: En los dispositivos se podrán utilizar: geolocalización, brújula, y mapas nativos. Todo ello para obtener la posición actual del dispositivo.
- Multimedia: Se podrá grabar y reproducir audio, reproducir video, y acceder a la galería nativa de los dispositivos. Además de poder acceder a la cámara frontal y trasera, y capturar imágenes y videos con ellas.
- Acceso a Facebook.
- Agenda.
- Calendario.
- Giroscopio.
- Herramientas de desarrollo IDE propia: Titanium Studio.

5.1.1. Pros y Contras.

Pros:

- Su software es libre y de código abierto.
- Permite desarrollar apps válidas para plataformas diferentes.
- Las apps resultantes poseen el comportamiento, aspecto y rendimiento de una app nativa.
- Proporciona un IDE propio, Titanium Studio, gratuito y basado en Eclipse.
- Emplea tecnologías web para desarrollar apps, por lo que los costes y tiempo de desarrollo disminuirían.
- Se tiene acceso a numerosos tutoriales y otras herramientas de ayuda para el desarrollo de las apps.
- Posee numerosas APIs que permiten tener acceso a recursos de las apps nativas.

Contras:

- El código de las apps solo es reutilizable de un 60 al 90% entre distintas plataformas móviles, ya que cada una de ellas poseen APIs propias.
- Su soporte de desarrollo es de pago.
- Para poder desarrollar aplicaciones para iOS será necesaria un ordenador con el Sistema Operativo de Apple instalado.
- Solo son soportadas las siguientes plataformas móviles: iOS, Android y BlackBerry.

5.2. PhoneGap.

PhoneGap es una tecnología libre y de código abierto que permite desarrollar apps nativas para múltiples plataformas mediante tecnologías web.

El UI para apps de PhoneGap es creado utilizando HTML, CSS, y JavaScript. La capa de UI de una app de PhoneGap consiste en un navegador web que ocupa el 100% de la pantalla del dispositivo.

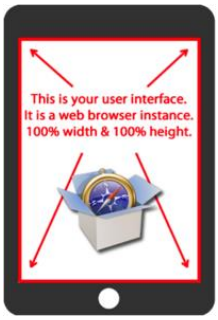


Figura 9: UI de una App de PhoneGap.

Las apps desarrolladas en PhoneGap son, en definitiva, apps desarrolladas empleando tecnologías web, que dan como resultado un archivo de aplicación binario que puede distribuirse a través de distintas plataformas. Las plataformas móviles soportadas por PhoneGap son: iOS, Android, Windows Phone, BlackBerry, Bada y Tizen.

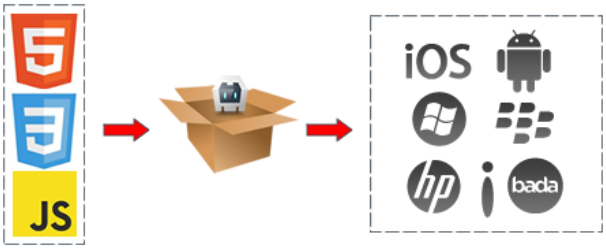


Figura 10: Empaquetamiento y Distribución de una App de PhoneGap.

En el caso de apps para iOS el archivo binario resultante sería tipo IPA, mientras que en Android sería tipo APK, y así sucesivamente con cada una de las distintas plataformas. Estos formatos resultantes en apps de PhoneGap, son los mismos formatos resultantes de cualquier app nativa, por lo que estas apps pueden ser distribuidas a través de sus App stores.

Por lo tanto, lo que PhoneGap permite es, en definitiva, desarrollar apps nativas empleando tecnologías web, es decir, permite desarrollar una app con un código 100% reutilizable para cada una de las plataformas soportadas.

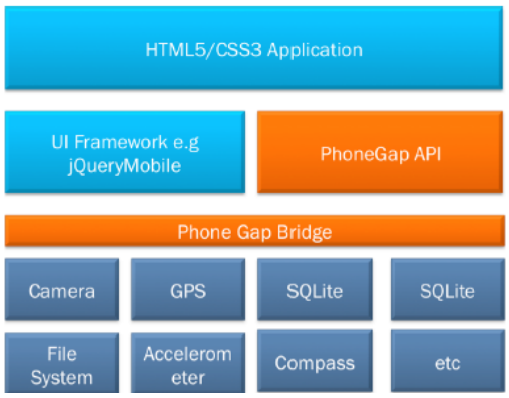


Figura 11: Arquitectura de PhoneGap.

Si se analiza la arquitectura de PhoneGap, se llega a la conclusión de que es un framework, que consiste esencialmente en una librería en JavaScript, que permite a las app desarrolladas acceder a características de los dispositivos. Así que, Phonegap tendrá un API que permitirá desarrollar código en JavaScript capaz de implementar funcionalidades nativas. Y además, actuará como puente entre ese código JavaScript y las características nativas, es decir, se encargará de la comunicación del código con las características de los dispositivos.

Entre las características implementadas por el API de PhoneGap se encuentran:

- Manejo de Eventos.
- Obtener información sobre el dispositivo: Se podrá obtener el nombre del dispositivo, su identificación, el nombre de la Plataforma u Sistema Operativo, la versión del Sistema Operativo y la versión de PhoneGap que se ha utilizado.
- Obtener información sobre las redes conectadas.
- Acelerómetro.
- Manejar notificaciones: Se podrán utilizar alertas, diálogos de confirmación, avisos y notificaciones sonoras.
- Usar Vibración.
- Trabajar con datos locales: Se podrá trabajar con datos locales empleando: bases de datos SQLite, LocalStorage, y el FileSystem (Sistema de Archivos).
- Trabajar con datos remotos: Se podrán subir y bajar archivos a servidores remotos mediante HTTP.
- Servicios de localización: Se podrá obtener la geolocalización de un dispositivo, y utilizar su brújula.
- Multimedia: Se podrá acceder a la cámara y la galería de fotos, reproducir y grabar audio, y capturar video.
- Agenda de contactos.

5.2.1. Pros y Contras.

Pros:

- Es un software libre y de código abierto.
- Permite crear apps híbridas válidas entre distintas plataformas.
- El empleo de tecnologías web disminuye el tiempo y coste de desarrollo.

- PhoneGap proporciona tutoriales para aprender a desarrollar apps con dicha herramienta.
- Se tienen acceso a numerosos APIs que permiten controlar componentes hardware y software de los dispositivos.
- Gran parte de su código es reutilizable entre distintas plataformas.
- Soporta numerosas plataformas móviles: Android, iOS, BlackBerry, Windows Phone 7, Windows Phone 8, Bada y Tizen.

Contras:

- No cuenta con un IDE propio. Dependiendo de las plataformas se utiliza un IDE diferente: para Android y Tizen se utiliza Eclipse, para Windows Phone se emplea Visual Studio, en iOS se utiliza Xcode, y finalmente en BlackBerry se utiliza Apache Ant.
- No se pueden desarrollar apps para todas las plataformas desde todos los Sistemas Operativos. Es decir, para poder empaquetar apps de iOS se necesitaría un ordenador con el Sistema Operativo Apple, y para poder empaquetar apps para Windows Phone, se necesitará un ordenador con el Sistema Operativo Windows.
- Se desarrolla apps híbridas en las que el rendimiento será inferior a la de una app nativa. Estará limitado por el componente WebKit nativo, que leerá e interpretará el código web.
- Una pequeña parte del código dependerá de la plataforma. Por ejemplo, cuando se implemente el evento producido por un botón back se tendrá en cuenta que este será soportado por dispositivos Android, BlackBerry y Windows Phone.
- No provee componentes o controles a nivel de UI. Para conseguir que el UI tenga un aspecto nativo es necesario del uso de frameworks de desarrollo de web móviles como JQuery Mobile o Sencha Touch.

5.3. jQuery Mobile.

jQuery Mobile es un framework de UI libre y de código abierto, basado en HTML5, CSS3 y la librería JavaScript jQuery, con la que compartirá algunas características. Este simplificará el proceso de creación de páginas web para dispositivos móviles, desde la escritura del código HTML, la maquetación con CSS y la creación de efectos con JavaScript.

jQuery Mobile eliminará la necesidad del diseño específico del UI para cada plataforma, llevará al máximo nivel el mantra de “escribe menos, haz más”. Y en lugar de escribir una única aplicación para cada plataforma móvil, permitirá diseñar un único sitio web o aplicación que trabajará para cada una de ellas.

Además, incluirá un sistema de navegación de Ajax que soportará transiciones entre diferentes páginas que serán transformadas en peticiones Ajax. Y que permitirá insertar diferentes elementos en el DOM de la página original.

También dispondrá de herramientas CSS, que harán que no sea necesario el diseño de los gráficos de forma manual por parte del usuario, sino que se dispondrá de diferentes temas

gráficos que aplicarán automáticamente el aspecto de la página y de los elementos que la componen.

Entre las diferentes plataformas soportadas por este framework se encuentran: iOS, Android, BlackBerry, Bada, Windows Phone 7, Windows Phone 8, Meego y Tizen.

Entre las características soportadas por jQuery Mobile se encuentran:

- Selectores jQuery.
- Manipulación del DOM de un documento.
- UI Nativo.
- Acceso a temas predefinidos por jQuery Mobile, para definir el aspecto del UI.
- Eventos.
- Notificaciones: Se podrán utilizar alertas, diálogos de confirmación y avisos.
- Animaciones: Se podrán animar las transiciones entre páginas, y los componentes del UI.
- Acceso a Datos Remotos vía Ajax.
- Giroscopio.

5.3.1. Pros y contras.

Pros:

- Es un software libre y de código abierto.
- Permite crear web apps válidas entre distintas plataformas.
- El empleo de tecnologías web disminuye el tiempo desarrollo y costes de desarrollo.
- jQuery Mobile proporciona acceso a tutoriales y demos.
- El código de las apps es 100% reutilizable entre las distintas plataformas soportadas.
- Soporta numerosas plataformas móviles: Android, iOS, BlackBerry, Bada, Windows Phone 7, Windows Phone 8, Meego y Tizen.

Contras:

- No cuenta con un IDE propio. Además dependiendo de las plataformas se utilizaría un IDE diferente: para Android y Tizen se utiliza Eclipse, en Windows se emplea Visual Studio, en iOS se utiliza Xcode, y finalmente en BlackBerry se utiliza Apache Ant.
- No se pueden desarrollar apps para todas las plataformas desde todos los Sistemas Operativos. Es decir, para poder empaquetar apps de iOS se necesitaría un ordenador con el Sistema Operativo Apple, y para poder empaquetar apps para Windows Phone, se necesitará un ordenador con el Sistema Operativo Windows.
- Se desarrollan web apps en las que el rendimiento será inferior a la de una app nativa e incluso una app híbrida. Estará limitado por el componente WebKit nativo, que leerá e interpretará el código web.
- Es incapaz de acceder a recursos del hardware y software de las plataformas de los dispositivos. Para poder acceder a estos recursos será necesario el uso de librerías como PhoneGap.

6. Presentación de la aplicación: Mi Pizzería.

Esta aplicación permite realizar el pedido de forma totalmente gratuita, sin necesidad de realizar una llamada o enviar un SMS a la Pizzería. Pero para ello será necesario tener acceso a Internet, ya que el pedido se realizará a un servidor remoto, y además el usuario recibirá un email con la confirmación de que el pedido se ha realizado con éxito.

Para ello se siguen los pasos siguientes:

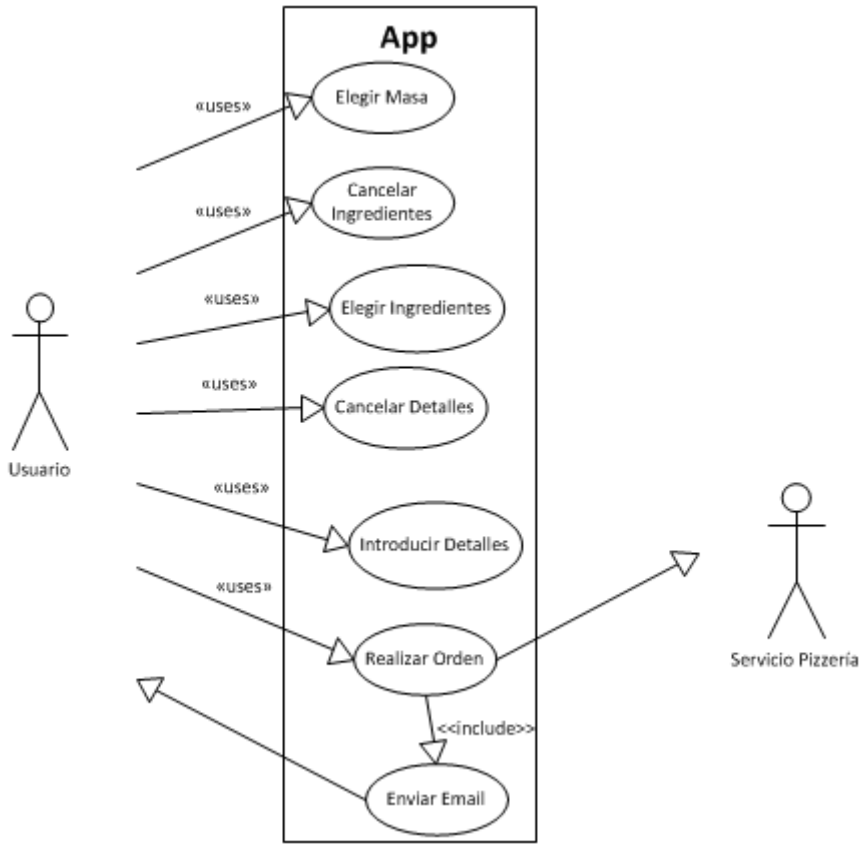
Figura 12: Aplicación explicada visualmente.

1. Se elige la masa de la pizza entre los siguientes tipos:
    a. Masa Hecha a Mano.
    b. Natural.
    c. De Pan.
    d. Rellena (De Queso).
    e. Fina y Crujientes.

    Y se pasa a la selección de los Ingredientes.
2. Se eligen los ingredientes (6 como máximo) que se desea que tenga la pizza, y se pasa a realizar el pedido o a cancelar la elección de los ingredientes, con lo que se volverá a realizar el paso anterior.
3. El usuario introduce los detalles del pedido, es decir, su nombre, dirección, población, código postal y su correo electrónico. Y realizará la orden al servidor de la pizzería, con el que podrán ocurrir las siguientes circunstancias:
    a. El pedido se realiza con éxito, con lo que recibe una confirmación, un email a la dirección proporcionada al servidor, y vuelve al paso 1.
    b. El pedido no se ha realizado con éxito, o porque no se puede acceder al servidor remoto o porque el email no ha sido recibido con éxito.

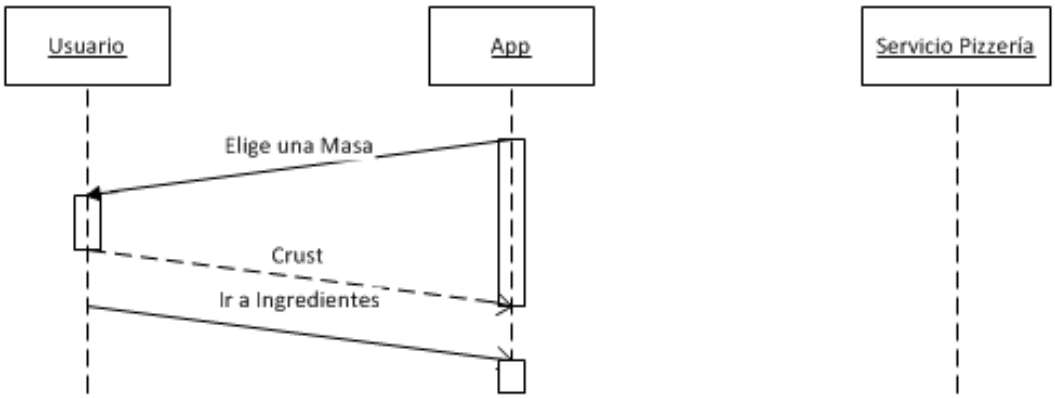
    Pero también puede cancelar el pedido, por lo que volverá al paso anterior.

6.1. Diagrama de Casos de Uso.

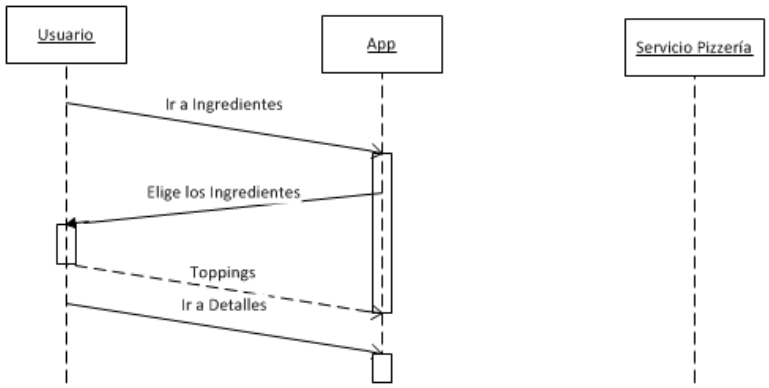


6.2. Diagramas de Secuencia.

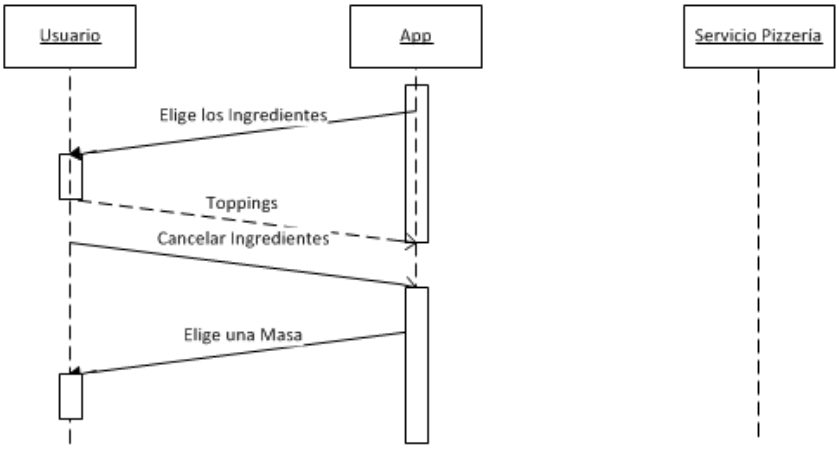
6.2.1. Diagrama de Secuencia de Elegir Masa.



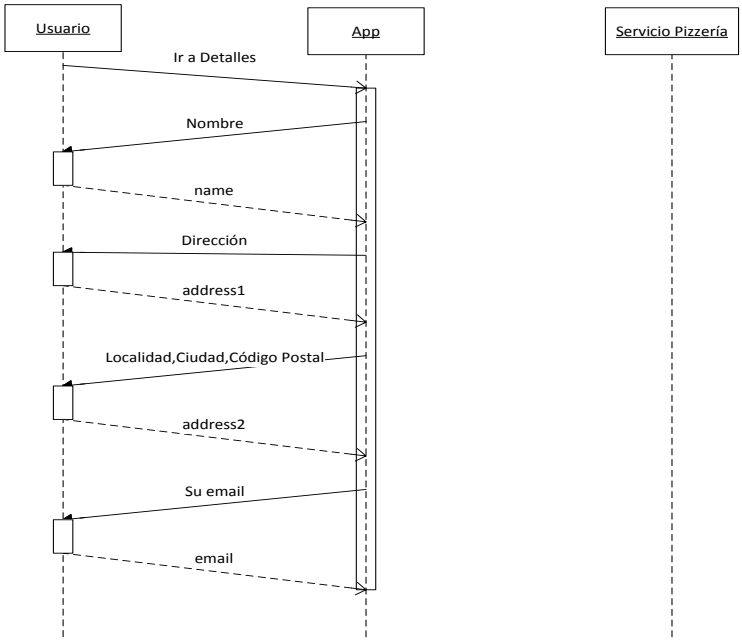
6.2.2. Diagrama de Secuencia de Elegir Ingredientes.



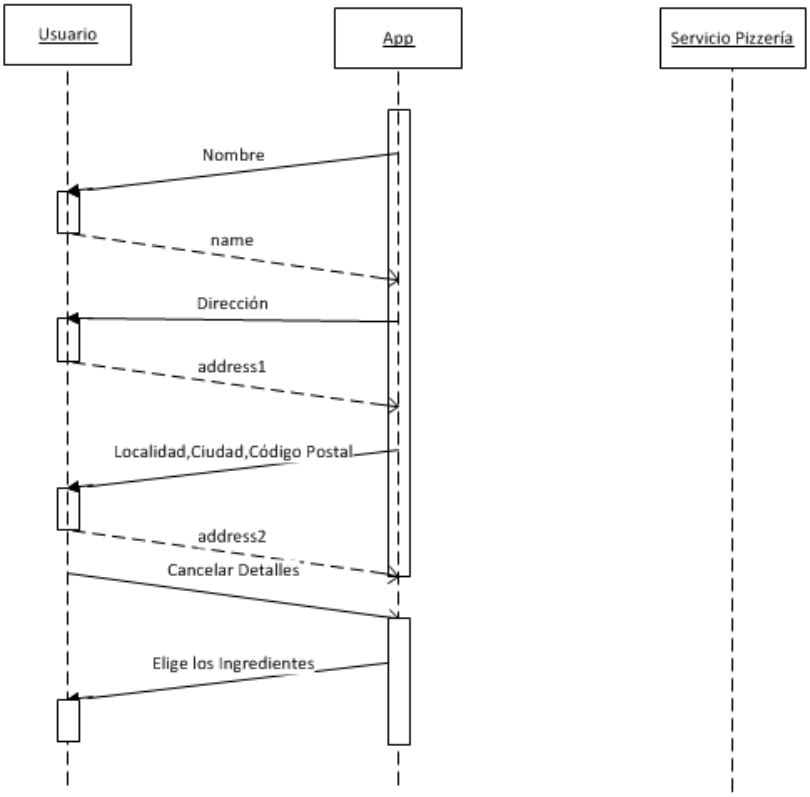
6.2.3. Diagrama de Secuencia de Cancelar Ingredientes.



6.2.4. Diagrama de Secuencia de Introducir Detalles.

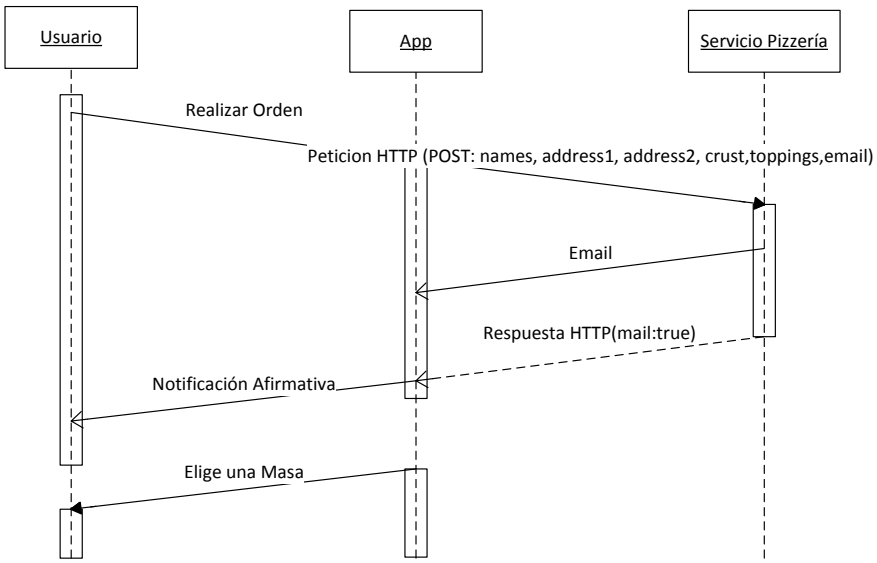


6.2.5. Diagrama de Secuencia de Cancelar Detalles.

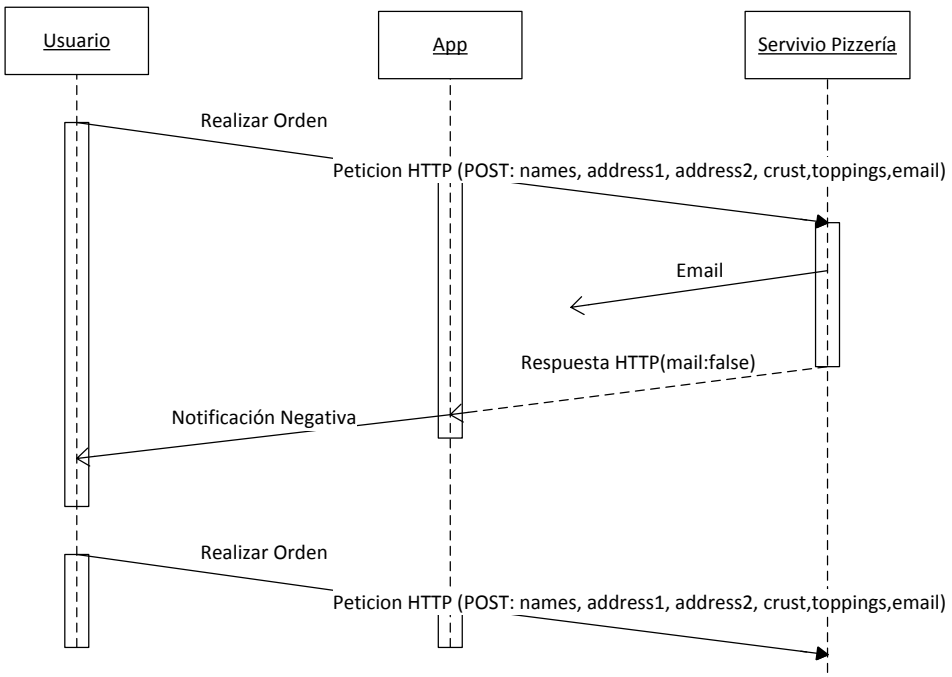


6.2.6. Diagrama de Secuencia de Realizar Orden.

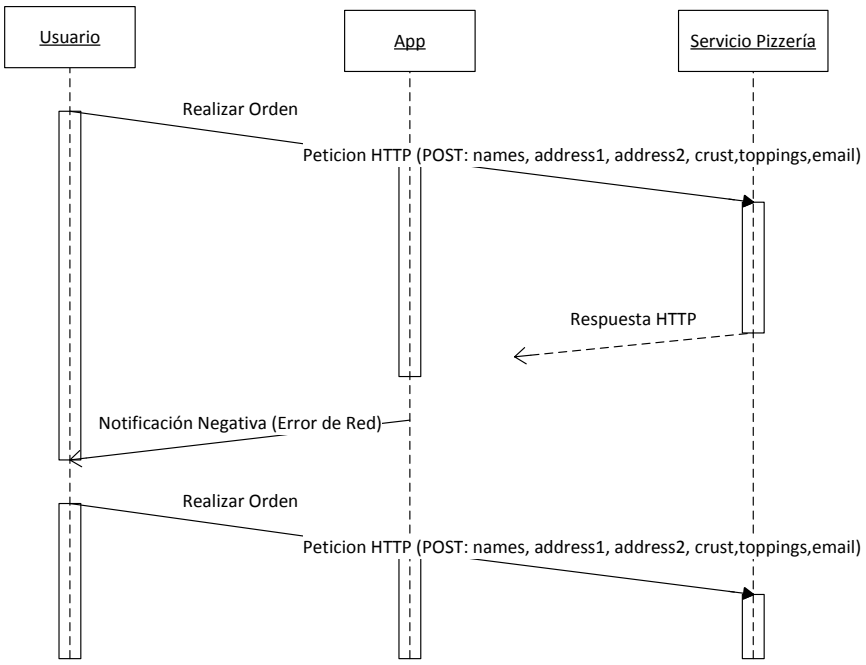
Caso 1: La Respuesta HTTP indica que la petición HTTP y el envío de Email se realizaron de forma satisfactoria.



Caso 2: La respuesta HTTP índica que el Email no se envió con éxito, con lo que el usuario deberá volver a realizar el orden.



Caso 3: No se recibe la respuesta a la petición HTTP, por lo que se mostrará un mensaje con el tipo de error que se ha producido, y el usuario deberá a volver a realizar el orden.



7. Implementación de aplicación.

7.1. Servidor Local.

Con el objetivo de probar la aplicación habrá que crear un servidor local. Y para ello existen diferentes programas software, utilizados para desarrollar sitios web y que suelen ser denominados de formas diferentes:

- WAMP: Si se usan las siguientes herramientas:
  - El Sistema Operativo Windows.
  - El servidor web Apache.
  - El gestor de base de datos MySQL.
  - Los lenguajes de programación PHP, Perl y Python.
- LAMP: Si se usan las siguientes herramientas:
  - El Sistema Operativo Linux.
  - El servidor web Apache.
  - El gestor de base de datos MySQL.
  - Los lenguajes de programación PHP, Perl y Python.
- MAMP: Si se usan las siguientes herramientas:
  - El Sistema Operativo Mac OS X.
  - El servidor web Apache.
  - El gestor de base de datos MySQL.
  - Los lenguajes de programación PHP, Perl y Python.

Entre los diferentes programas para crear servidores web se encuentran XAMPP, BitNami o AppServ. Pero será XAMPP la herramienta utilizada en el caso de estudio.

7.1.1. XAMPP.

XAMPP es un servidor multiplataforma, formado por un servidor web Apache, un sistema gestor base de datos MySQL, y los lenguajes de programación web PHP y Perl. Además incluirá lo servidores: Mercury para el envío de Emails, Tomcat para servlets JSP, y el servidor FileZilla para gestionar cuentas FTP.

Podrá obtenerse en la página https://www.apachefriends.org/index.html.En la cual podrá elegirse el instalador del Sistema Operativo de la máquina en la que se instalará el servidor local.

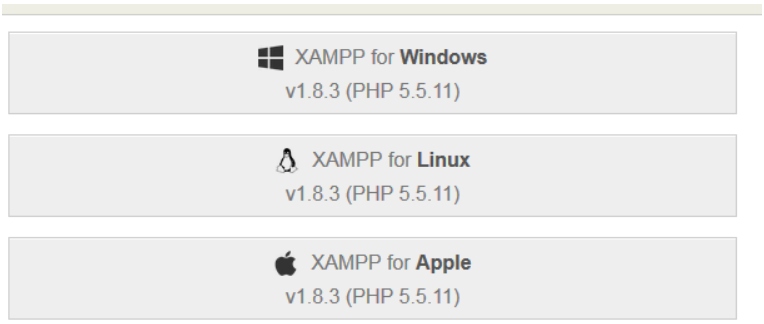


Figura 40: Enlaces para seleccionar el instalador de XAMPP para cada S.O.

7.1.2. Instalación de XAMPP en Windows 7.

Para instalar XAMPP en la máquina antes de esto es recomendable comprobar si hay un servidor local instalado, y para ello se debe abrir el navegador y escribir la dirección http://localhost , y si se obtiene el mensaje de error "No se puede conectar", entonces se procederá a instalar XAMMP.

Para ello se seguirán los pasos siguientes:

1. Se irá a https://www.apachefriends.org/index.html y se elegirá el instalador para Windows.
2. Se ejecutará el instalador XAMMP, el cuál mostrará los siguientes dos avisos:
    a. El primero referente al antivirus instalado en la máquina, indicando que este puede interferir en la instalación del software.

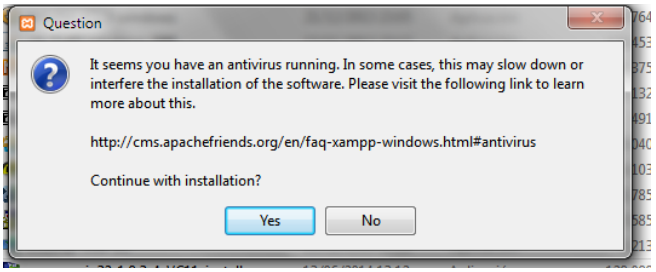


Figura 41: Instalador de XAMPP: Aviso por antivirus.

    b. El segundo si está activado el Control de Cuentas de Usuario, recordando que algunos de los directorios tienen permisos restringidos.

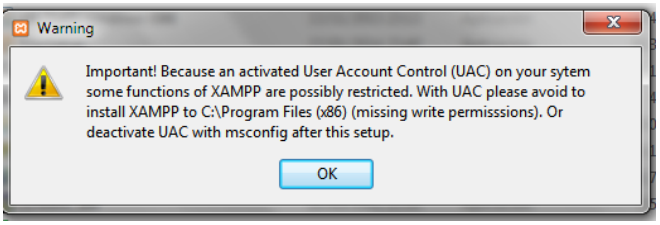


Figura 42: Instalador de XAMPP: Aviso por Cuentas de Usuario.

3. Después se inicia el asistente de instalación, y se siguen los pasos siguientes:
    a. Se hace clic en el botón *Next*.
    b. Se seleccionan los componentes que se deseen instalar.

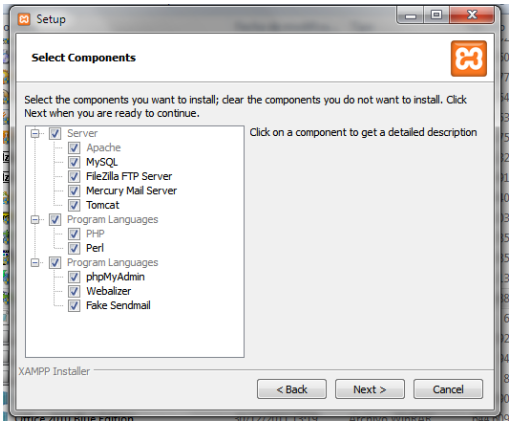


Figura 43: Instalador de XAMPP: Selector de Componentes para instalar.

    c. Se elige la carpeta donde se instalará XAMPP.

d. Se desmarcará la casilla para obtener información sobre los instaladores de aplicaciones para XAMPP creados por Bitnami.
e. Finalmente se instala XAMPP, se hace clic en *Next* y se inicia el proceso de instalación de archivos.
f. Se muestra la pantalla que confirma el fin de instalación de XAMPP, se pulsa *Finish* y se desmarca la casilla de inicio del panel de control de XAMPP.

Y después su instalación, se procede a instalar el siguiente archivo submit_order.php en el servidor local. Para ello únicamente se copiará el archivo submit_order.php, en la carpeta htdocs contenida en el directorio donde fue instalado XAMPP:

submit_order.php

```
<?php
//-- POST estas variables desde details.js
$names    = $_POST['names'];
$address1 = $_POST['address1'];
$address2 = $_POST['address2'];
$crust    = $_POST['crust'];

//-- limpia el array JavaScript
$toppings = str_replace('"','',substr(substr(stripslashes($_POST['toppings']),1),0,-1));
$toppings = explode(",\n", $toppings);

//-- Donde el pedido será enviado
$to = $_POST['email'];
$subject = "Pedido de pizza";
$message = "Un nuevo pedido ha sido enviado.<br/>";
$message. = $names."<br/>";
$message. = $address1 . "<br/>";
$message .= $address2 . "<br/><br/>";
$message .= "Pizza ". $crust . "con :<br/>";

$message .= "<ul>";
if (strlen($toppings[0]) == 1)
{
   $message .= "<li>Básica (Pizza de Queso)</li>";
}
else
{
   for ($i = 0; $i < count($toppings); $i++)
   {
      $message .= "<li>" . $toppings[$i] . "</li>";
   }
}
$message .= "</ul>";

//-- Las cabeceras permiten enviar código HTML como un email
$headers = "MIME-Version: 1.0\r\n";
$headers .= "Content-Type: text/html; charset=UTF-8\r\n";
$headers .= "From: Servicio Pizzeria <noreply@thepizzaplace.com>\r\n";


//-- Si el mensaje fue enviado, return true, si no return false. Esto se maneja en el método onload de details.js
if (mail($to,$subject,$message,$headers))
{
   $response = array('mail' => true);
}
else
{
   $response = array('mail' => false);
}
```

```
echo json_encode($response);
?>
```

Este servidor local atenderá las peticiones de pizza, y enviará un mensaje de confirmación a la cuenta de email indicada en submit_order.php. Pero para que el servidor local sea capaz de enviar emails, este deberá ser configurado siguiendo los pasos siguientes:

1. Se edita php.ini, en el servidor local, cambiando las dos líneas siguientes contenidas en la sección [mail function]:

```
; XAMPP IMPORTANT NOTE (1): If XAMPP is installed in a base directory with spaces (e.g. c:\program filesC:\xampp) fakemail and mailtodisk do not work correctly.
; XAMPP IMPORTANT NOTE (2): In this case please copy the sendmail folder in your root folder (e.g. C:\sendmail) and use this for sendmail_path.
; XAMPP: Comment out this if you want to work with fakemail for forwarding to your mailbox (sendmail.exe in the sendmail folder)
sendmail_path = "\"C:\xampp\sendmail\sendmail.exe\" -t"

; XAMPP: Comment out this if you want to work with mailToDisk, It writes all mails in the C:\xampp\mailoutput folder
;sendmail_path="C:\xampp\mailtodisk\mailtodisk.exe"
```

   Lo que hacen estas líneas es determinar qué servicio smtp será utilizado para enviar el email, es decir, que se utilizará el servicio sendmail, en lugar de mailtodisk.
2. Se edita el archivo sendmail, contenido en el directorio sendmail de XAMPP, para introducir la cuenta de correo electrónico desde la que se enviarán los emails.
   Para ello se seguirán los pasos siguientes:
   a. Se introduce el servidor smtp, de la cuenta utilizada, en la variable smtp_server.
   b. Se incluye el puerto utilizado para enviar el mail, en la variable smtp_port.
   c. Se introducen el nombre y el password de la cuenta email que se ha utilizado.

```
smtp_server=smtp.gmail.com

smtp_port=587

auth_username=mi_gmail@gmail.com
auth_password=password
```

Y finalmente, se abre el panel de XAMPP, y se pone en marcha el servidor Apache.

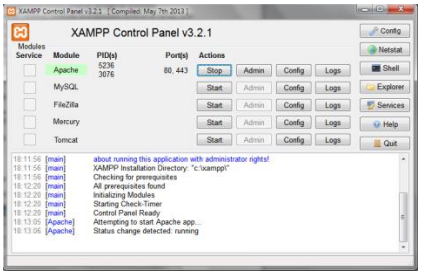


Figura 44: Panel de Control de XAMPP.

NOTA: Cuando se desee cerrar el servidor, se pulsará Stop y después se dará a Quit.

7.2. Desarrollo de la aplicación con Titanium Studio.

7.2.1. Descripción y características.

Titanium Studio es un Entorno de Desarrollo Integrado (IDE), que permite a los usuarios desarrollar y diseñar sus proyectos. Además, posee extensiones que permiten correr las aplicaciones en un simulador o en un dispositivo, y enviar las aplicaciones a Apple´s AppStore y Android Marketplace.

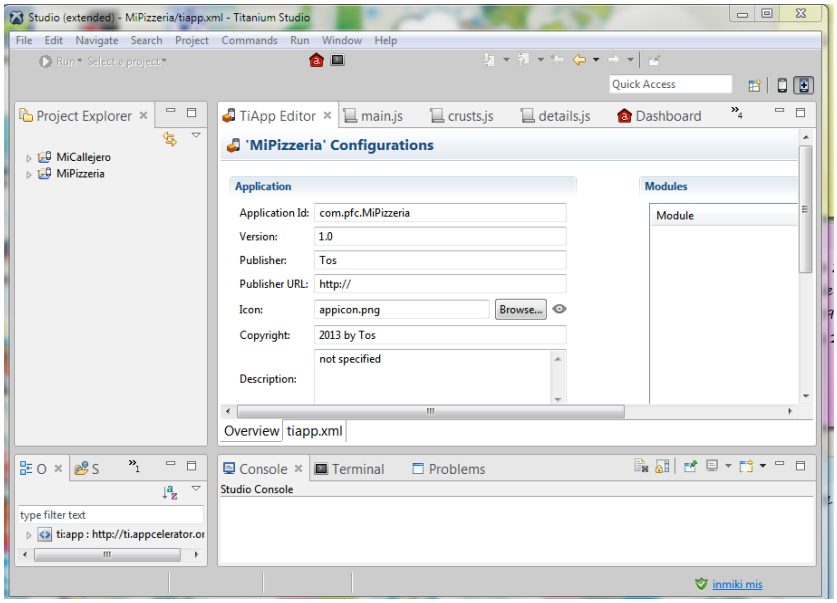


Figura 12: Ventana Principal de Titanium Studio.

Entre sus características más importantes se encuentran:

- Perspectivas, editores y vistas: Titanium Studio permite configurar el entorno de trabajo configurando las ventanas y editores.
- Gestión de Proyectos: Este IDE proporcionará asistentes para la creación de proyectos, además de la posibilidad de navegar entre sus directorios y archivos.
- Colección de plugins: Se podrán acceder a numerosos plugins o módulos.
- Revisor Automático de Sintaxis: El editor muestra alertas de errores sintácticos, y permite al programador corregirlos mientras se escribe la aplicación.

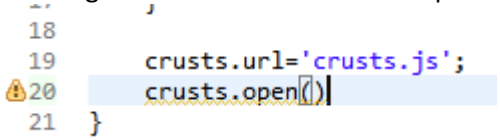


Figura 13: Revisor de Sintaxis.

- Autocompletado: Titanium Studio contiene un API con numerosos namespaces, y una lista de registros con las variables empleadas. Esto permite que mientras que se escribe el código aparezca la opción de elegir automáticamente los namespaces o

variables que necesiten ser utilizadas en cada momento.

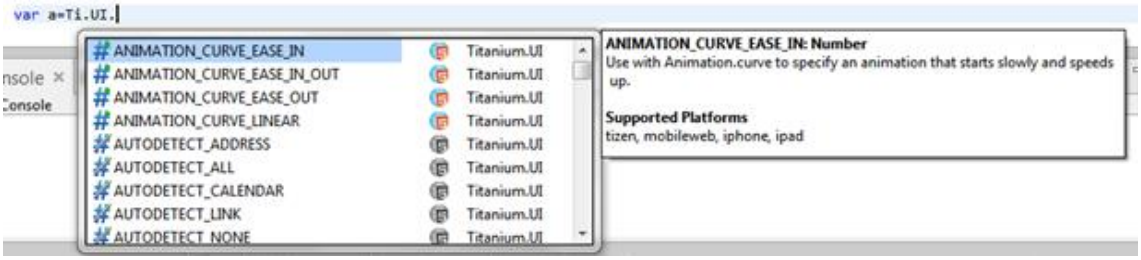


Figura 14: Autocompletado en Titanium.

- Depurador de código: Permite visualizar información de depuración, los breakpoints introducidos o el valor de las variables.
- Acceso a Dashboard: Desde donde se puede obtener información sobre los SDK instalados y las aplicaciones creadas por el usuario. Además de tener acceso a documentación para aprender a desarrollar aplicaciones y poder descargar código de aplicaciones ejemplo. Y también permite acceder acceso al Market Place de Titanium Studio.

7.2.2. Instalación y Configuración.

Para poder instalar Titanium se deberán cumplir una serie de prerrequisitos:

- Sistema Operativo: Tener instalado una versión reciente de Windows, Mac OS X o Ubuntu.
    - Apple Mac OS X: Deberá tener instalada la versión 10.8.4 o posterior.
    - Windows: Deberá tener instalado Windows 7 o Windows 8.
    - Ubuntu: Deberá tener instalada la versión Ubuntu 12.04 LTS.
- Memoria : Tener al menos 4 GB de memoria RAM, ya que se necesita 1 GB para Titanium Studio, y entre 1 y 2 GB para cada uno de los SDK.
- Java Runtime: Tener instalado Oracle JDK, la versión 6 a posterior.
- Node.js: Tener instalado node.js. Consiste en una plataforma con el motor JavaScript de Google que permite construir fácilmente aplicaciones rápidas de red escalables. Node.js utiliza un modelo de entrada/salida por eventos sin bloqueo que lo hace ligero y eficiente, ideal para aplicaciones en tiempo real de datos intensivos que se ejecutan a través de dispositivos distribuidos. Es por lo tanto, un entorno de programación en la capa del servidor basado en JavaScript.
- Git: Tener instalado Git, que es un sistema de control de versiones distribuido cuyo objetivo es el de permitir mantener eficientemente una gran cantidad de código a una gran cantidad de programadores.

Estos prerrequisitos se cumplirán, ya que en el caso de estudio se instaló Titanium Studio en un ordenador con Sistema Operativo Windows 7 y 4GB de memoria RAM.

Para instalar Titanium Studio se siguen los pasos siguientes:

1. Se descarga la última versión de JDK Oracle desde la página web http://www.oracle.com/technetwork/java/javase/downloads/index.html, y se instala.

2. Se descarga la última versión de node.js en http://nodejs.org/download/, y se instala. En caso de no realizar esta acción, se instalará automáticamente cuando se actualice Titanium Studio por primera vez.

3. Se descarga la última versión de Git en http://git-scm.com/downloads/, y se instala. Si no se realiza esta acción, el instalador de Titanium Studio lo descargara e instalara automáticamente.

4. Y por último se descarga e instala Titanium Studio. Para poder realizar la descarga el usuario en primer lugar deberá registrarse en https://my.appcelerator.com/auth/signup.
Y cuando reciba en su correo la confirmación de registro, deberá seguir el link recibido en su cuenta de correo, entrar en su cuenta Titanium y descargar la versión Titanium Studio deseada.

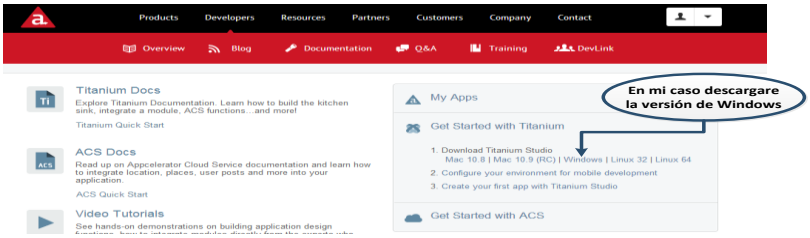


Figura 15: Acceso a Cuenta en Titanium.

Una vez descargado e instalado Titanium Studio, se deberán seguir los pasos siguientes para su correcta configuración:

1. Se selecciona el Workspace, o espacio de trabajo en que se desarrollarán las aplicaciones:

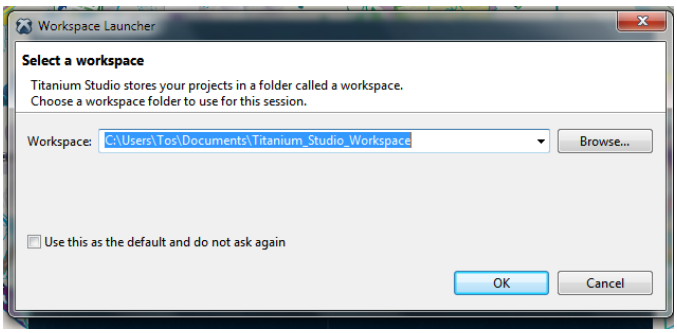


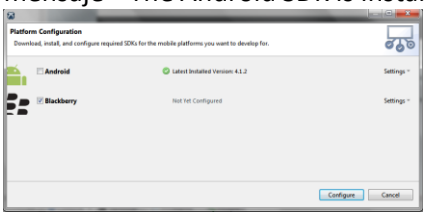
Figura 16: Ventana de Selección de Espacio de Trabajo.

2. El usuario introduce sus credenciales, es decir, los datos de su cuenta Titanium :

Figura 17: Ventana para introducir datos de la cuenta Titanium.

3. Y por último se instalan los SDKs correspondientes, en este caso al tratarse de la versión Windows de Titanium Studio no se podrá instalar el SDK correspondiente para iOS, ya que para poder instalarlo se necesitara un ordenador Apple Mac OS.

Por ejemplo, para instalar el SDK de Android hay que ir a Dashboard, instalar SDK de Android y cuando su instalación se haya completado, regresar a Dashboard, actualizar Android SDK para que se configure correctamente y esperar a que salga el mensaje " The Android SDK is installed successfully" :



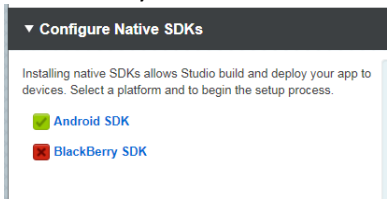
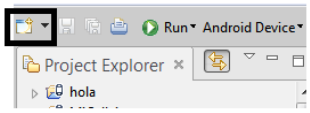


Figura 18: Ventanas de Instalación y estado del SDK de Android.

7.2.3. Creación de la App.

Para crear el proyecto:

1. En primer lugar se crea el fichero para el nuevo proyecto, existen dos formas distintas de hacerlo:

- La primera es eligiendo: File->New->Mobile App Project.
- La segunda es, pulsando el simbolo 

, y seleccionando Mobile Project.

2. Ahora aparecerá la siguiente ventana, en la que se configurarán las características del proyecto que se desea realizar.

Aquí se elige la opción Classic -> Single Window Application ->Next, para crear una aplicación desde un lienzo en blanco.

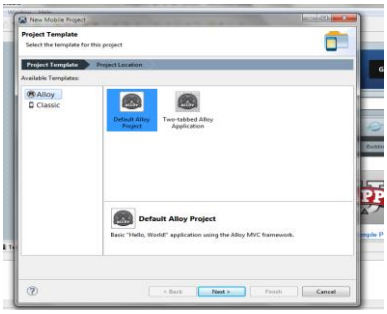


Figura 19: Ventana de Configuración del Proyecto Titanium.

3. Se procede a rellenar y seleccionar algunos de los campos siguientes:

- **Project Name:** Se introduce un nombre único para el proyecto. En este caso Mi Pizzeria.
- **Use Default Location:** Mantener esta opción elegida permitirá a Titanium Studio generar y guardar los ficheros del proyecto en su localización por defecto.

- **App Id:** Esta información se utiliza para subir el app a App Store o Android MarketPlace. Su formato es***: com.nombrecompañia.nombreproyecto***. En este caso se introducirá com.pfc.MiPizzeria.
- **Company/Personal URL:** Este es un campo opcional, y sirve para asociar una URL a la app.
- **Titanium SDK Version:** Este es un campo muy importante, sirve para seleccionar la versión de Titanium SDK utilizada. A medida de que se añadan nuevas características a Titanium API, la API cambiará. Dependiendo de estos cambios, las apps necesitarán rescribirse y recompilarse.
- **Development Targets:** Sirve para seleccionar las plataformas para las que se desarrolla la aplicación. Sus opciones son: Android, Mobile Web y BlackBerry. Para el caso de estudio se elegirá Android.
- **Cloud Settings:** Esto otorga servicios en nube para el desarrollo de la app. En este caso no se elegirá esta opción.

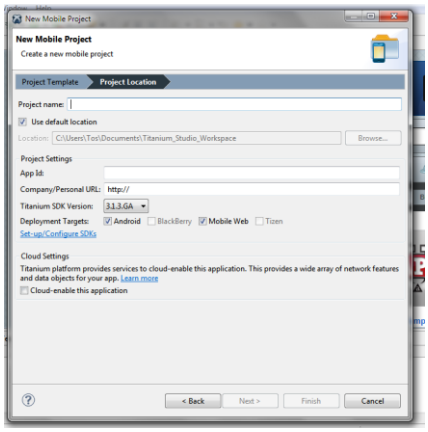


Figura 20: Ventana de Introducción de Datos del Proyecto Titanium.

4. Y finalmente, se pulsa *Finish*.

Nota: Si se elige TiApp Editor se podrán observar la configuración introducida, para observarla con más detalle se puede observar su fichero tiapp.xml, el cual será descrito en el apartado siguiente.

7.2.4. Estructura de la App.

Los ficheros incluidos en el proyecto son los siguientes:

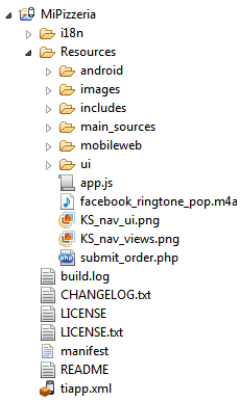


Figura 21: Ficheros del Proyecto Titanium.

- manifest: Es la Manifest File, el cual es leído por Titanium durante el tiempo de ejecución y determina los módulos y su versión que serán cargados para la aplicación. Es decir, determina que módulos serán empaquetados con la aplicación.

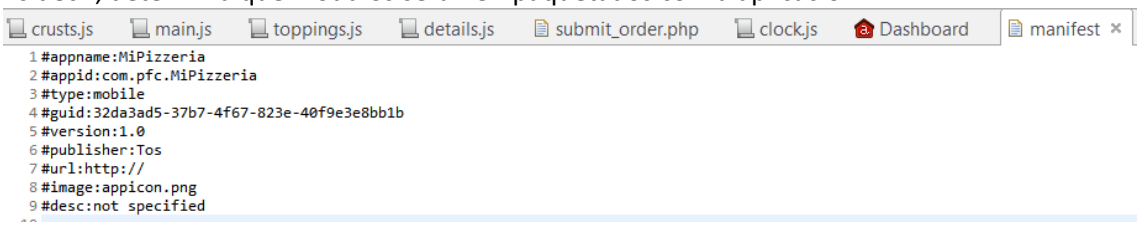


Figura 22: Manifest de Proyecto Titanium.

- tiapp.xml: Es el archivo Titanium Application que determina los detalles introducidos en la creación del nuevo proyecto que serán utilizados para introducir los datos de los archivos tiapp.xml y manifest.
- Resources: Es el directorio que contiene los archivos de código y los gráficos de la aplicación.
    - images: Contiene las imágenes utilizadas por la app.
    - includes: Contiene el archivo JavaScript clock.js utilizado para obtener la hora actual y mostrarla en cada una de las ventanas que componen la aplicación.
    - main_sources: Contiene los archivos main.js, crust.js, toppings.js y details.js, utilizados para configurar y mostrar las ventanas que componen la app.
    - app.js: Es el archivo principal que es cargado cuando la app es ejecutada.
    - Facebook_ringtone_pop.m4a: Es al archivo de audio que se reproduce cada vez que se selecciona un ingrediente.

7.2.5. Características de Titanium implementadas en la app.

7.2.5.1. Estructura del UI y sus Eventos: El UI de la aplicación, estará formado por una serie de ventanas por las que se navegará, y las que se hará referencia de la forma siguiente:

1. Se crea la ventana principal en app.js.

app.js
```
var main = Ti.UI.createWindow({
            url:'main_sources/main.js',
            height:Ti.Platform.displayCaps.platformHeight,
             width:Ti.Platform.displayCaps.platformWidth,
             fullscreen:true,
             navBarHidden:true
});
main.open();
```

2. Desde main.js, se crean las ventanas correspondientes, a las masas, los ingredientes y los detalles:

main.js
```
var win = Ti.UI.currentWindow;
 //
 var crusts = Ti.UI.createWindow();
 var toppings = Ti.UI.createWindow();
 var details = Ti.UI.createWindow();
 //
```

Y después se crean una serie de eventos globales, accesibles desde todas ventanas, y que se utilizará para navegar por las ventanas y para pasar datos a través de ellos.

Un ejemplo de esto es el caso Cancelar Ingredientes, en el que se siguen los pasos siguientes:

1. En main.js se crea el evento global mediante el módulo *Ti.App*, al que se le pasarán su función y evento correspondientes.

main.js

```
//Añadir función para abrir ventana masas
 function openCrust(e){
     toppings.close();
     if(e.crust){
         crusts.crust=e.crust;
      }
     crusts.url='crusts.js';
     crusts.open();
 }
 Ti.App.addEventListener('cancelToppings',openCrust);
```

2. Mientras que en toppings.js, se creará un botón cancelar. Y se asignará a este el evento click, que cuando sea escuchado ejecutará el evento cancelToppings al que se le pasará un objeto con la masa guardada.

toppings.js

```
 //Añadir Botón Cancelar Ingredientes Introducidos
 var cancel = Ti.UI.createButton({
                 width:137,
                 height:75,
                 backgroundImage:'../images/cancel.png',
                 top:385,
                 left:10,
                 opacity:0
 });
 win.add(cancel);
cancel.addEventListener('click',function(e){
                 win.removeAllChildren();
                 Ti.App.fireEvent('cancelToppings',{crust:win.crust});
 });
```

Nota: Cada vez que se cierre una ventana se deberán quitar cada sus elementos mediante el método removeAllChildren (), ya que si no se producirán problemas para visualizar los elementos al volver a abrir las ventanas.

7.2.5.2. Componente de UI: Lista tipo Carrusel.

Esta lista se creará para elegir las masas, mediante una lista de elementos tipo deslizable, en crusts.js. En la que cada vez que se desee elegir la masa siguiente, se deberá seleccionar y deslizar horizontalmente la masa anterior.

Se crearan vistas para cada una de las masas, mediante el método *Ti.UI.createView ()*, y un array con un conjunto de elementos que contengan el nombre de la masa y la ruta a la imagen masa.

crusts.js

```
//Añadir ventanas
var win=Ti.UI.currentWindow;
//
win.backgroundImage = '../images/bg_main.png';
//Vistas de las masas
var hechaMano=Ti.UI.createView({
             width:216,
```

```
        height:156,
        backgroundImage:'../images/crust/hand.png'
});
var natural=Ti.UI.createView({
        width:216,
        height:156,
        backgroundImage:'../images/crust/natural.png'
});
//Resto de vistas
var returnCrust;
//Referencia a masas
var crusts=[
        {title:'Hecha a Mano',path:'../images/crust/hand.png'},
        {title:'Natural',path:'../images/crust/natural.png'},
        {title:'De Pan',path:'../images/crust/pan.png'},
        {title:'Rellena',path:'../images/crust/stuffedCrust.png'},
        {title:'Fina y Crujiente',path:'../images/crust/thinNcrispy.png'}
];
```

Después se creará la lista deslizable mediante el método *Ti.UI.createScrollableView ()*, al que se le pasará en la propiedad *views* todas las vistas de las masas.

crusts.js

```
//Crear una scroll view
var scrollView=Ti.UI.createScrollableView({
        views:[hechaMano,natural,MasaPan,MasaRellena,FinaCrujiente],
        showPagingControl:true,
        clipViews:false,
        top:180,
        left:30,
        right:30,
        height:180,
        opacity: 0
});
//Es la función utilizada asignar los elementos del array a las vistas
if(win.crust){
        for(i=0;i<crusts.length;i++){
                if(win.crust == crusts[i].title){
                        returnCrust=i;
                        break;
                }
        }
        scrollView.scrollToView(returnCrust);
}
```

Más adelante se creará la etiqueta que mostrará el nombre de la masa elegida.

crusts.js

```
//Añadir el tipo de masa elegida
var crustType=Ti.UI.createLabel({
        text:'Hecha a Mano',
        font:{
                fontFamily:'Verdana',
                fontWeight:'bold',
                fontSize:16
        },
        color:'#fff',
        shadowColor:'#333',
        shadowOffset:{x:1,y:1},
        textAlign:'center',
        width:Ti.Platform.displayCaps.platformWidth,
```

```
        height:20,
        top:170,
        opacity:0
});
if(returnCrust!=null){
        crustType.text=crusts[returnCrust].title;
}
```

Se procederá a asignar el evento scroll (deslizar) al scrollView, el cual cada vez que sea ejecutado cambiará el nombre de la masa seleccionada.

crusts.js
```
scrollView.addEventListener('scroll',function(){
        crustType.text=crusts[scrollView.currentPage].title;
});
```

Se añaden todos los componentes a la ventana.

crusts.js
```
//Añadir componentes a la ventana
win.add(scrollView);
win.add(crustTitleView);
win.add(crustType);
win.add(next);
```

Y finalmente, se pasa la ruta de la imagen y el nombre de la masa elegida asignándole al botón next un evento click, que ejecutará el evento global toppings creado en main.js.

main.js
```
function openToppings(e){
        if(e.toppings){
                details.close();
        }
        else{
                crusts.close();
        }
        toppings.url = 'toppings.js';
        toppings.crust = e.crust;
        toppings.path = e.path;
        toppings.returnToppings = e.toppings;
        toppings.open();
}
//Evento Global
Ti.App.addEventListener('toppings',openToppings);
```

crusts.js
```
//Crear botón para seguir con los ingredientes de la pizza
var next=Ti.UI.createButton({
        width:137,
        height:75,
        backgroundImage:'../images/toppings_next.png',
        top:385,
        opacity:0
});
//
if(Ti.Platform.osname == 'android'){
        next.image='../images/toppings_next';
}
//Añadir función al botón
next.addEventListener('click',function(e){
```

```
        win.removeAllChildren();

        Ti.App.fireEvent('toppings',{
                crust:crusts[scrollView.currentPage].title,
                path:crusts[scrollView.currentPage].path
        });
});
```

Figura 23: Lista tipo Carrusel

Y después de crear la lista se pasara la variable path al evento global toppings, esto hará que en la ventana toppings.js, se utilice la vista de la pizza para mostrar la masa que se seleccionó en la ventana anterior.

toppings.js
```
//Añade imagen de la pizza
var pizza=Ti.UI.createView({
        width:216,
        height:156,
        top:270,
        backgroundImage:win.path
});
```

7.2.5.3. Componente de UI: Lista Deslizable.

Esta lista deslizable se utilizará para seleccionar los ingredientes que se desee que contenga la pizza.

Se crea el objeto de la lista deslizable mediante el método *Ti.UI.createScrollView ()*, y se definen dos variables, una con el número máximo de ingredientes y otra con el número de ingredientes seleccionados. Y un array que contenga la ruta, nombre y un contenedor para los ingredientes elegidos.

toppings.js
```
//Añadir ventanas
var win=Ti.UI.currentWindow;
//
win.backgroundImage = '../images/bg_main.png';
//Añadir ScrollView y núm. máximo y mínimo de ingredientes
```

```
var scrollView=Ti.UI.createScrollView();
var maxIngredientes=6;
var numToppings=0;
//Toppings
var toppings=[
		{title:'Bacon',path:'../images/toppings/bacon_bits.png',container:null},
		{title:'Ternera',path:'../images/toppings/beef.png',container:null},
		{title:'Pollo a la plancha',path:'../images/toppings/grilled_chicken.png',container:null},
		{title:'Jamon',path:'../images/toppings/ham.png',container:null},
		{title:'Salami (Desmenuzado) ',path:'../images/toppings/italian_sausage_crumbled.png',container:null},
		{title:'Salami (Cortado)',path:'../images/toppings/italian_sausage_sliced.png',container:null},
		{title:'Jalapeños',path:'../images/toppings/jalapenos.png',container:null},
		{title:'Champiñones',path:'../images/toppings/mushrooms.png',container:null},
		{title:'Olivas Negras',path:'../images/toppings/olives_black.png',container:null},
		{title:'Olivas Verdes',path:'../images/toppings/olives_green.png',container:null},
		{title:'Cebolla Roja',path:'../images/toppings/onions_red.png',container:null},
		{title:'Cebolla Blanca',path:'../images/toppings/onions_white.png',container:null},
		{title:'Pepperoni',path:'../images/toppings/pepperoni.png',container:null},
		{title:'Chiles Banana',path:'../images/toppings/peppers_banana.png',container:null},
		{title:'Chiles Verdes',path:'../images/toppings/peppers_green.png',container:null},
		{title:'Chiles Rojos',path:'../images/toppings/peppers_red.png',container:null},
		{title:'Piña',path:'../images/toppings/pineapple.png',container:null},
		{title:'Cerdo',path:'../images/toppings/pork.png',container:null},
		{title:'Tomate Troceado',path:'../images/toppings/tomatoes_diced.png',container:null},
		{title:'Tomates Marinados',path:'../images/toppings/tomatoes_marinated.png',container:null},
		{title:'Tomates Pera',path:'../images/toppings/tomatoes_roma.png',container:null},
];
```

Después se crea una vista la masa elegida y otra para mostrar los ingredientes seleccionados en la lista.

toppings.js

```
//Añade imagen de la pizza
var pizza=Ti.UI.createView({
		width:216,
		height:156,
		top:270,
		backgroundImage:win.path
});
//Añadir la vista de los ingredientes
var toppingsHolder=Ti.UI.createView({
		width:216,
		height:156
});
//Añadir los ingredientes a la pizza(La vista)
pizza.add(toppingsHolder);
```

Ahora se define la función *createToppingsList ()*, para crear la lista, y en la que se definirán etiquetas con los nombres de los ingredientes, y unas vistas que muestren si se ha marcado o desmarcado un ingrediente. Para ello, cada vez que se seleccione un elemento, si este ha sido seleccionado con anterioridad, se desmarca el elemento, se decrementa el número de ingredientes seleccionados y se elimina el contenedor que fue asignado al ingrediente. O en el caso contrario, se marca el elemento de la lista, se muestra la imagen del ingrediente, se incrementa el número de ingredientes seleccionados y se crea un contenedor para el ingrediente.

Sin embargo, antes de permitir que un ingrediente sea seleccionado, se debe tener en cuenta que el número máximo de ingredientes para una pizza son 6, y actuar en consecuencia.

toppings.js

```
//Funciones
function toppingListClick(e){
	if (e.source.selected){
		e.source.selected = false;
		e.source.backgroundImage = '../images/checkbox_no.png';
		numToppings -= 1;
		if (toppings[e.source.toppingID].container != null){
			toppingsHolder.remove(toppings[e.source.toppingID].container);
			toppings[e.source.toppingID].container = null;
		}
	}
	else{
		if (numToppings<maxIngredientes){
			e.source.selected = true;
			e.source.backgroundImage = '../images/checkbox_yes.png';
			var aTopping = Ti.UI.createView({
				backgroundImage:toppings[e.source.toppingID].path
			});
			if (Ti.Platform.osname == 'android'){
				aTopping.image = toppings[e.source.toppingID].path;
			}
			else{
				aTopping.opacity = 0;
				aTopping.animate({
					opacity:1,
					duration:500
				});
			}
			toppingsHolder.add(aTopping);
			toppings[e.source.toppingID].container = aTopping;
			numToppings += 1;
		 }
		else{
			alert ("Has Elegido demasiados ingredientes: " + numToppings + " son el
máximo.");
		}
	}
}
function createToppingsList(){
	scrollView.opacity=0;
	scrollView.top=155;
	scrollView.height=120;
	scrollView.contentWidth=Ti.Platform.displayCaps.platformWidth;
	scrollView.contentHeight='auto';
	scrollView.showVerticalScrollIndicator=true;
	win.add(scrollView);
	for (var i = 0; i < toppings.length; i++){
		var toppingLabel = Ti.UI.createLabel({
			text:toppings[i].title,
			font:{
				fontFamily:'Verdana',
				fontWeight:'bold',
				fontSize:14
			},
			color:'#fff',
			shadowColor:'#333',
			shadowOffset:{x:1,y:1},
			textAlign:'left',
			width:Ti.Platform.displayCaps.platformWidth - 10,
```

```
                left:10
        });
        //Se añade la propiedad checked a la vista de los componentes de la lista
        var checkbox = Ti.UI.createView({
                width:340,
                height:16,
                backgroundImage:'../images/checkbox_no.png',
                selected:false,
                toppingID:i
        });
        if(win.returnToppings){
                for(var j=0;j<win.returnToppings;j++){
                        var aTopping=Ti.UI.createView({
                                backgroundImage:toppings[i].path
                        });
                        if(Ti.Platform.osname=='android'){
                                aTopping.image=toppings[i].path;
                        }
                        else{
                                aTopping.opacity=0;
                                aTopping.animate({
                                        opacity:1,
                                        duration:500
                                });
                        }
                        toppingsHolder.add(aTopping);
                        toppings[i].container=aTopping;
                        checkbox.backgroundImage='../images/checkbox_yes.png';
                        checkbox.selected=true;
                        numToppings+=1;
                }
        }
        var toggler = Ti.UI.createView({
                width:Ti.Platform.displayCaps.platformWidth,
                height:20,
                top: i * 20
        });
        checkbox.addEventListener('singletap',toppingListClick);
        toggler.add(toppingLabel);
        toggler.add(checkbox);
        scrollView.add(toggler);
    }
    scrollView.animate({
        opacity:1,
        duration:500
    });
}
createToppingsList();
```

Y finalmente se añaden los elementos creados a la ventana.

toppings.js

```
win.add(pizza);
win.add(toppingsTitleView);
win.add(details);
win.add(cancel);
```

Figura 24: Lista tipo Deslizable.

Y después de definir la lista se le pasa a la ventana details.js el nombre y la ruta de la imagen de la masa elegida, y la lista de los ingredientes seleccionados. Utilizando para ello el botón details, al cual se le asignará el evento click, que ejecutará el evento global details, creado en main.js.

main.js

```
function openDetails(e){
        toppings.close();
        details.crust = e.crust;
        details.path = e.path;
        details.toppings = e.toppings;
        details.url = 'details.js';
        details.open();
}
Ti.App.addEventListener('details',openDetails);
```

toppings.js

```
details.addEventListener('click',function(e){
        win.removeAllChildren();

        var pizzaInfo = [];
        for (var i = 0; i < toppings.length; i++){
                if (toppings[i].container != null){
                        pizzaInfo.push(toppings[i].title);
                }
        }
        Ti.App.fireEvent('details',{crust:win.crust,path:win.path,toppings:pizzaInfo});

});
```

7.2.5.4. Componente de UI: Etiqueta con Hora actual.

Para poder mostrar la hora actual en todas y cada una de las ventanas que componen la aplicación, se deberá incluir el archivo clock.js a las ventanas crusts.js, toppings.js, y details.js.

Para ello se utiliza el método *Ti.include ()*:

```
Ti.include ('../includes/clock.js');
```

Nota: Aunque se utiliza el método Ti.include (), es recomendable utilizar CommonJS el cuál es un estándar que permite utilizar código JavaScript para crear módulos con librerías y componentes propios. Ya que será eliminado en un futuro próximo.

Mientras que en clock.js, se utiliza la función *new Date ()*, utilizada para crear una variable de tiempo, que servirá obtener mediante los métodos *getHours ()*, y *getMinutes ()*, la hora y minutos actuales.

clock.js

```
var hora = Ti.UI.createLabel({
        font:{
                fontFamily:'Verdana',
                fontWeight:'bold',
                fontSize:14
        },
        color:'#fff',
        shadowColor:'#333',
        shadowOffset:{x:1,y:1},
        textAlign:'right',
        width:Ti.Platform.displayCaps.platformWidth,
        height:20,
        top:85,
        left:-13
});
function getHora(){
        var d = new Date();
        var HoraAct = d.getHours();
        var MinutoAct = d.getMinutes();
        MinutoAct = MinutoAct + '';
        if (MinutoAct.length == 1)
        {
                MinutoAct = '0' + MinutoAct;
        }
        hora.text = HoraAct + ':' + MinutoAct + ' ';
}

getHora();
win.add(hora);
```

Figura 25: Etiqueta con Hora Actual.

7.2.5.5. Componente de UI: Entrada de Texto (Text Field).

Para poder crear un Text Input se utilizará el método *Ti.UI.createTextField (),* que contendrá las siguientes propiedades:

- hintText: Consiste en el texto que será mostrado cuando el Text Field está vacío, y desaparecerá cuando el usuario introduzca el texto necesario. Esta propiedad se utiliza sobre todo para decirle al usuario que información debe introducir.
- keyboardType: Sirve para establecer el tipo de teclado que utilizará el usuario cuando introduzca el texto. Entre los diferentes valores que puede contener esta propiedad se encuentran:
    - Titanium.UI.KEYBOARD_ASCII: Si se desea utilizar un teclado ASCII.
    - Titanium.UI.KEYBOARD_DECIMAL_PAD: Si se desea utilizar un teclado que sólo contenga números.
    - Titanium.UI.KEYBOARD_DEFAULT: Si se desea emplear el teclado utilizado por defecto en el dispositivo.
- returnKeyType: Especifica el texto que se mostrará en la tecla Retorno del teclado cuando este campo este enfocado. Entre los diferentes valores que podrá contener esta propiedad se encuentran:
    - RETURNKEY_DEFAULT: Usa la tecla de Retorno utilizada por defecto del teclado virtual.
    - RETURNKEY_DONE: Establece “Done” como el texto de la tecla Retorno.
    - RETURNKEY_EMERGENCY_CALL: Establece “Emergency Call” como el texto de la tecla retorno
    - RETURNKEY_NEXT: Establece “Next” como el texto de la tecla retorno.
    - RETURNKEY_SEND: Establece “Send” como el texto de la tecla retorno.
- suppressReturn: Determina si la tecla retorno debe ser suprimida durante la entrada de texto.

Estas entradas de texto serán implementadas en la ventana details.js, para que el usuario introduzca sus datos personales necesarios para autorizar la petición, de la forma siguiente:

details.js

```
var names = Titanium.UI.createTextField({
   color:'#336699',
   top:100,
   left:10,
   width:300,
   height:40,
   hintText:'Nombre',
   backgroundImage:'../images/textfield.png',
   paddingLeft:8,
   paddingRight:8,
   keyboardType:Titanium.UI.KEYBOARD_DEFAULT,
   returnKeyType:Titanium.UI.RETURNKEY_NEXT,
   suppressReturn:false
});
var address1= Titanium.UI.createTextField({
   color:'#336699',
   top:140,
   left:10,
   width:300,
   height:40,
```

```
    hintText:'Su dirección',
    font:{
            fontFamily:'Verdana',
            fontSize:14
    },
    backgroundImage:'../images/textfield.png',
    keyboardType:Titanium.UI.KEYBOARD_DEFAULT,
    returnKeyType:Titanium.UI.RETURNKEY_NEXT,
    suppressReturn:false
});
var address2 = Titanium.UI.createTextField({
    color:'#336699',
    top:180,
    left:10,
    width:300,
    height:40,
    hintText:'Población.Ciudad,Código Postal',
    backgroundImage:'../images/textfield.png',
    paddingLeft:8,
    paddingRight:8,
    keyboardType:Titanium.UI.KEYBOARD_DEFAULT,
    returnKeyType:Titanium.UI.RETURNKEY_DEFAULT
});
var email= Titanium.UI.createTextField({
    color:'#336699',
    top:220,
    left:10,
    width:300,
    height:40,
    hintText:'Su email',
    font:{
            fontFamily:'Verdana',
            fontSize:14
    },
    backgroundImage:'../images/textfield.png',
    keyboardType:Titanium.UI.KEYBOARD_DEFAULT,
    returnKeyType:Titanium.UI.RETURNKEY_NEXT,
});
```

Y después se establecerá un escuchador de eventos que se ejecutará cada vez que se pulse la tecla retorno cuándo se esté introduciendo texto. Lo que hará que se pase a enfocar al siguiente Text Field.

details.js

```
names.addEventListener('return',function(){
        address1.focus();
});

address1.addEventListener('return',function(){
        address2.focus();
});

address2.addEventListener('return',function(){
        email.focus();
});
```

Y finalmente, se añadirán estos componentes a la ventana:

details.js

```
win.add(names);
win.add(address1);
```

```
win.add(address2);
win.add(email);
```

Figura 26: Entrada de Texto (TextField).

7.2.5.6. Componente de UI: Etiqueta con el Pedido Realizado.

Antes de poder completar el pedido, se mostrará información sobre la masa y los ingredientes elegidos. Para que el usuario pueda comprobar que su pedido es el correcto.

Para ello se implementará la función *getPizza (),* que obtendrá los valores de las variables globales crust y toppings, utilizadas para guardar la masa e ingredientes seleccionados.

details.js

```
//Mostrar el Pedido Realizado
function getPizza(){
	var text='Pizza '+win.crust+' con:\n';
	if(win.toppings.length == 0){
		text+='* Básica (Pizza de Queso)\n';
	}
	else{
		for(var i=0;i<win.toppings.length;i++){
			text+='* '+win.toppings[i]+'\n';
		}
	}
	return text;
}
//
var pizzaInfoText=Ti.UI.createLabel({
	text:getPizza(),
	font:{
		fontFamily:'Verdana',
		fontSize:14
	},
	color:'#fff',
	shadowColor:'#333',
	shadowOffset:{x:1,y:1},
	textAlign:'left',
	with:Ti.Platform.displayCaps.platformWidth,
	height:160,
	top:220,
	left:10
```

```
});
win.add(pizzaInfoText);
```

Figura 27: Etiqueta con el Pedido Realizado

7.2.5.7. Obtener Información sobre los dispositivos.

Será necesario obtener el nombre del Sistema Operativo para asegurarse de que al asignar una imagen a un elemento esta se asigne con éxito. Ya que la propiedad backgroundImage en ocasiones no funciona correctamente en dispositivos Android.

Para obtener esta información se utilizará el módulo *Titanium.Platform*, y una de las propiedades siguientes:

- platform: Es el nombre de la plataforma del dispositivo. Puede ser:
  - android.
  - iPhone OS.
  - TIZEN.
  - Mobile Web.
- osname: Es el nombre corto del sistema operativo del dispositivo. Puede ser:
  - android.
  - iPhone.
  - ipad.
  - tizen.
  - mobileweb.

En el caso utilizado, se utilizó la propiedad osname.

toppings.js

```
//Añade imagen de la pizza
var pizza=Ti.UI.createView({
        width:216,
        height:156,
        top:270,
        backgroundImage:win.path
});
//Añadir la vista de los ingredientes
```

```
var toppingsHolder=Ti.UI.createView({
		width:216,
		height:156
});
//Añadir los ingredientes a la pizza(La vista)
pizza.add(toppingsHolder);
//Añadir botón para introducir los detalles(siguientes ventana)
var details=Ti.UI.createButton({
		width:137,
		height:75,
		backgroundImage:'../images/details.png',
		top:385,
		right:10,
		opacity: 0
});
//Añadir Botón Cancelar Ingredientes Introducidos
var cancel = Ti.UI.createButton({
		width:137,
		height:75,
		backgroundImage:'../images/cancel.png',
		top:385,
		left:10,
		opacity:0
});
//
if (Ti.Platform.osname == 'android'){
		details.image = '../images/details.png';
		cancel.image = '../images/cancel.png';
		pizza.image = win.path;
}
else{
		pizza.opacity = 0;
}
```

7.2.5.8. Notificaciones.

Existen 3 tipos de notificaciones:

- Alertas.
- Diálogos de confirmación.
- Avisos.

Para crear una alerta se utilizará el método global *alert ()*. Un ejemplo de su se encuentra en la función *toppingListClick (e)*, que se muestra una alerta cuando se intente elegir más ingredientes de los permitidos.

toppings.js

```
function toppingListClick(e){
  if (e.source.selected){
    //Si el ingrediente ya ha sido elegido
  }
  else{

    if (numToppings<maxIngredientes){
      //Se elige un Nuevo ingrediente, y no se ha sobrepasado el
      //número máximo de ingredientes permitidos
    }
    else{
      alert ("Has Elegido demasiados ingredientes: " + numToppings + " son el máximo.");
    }
```

```
    }
}
```

Figura 28: Alerta.

Mientras que para crear un diálogo de confirmación, que es igual a una alerta salvo en el hecho de que puede contener uno o más botones, se utiliza el método *Titanium.UI.AlertDialog()* que contiene las propiedades siguientes:

- buttonNames: Es un string con el nombre de cada uno de los botones creados. Hay que destacar que en Android no habrá definido ningún botón por defecto, y serán soportados un máximo de tres botones. Mientras que en iOS, existirá por defecto un botón OK.
- cancel: Es un número, el cual corresponde al índice del botón cancelar. Por ejemplo, si se tuviesen tres botones: OK, Cancel, y Back, con los índices 0 1 y 2 respectivamente, el valor de la propiedad cancel sería 1.
- message: Es el mensaje del diálogo.
- ok: Es el texto del botón OK.
- title: Es un string con el título del diálogo. Si esta propiedad no es definida, aparecerá un diálogo sin barra de título.

Para mostrar un diálogo de confirmación se utilizará el método *show ().*

En el caso de estudio se creará un diálogo de confirmación cuando el pedido de la pizza se haya realizado con éxito. Que cuando sea cerrado, ejecutará el evento global resetApp de main.js, que cerrará la ventana details.js, y abrirá de nuevo la ventana de selección de masa crusts.js.

details.js

```
Var alertDialog=Titanium.UI.createAlertDialog({
                    title:'Éxito',
                    message: 'Tu pedido ha sido enviado',
                buttonNames:[‘OK’]
            });
            alertDialog.show();
            alertDialog.addEventListener('click',function(e){
                    win.removeAllChildren();
```

```
                    Ti.App.fireEvent('resetApp');
                });
});
```

main.js

```
function resetApp(){
   details.close();
   openCrust({});
}
Ti.App.addEventListener('resetApp',resetApp);
```

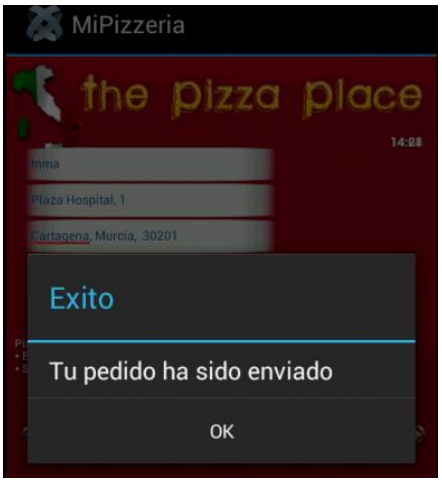


Figura 29: Diálogo de Confirmación.

7.2.5.9. Animaciones.

Las animaciones pueden añadir efectos visuales a los elementos que componen las apps. Las animaciones se clasifican en:

- **Animaciones básicas:** Permiten modificar las propiedades de los elementos, como su posición, su color o su opacidad. Para ello se emplea el método *animate ()* que permite modificar las propiedades de un elemento, mediante un objeto tipo *Titanium.create.Animation ().*
- **Animaciones matriciales:** Una transformación matricial es aquella capaz de cambiar las coordenadas, tamaño y posición o rotar un elemento. Y que puede realizar la transformación en 2 y 3 dimensiones.
    - Animaciones matriciales 2D: Estas animaciones se realizan en espacios de 2 dimensiones. Para implementar una transformación, se necesitará en primer lugar crear una instancia de un objeto *Ti.UI.create2DMatrix* e indicar en el elemento de la animación, en su propiedad *transform,* de que se trata de una animación matricial 2D insertándole el objeto *Ti.UI.createMatrix2D*.
    - Animaciones matriciales 3D: Este tipo de animaciones son solo soportadas por iOS. Y son capaces de rotar, mover, y cambiar el tamaño de los elementos, de la misma forma que las animaciones 2D con la diferencia que este caso las transformaciones se realizan en 3 dimensiones.
- **Transiciones:** Las transiciones son animaciones sólo soportadas por iOS. Se utilizan cuando se produce el cambio de una ventana a otra, o entre vistas. Para ello se define

la propiedad *transition* en el objeto animado, y se utiliza el módulo *Ti.UI.iPhone.AnimationStyle.typeoftransition* para especificar el tipo de transición utilizado.

Un ejemplo de animación básica se encuentra en todos los elementos que componen las ventanas, ya que a todos y cada uno de ellos se les asigna una opacidad con valor igual a 0, por lo que al cargar las ventanas por primera vez sus elementos son transparentes, y transcurrido un cierto tiempo, estos se vuelven totalmente opacos.

crusts.js

```
//Titulo
var crustTitle=Ti.UI.createLabel({
        //Resto propiedades
});
//Fondo del titulo
var crustTitleView=Ti.UI.createView({
        //Resto propiedades
        opacity: 0
});
//Añadir en el fondo el titulo
crustTitleView.add(crustTitle);
//Añadir el tipo de masa elegida
var crustType=Ti.UI.createLabel({
        //resto de propiedades
        opacity:0
});
if(returnCrust!=null){
        crustType.text=crusts[returnCrust].title;
}
//
//Crear botón para seguir con los ingredientes de la pizza
var next=Ti.UI.createButton({
        //resto propiedades
        opacity:0
});
//
crustTitleView.animate({
        opacity:1,
        duration:500
});
crustType.animate({
        opacity:1,
        duration:500
});
next.animate({
        opacity:1,
        duration:500
});
```

Figura 30: Animación de Componentes.

7.2.5.10. Trabajar con Datos Remotos.

Se explicará cómo las apps pueden interactuar con servidores remotos, y como podrán comunicarse o subir/bajar contenidos de estos. Para poder interactuar con estos se utilizará:

- Protocolo HTTP: Se suele utilizar para comunicarse con un servidor remoto o descargar y subir archivos a este. Y puede trabajar mediante:
  - Datos JSON.
  - Datos XML.
- Servicios web SOAP.

1. Protocolo HTTP.

-Clientes HTTP: Las app pueden interactuar con servidores remotos a través del protocolo HTTP empleando el objeto *Ti.Network.HTTPClient*. Para trabajar con servidores remotos se declara un objeto *HTTPClient* al que se le pasan una serie de parámetros. Como pueden ser las funciones callback asociadas al ciclo de vida de la peticiones mediante los parámetros onload y onerror, o el tiempo en milisegundos que transcurrirá hasta que la petición sea abortada, mediante el parámetro timeout.

Además se seguirán tres pasos para realizar una petición HTTP:

1. Se crea un objeto HTTPClient, mediante el método *createHTTPClient ()*.
2. Se abre una conexión aplicándole al objeto HTTPClient el método *open ()*, al que se le pasa como primer parámetro el tipo de petición que se desea realizar (GET o POST), y como segundo parámetro la url del servidor remoto, al que se realiza la petición.
3. Se envía la petición HTTP, aplicándole al objeto HTTPClient el método *send ()* con la información enviada como parámetros.

Existen dos tipos de peticiones HTTP, GET o POST.

- Peticiones GET: La mayoría de las ocasiones el envío de peticiones GET no es útil para la app, sino que la utilidad se encuentra en la respuesta del servidor a esa petición. Con el objetivo de acceder a esos datos, se definen una serie de funciones callback que son ejecutadas en puntos específicos en el ciclo de vida de la petición. Como *onload*

que es llamada como confirmación de que la petición ha sido recibida correctamente, y *oneerror* que es llamada cuando hay un error. En esas funciones callback las respuestas son:

- this.responseText: Posee la información útil devuelta en texto sin formato.
- this.responseXML: Posee la información útil como una instancia a un documento XML.
- this.responseData: Posee la información útil como un BLOB (Dato Binario).

- Peticiones POST: En ocasiones se necesitara enviar datos al servicio web a través de peticiones POST o PUT.

details.js

```
var orderReq=Titanium.Network.createHTTPClient();
order.addEventListener('click',function(){
	if(names.value==''||address1.value==''||address2.value==''||email.value==''){
		alert('Rellene todos los apartados');
	}
	else{
		names.enabled=false;
		address1.enabled=false;
		address2.enabled=false;
		email.enabled=false;
		order.enabled=false;
		cancel.enabled=false;
		//orderReq.open('POST','http://10.0.2.2:80/submit_order.php');
		orderReq.open('POST','http://IPServidor:80/submit_order.php');
		var params={
			names:names.value,
			address1:address1.value,
			address2:address2.value,
			crust:win.crust,
			toppings:JSON.stringify(win.toppings),
			email:email.value
		};
		orderReq.send(params);

	}
});
```

Nota: Cuando se desee conectarse con el servidor ejecutado desde el servidor local, si la aplicación se ejecuta desde un dispositivo Android, la aplicación deberá conectarse a la dirección IP de la máquina donde se encuentra el servidor. Esto sólo sirve cuando ambas máquinas están conectadas a la misma red WIFI.

Mientras que si la aplicación se ejecuta en el emulador, para acceder a este servidor local, la aplicación deberá conectarse a la dirección IP 10.0.2.2, que es por defecto la dirección IP que se le asigna al localhost en el emulador.

Figura 31: El usuario no ha introducido todos sus datos.

-Ciclo de vida XHR de las peticiones: Sirve para conocer el estado en la comunicación con un servicio web. Entre las diferentes funciones callback que pueden ser llamadas para obtener el ciclo de vida de una petición se encuentran:

- onload: Es llamada cuando se recibe la respuesta HTTP de la petición. Junto con los datos listos para ser utilizados.
- onerror: Es llamada cuando ocurre un error en la petición HTTP. Como por ejemplo, que no se tenga acceso a la red.

2. Trabajar con datos JSON.

El mejor formato para transporte de datos usado por JavaScript es JSON (JavaScript Object Notation). Los datos JSON tienen la ventaja de que pueden ser serializados fácilmente dentro y fuera de objetos JavaScript. Por lo que necesitan menos tiempo para transferirse.

Titanium integra el soporte para la serializar JSON en el namespace *JSON*. Las dos funciones proporcionadas por este son:

- JSON.stringify (): Toma un objeto JavaScript y lo convierte en un string.
- JSON.parse (): Toma un string en formato JSON y lo convierte en un objeto JavaScript.

Si se tiene un servicio web con respuesta en formato JSON, se puede serializar la respuesta dentro de la función *onload*. Los datos devueltos por el servicio web pueden almacenarse como una propiedad del objeto *HTTPClient.*

- Recibimiento y verificación de datos JSON: Se recuperan los datos JSON utilizando el objeto *HTTPClient*. Dentro de la función callback *onload*, *this.responseText* contiene la respuesta de la dirección URL. Esta es la propiedad que se utilizará para procesar datos JSON.
- Enviar datos JSON: El método *send ()* convierte un string automáticamente a datos JSON.

details.js

```
orderReq.onload=function(){
        var json=this.responseText;
        var response=JSON.parse(json);
```

```
		//
		if(response.mail==true){
			var alertDialog=Titanium.UI.createAlertDialog({
				title:'Éxito',
				message: 'Tu pedido ha sido enviado',
			buttonNames:['OK']
			});
			alertDialog.show();
			alertDialog.addEventListener('click',function(e){
				win.removeAllChildren();
				Ti.App.fireEvent('resetApp');
			});
		}
		else{
			alert('PHP ha fallado al enviar pedido al correo electrónico');
			names.enabled=true;
			address1.enabled=true;
			address2.enabled=true;
			email.enabled:true;
			order.enabled=true;
			cancel.enabled=true;
		}
};
//Error de red
orderReq.onerror=function(event){
		alert('Error de red: '+JSON.stringify(event));
		names.enabled=true;
		address1.enabled=true;
		address2.enabled=true;
		email.enabled:true;
		order.enabled=true;
		cancel.enabled=true;};
```

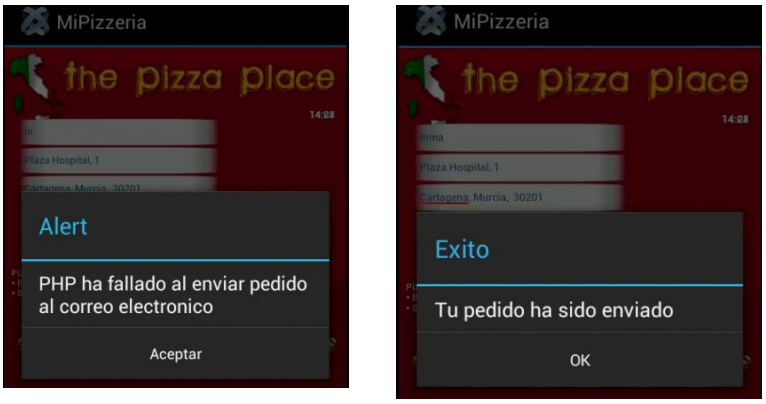


Figura 32: Las respuestas del servidor al pedido.

7.2.5.11. Multimedia.

Se emplearán los Audio APIs para reproducir un sonido cada vez que se elige un ingrediente.

Para ello se utilizará el módulo *Ti.Media*, que contiene las APIs para reproducir o grabar sonidos. En este caso se utilizará el API *Ti.Media.Sound* que se utiliza para reproducir sonidos. Pero hay que tener en cuenta que cuando se reproduce un sonido junto con el objeto *Ti.Media.Sound*, el archivo de sonido es cargado en la memoria antes de ser reproducido, y que por lo tanto el uso de memoria se incrementa.

Para crear un objeto *Sound* se utiliza el método *Ti.Media.createSound ()*, con él que se podrán utilizar los métodos siguientes:

- play (): Reproduce un sonido.
- pause (): Pone en pausa un sonido.
- stop (): Para la reproducción de un sonido.
- setVolume (): Controla el volumen de un sonido. Pero se podría utilizar la propiedad volume en su lugar.

Y las propiedades:

- url: Acepta el directorio local en el que se encuentra el archivo del sonido, o la URL a un archivo de sonido remoto.
- volume: Establece el volumen del sonido.
- preload: Se utiliza para establecer si se carga el sonido antes de su reproducción (se le da el valor true), o no (se le da el valor false).

toppings.js

```
function toppingListClick(e){
        var player = Ti.Media.createSound ({url:"../facebook_ringtone_pop.m4a"});
        player.play();
     if (e.source.selected){
         //Se quitan ingredientes
     }
     else{
         //Se  incluyen ingredientes
    }
}
function createToppingsList(){
      //resto de código
      checkbox.addEventListener('singletap',toppingListClick);
}
```

7.2.6. Simulación de la App.

Para poder realizar la simulación de la aplicación en un emulador se configurara el emulador siguiendo los pasos siguientes:

1. Se abre la ventana Run Configurations, de una de las dos formas siguientes:
- En la barra principal de Titanium, se selecciona *Run* y se elige la opción Run Configurations.
- Se va al explorador de proyectos, se pulsa el botón derecho sobre la carpeta de la app y se selecciona Run As->Run Configurations.

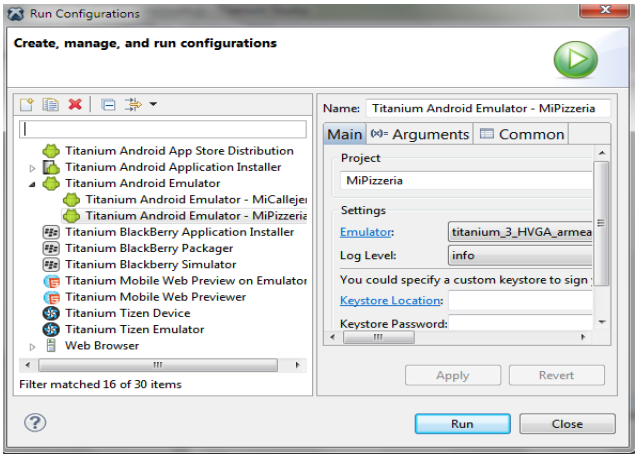


Figura 33: Ventana Run Configurations de Titanium Studio.

2. Se selecciona el enlace Emulator y se abre la ventana Preferences (Filtered), que permite ver las diferentes configuraciones de cada una de las plataformas instaladas por Titanium. En este caso de la plataforma Android.

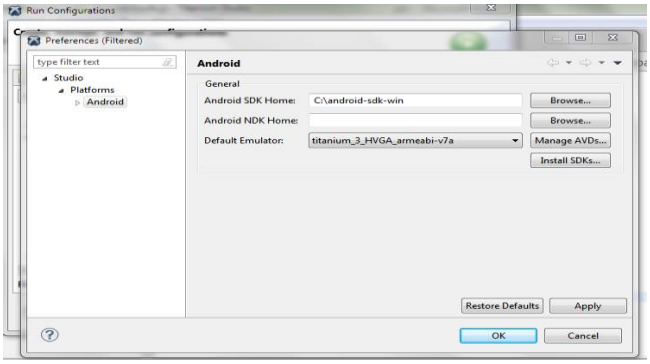


Figura 34: Ventana Preferences Filtered (Preferencias) de Simulación en Android.

3. Aquí se pulsa *Manage AVDs* y se abre la ventana de configuración de dispositivos virtuales de Android (Android Virtual Device Manager).
4. Se pulsa el botón *New*, para crear el nuevo emulador o dispositivo virtual, y se le introduce su configuración, es decir, su nombre, el dispositivo emulado o la versión de Android que utilizará dicho dispositivo.
5. Y se pulsa *OK*, para confirmar la creación del dispositivo virtual.
6. Se regresa a la ventana Preferences (Filtered), y se selecciona en el apartado Default Emulator el dispositivo virtual que se ha creado. Se da a *Apply* para aplicar el dispositivo creado y se pulsa *OK*, para confirmar la configuración.

Se ejecuta el emulador, de una de las dos formas siguientes:

- Iniciándolo en la ventana Run Configurations, al pulsar *Run*.
- Se va al explorador de archivos, se hace clic con el botón derecho sobre la carpeta que contiene el proyecto y se elige Run As->Android Emulator.

Nota: Puede ser que la primera vez que se ejecute el dispositivo virtual, tardé demasiado en instalar la aplicación, si es así, es recomendable volver a hacer clic con el botón derecho sobre la carpeta de la

aplicación y pulsar Run As->Android Emulator. Y observar que en Console, aparezca que la aplicación está empezando a compilarse e instalarse.

7.2.7. Instalación en Dispositivo.

Para poder instalar la aplicación en un dispositivo HTC Desire se seguirán los pasos siguientes:

1. Se baja el driver de HTC en http://www.htc.com/us/support/.
2. En el dispositivo se marca la casilla que permite instalar aplicaciones de orígenes desconocidos, y se habilita la depuración Android.
3. Se conecta el dispositivo Android al ordenador mediante un cable USB.
4. Se instala la aplicación desde Titanium Studio:
    a. Se va a Run->Run Configurations, y se elige el dispositivo en el que se desea instalar la aplicación.

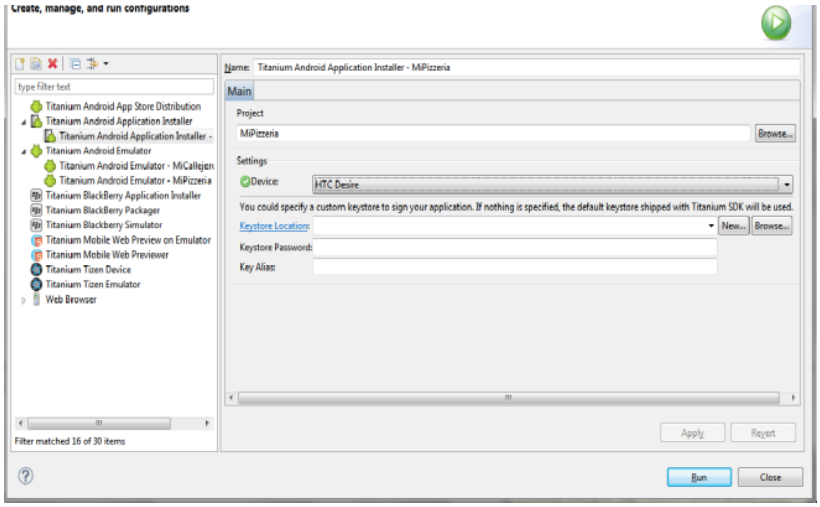


Figura 35: Ventana Run Configurations para Dispositivos Android en Titanium Studio.

    b. Se pulsa *Run.*

7.3. Desarrollo de la aplicación con PhoneGap y JQuery Mobile.

7.3.1. Descripción y Características del entorno de desarrollo: Eclipse.

Eclipse es un IDE (Entorno de Desarrollo Integrado), el cuál puede ser extendido a través de plugins. Y que proporciona herramientas para desarrollar, ejecutar y depurar aplicaciones.

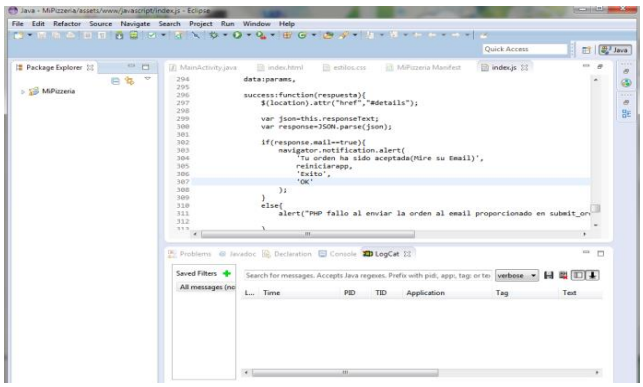


Figura 36: Ventana Principal de Eclipse.

Entre sus principales características se encuentran:

- Perspectivas, editores y vistas: Eclipse permite configurar el entorno de trabajo definiendo su aspecto configurando las ventanas y editores.
- Gestión de Proyectos: Este IDE proporcionará asistentes para la creación de proyectos, además de la posibilidad de navegar entre sus directorios y archivos.
- Depurador de código: Se incluye un depurador, el cual permitirá ejecutar la aplicación obteniendo paso a paso lo que pasa con el código, para así poder mejorarlo.
- Colección de plugins: Se podrán acceder a numerosos plugins o módulos, que permitirán acceder a componentes de software, capaces por ejemplo, de permitir programar en C/C++ y Python, o gestionar base de datos.
- Revisor Automático de Sintaxis: El editor muestra alertas de errores sintácticos, y permite al programador corregirlos mientras se escribe la aplicación.
- Autocompletado: Eclipse contiene un API y una lista de registros con las variables empleadas. Esto permite que mientras que se escribe el código aparezca la opción de autocompletado y se pueda seleccionar los métodos o variables que se desean emplear sin necesidad de tener que escribirlas.

7.3.2. Instalación de Eclipse.

Antes de poder instalar Eclipse deberán seguirse una serie de prerrequisitos:

- Sistema Operativo: Tener un ordenador con uno de los siguientes S.O instalados:
  - Windows 32-bit o 64-bit.
  - Mac OS X (Cocoa 32) o Mac OS X (Cocoa 64).
  - Linux 32-bit o Linux 64-bit.
- Java Runtime: Tener instalado Oracle JDK, versión 6 o posterior.

Y una vez se cumplan estas condiciones se seguirán los pasos siguientes para instalar Eclipse:

1. Se descarga la última versión de JDK Oracle desde la página web http://www.oracle.com/technetwork/java/javase/downloads/index.html, y se instala.
2. Se descarga Eclipse desde la página http://www.eclipse.org/downloads/, y se descomprime en el lugar deseado.

Después se descarga e instala el kit de desarrollo de Android (SDK), y el Plugin ADT, para lo que se deberán cumplir una serie de prerrequisitos:

- Sistema Operativo: Tener un ordenador con uno de los siguientes S.O instalados:
  - Windows XP, Vista o Windows 7.
  - Mac OS X versión 10.5.8 o posterior (sólo x86).
  - Ubuntu Linux versión 8.04 o posterior.
- Eclipse IDE: Eclipse Indigo (3.7.2) o posterior.
- JDK 6 o posterior.

Para instalar el kit de desarrollo de Android (SDK), se seguirán los pasos siguientes:

1. Se descarga la herramienta SDK para Windows en http://developer.android.com/sdk/index.html, y se elige la opción "USE AN EXISTING IDE".
2. Después se aceptan los términos de uso y comienza la descarga.
3. Se ejecuta el instalador descargado, y se abre automáticamente el administrador de SDK en el que se eligen los paquetes siguientes:
    - Los paquetes más recientes de Herramientas (Tools).
    - La versión más reciente de Android.
    - Y en Extras, la librería de soporte (Android Support Library) y el driver de USB (Google USB Driver).
4. Se inicia la instalación de todos los paquetes elegidos.

Para instalar el plugin ADT en Eclipse se seguirán los pasos siguientes:

1. Se abre Eclipse, y se agrega el repositorio de desarrolladores de Android. Para ello se hace clic en *Help* y se selecciona *Install New Software*, lo que abrirá la ventana "Available Software".
2. Se hace clic en *Add*, y en los campos Name y Location de la ventana "Add repository" se añade:
    - Name: ADT Plugin.
    - Location: http://dl-ssl.google.com/android/eclipse.

    Y se pulsa *OK*.
3. En la ventana "Install", se seleccionan todos los archivos para descargarlos, y se pasa a la ventana siguiente.
4. Se aceptan todas las licencias de las herramientas que se van a instalar, y se finaliza la instalación.

    Nota: Puede que parezca una advertencia diciendo que la validez del software no puede ser establecida, dicha advertencia deberá ser ignorada.
5. Se reinicia Eclipse para completar la instalación, y entonces aparecerá una ventana en dónde se debe introducir la ubicación de SDK Android, para ello en la ventana de bienvenida se elegirá "Use existing SDK", y en el campo SDK Location se introducirá el directorio en el que se instaló el SDK de Android, y después se aplica y confirma la configuración.

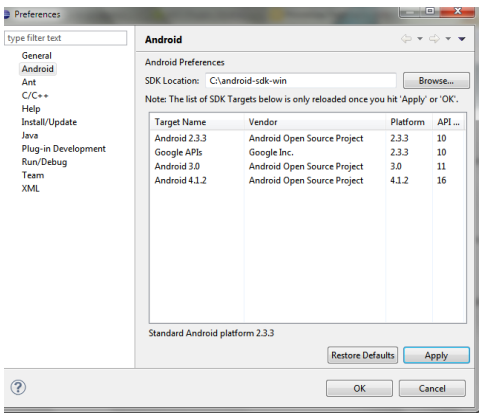


Figura 37: Ventanas para Introducir las Preferencias de Android.

7.3.3. Creación de la App.

Los pasos que se seguirán para crear un proyecto Android en Eclipse son:

1. Se crea un nuevo proyecto en Eclipse, y para ello se elige la opción File->New->Project, y en la ventana que se abre se elige la opción New Android Application.
2. Se rellenan o seleccionan los campos siguientes:

- Application Name: Se introduce el nombre de la aplicación, en este caso MiPizzeria.
- Project Name: Se introduce el nombre del proyecto creado, en este caso MiPizzeria.
- Package Name: Se introduce el identificador único de la app creada, sigue el formato com.company.application. En este caso se introduce com.pfc.mipizzeria.
- Minimum Require SDK: Se selecciona la versión mínima de Android permitida por SDK.
- Target SDK: Se selecciona el nivel API utilizado por SDK.
- Compile Width: Se selecciona la versión de API con la que la aplicación será compilada.
- Theme: Se elige el tema, o estilo, que se utilizará por defecto en la aplicación.
  Y se pulsa *Next.*

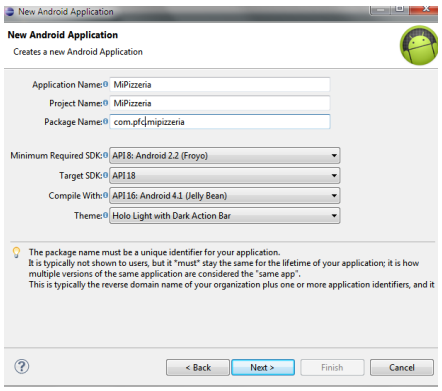


Figura 38: Ventana para introducir datos del Proyecto de Eclipse.

3. Se selecciona el icono que será utilizado por la aplicación, y se da a *Next*.
4. Se crea una Actividad para la aplicación, para ello se elige la opción Blank Activity y se pulsa *Next*.
5. Se configura la actividad creada, introduciendo su nombre y el tipo de navegación que utilizará.
6. Se finaliza la creación, y se pulsa *Finish*.

7.3.4. Instalación de PhoneGap.

Para realizar la instalación de PhoneGap se siguen los pasos siguientes:

1. Se descarga PhoneGap en http://phonegap.com/install/, y se elige la versión deseada.
   Nota: En el caso de estudio se descargó la versión 2.7.0, aunque también se podría descargar hasta la versión 2.9.0, para poder seguir los pasos siguientes.
2. Se crean los directorios */assets* y */libs* en el directorio raíz.
3. En el directorio *assets*, se crea un subdirectorio */www*.
4. En *assets/www* se crea el archivo index.html.

5. Se va al directorio donde se descomprimió PhoneGap, y se copian los siguientes archivos y directorios:
   - Se copia cordova-2.7.0.jar, que contiene la librería de PhoneGap en */libs*.
   - Se copia cordova-2.7.0.js en *assets/www*.
   - Se copia el directorio *xml* en el directorio *res* del proyecto.
6. Se procede a incluir las librerías de PhoneGap, y para ello se pulsa sobre el archivo .jar con el botón derecho, y se elige Build Path-> Configure Build Path. En la sección Libraries se añade el archivo al pulsar *Add JARs*, se elige el archivo cordova-2.7.0.jar y se da a *OK*.Y para asegurarse que la nueva librería se incluye correctamente se refresca el proyecto, pulsando Refresh.
7. En index.html se incluye el archivo cordova-2.7.0.js en la cabecera de la página.

index.html

```
<head>
<script type="text/javascript" charset="utf-8" src="cordova-2.7.0.js"></script>
<!--Resto Librerías-->
</head>
```

8. Se edita el archivo Java principal que se encuentra en el directorio src de la forma siguiente:
   - Se añade la librería import org.apache.cordova.*;
   - Se cambia la clase extendida de Activity a DroidGap.
   - Se establece el permiso de acceso a la función onCreate como public.
   - Se elimina el método onCreateOptionsMenu ().
   - Se utiliza el método setIntegerProperty (), para establecer la imagen que se mostrará mientras que se carga la aplicación.
   - Se elimina el método setContentView (), y se añade en su lugar super.loadUrl (file:///android_asset/www/index.html,10000).

```
1  package com.pfc.mipizzeria;
2
3  import android.os.Bundle;
7
8  public class MainActivity extends DroidGap {
9
10     @Override
11     public void onCreate(Bundle savedInstanceState) {
12         super.onCreate(savedInstanceState);
13         super.setIntegerProperty("splashscreen",R.drawable.icon);
14         super.loadUrl("file:///android_asset/www/index.html",10000);
15     }
16
17 }
18
```

Figura 39: MainActivity.java.

9. Se edita el manifiesto de Android, al hacer clic con el botón derecho y elegir Open Width->XML Editor.
   - Se incluyen los permisos de acceso a los APIs y de soporte de diferentes tamaños de pantalla, entre las etiquetas <uses_sdk…/> y <application…/>.

```
<supports-screens
android:largeScreens="true"
android:normalScreens="true"
android:smallScreens="true"
android:resizeable="true"
android:anyDensity="true" />
```

```
<uses-permission android:name="android.permission.VIBRATE" />
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_LOCATION_EXTRA_COMMANDS" />
<uses-permission android:name="android.permission.READ_PHONE_STATE" />
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.RECEIVE_SMS" />
<uses-permission android:name="android.permission.RECORD_AUDIO" />
<uses-permission android:name="android.permission.MODIFY_AUDIO_SETTINGS" />
<uses-permission android:name="android.permission.READ_CONTACTS" />
<uses-permission android:name="android.permission.WRITE_CONTACTS" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.GET_ACCOUNTS" />
<uses-permission android:name="android.permission.BROADCAST_STICKY" />
```

Figura 40: Campos <uses_sk> y <supports-screens> en AndroidManifest.xml.

- Y se añade el soporte de cambio de orientación dentro de la sección <activity…/> que se encuentra dentro de <application…/>.

```
<application
    android:allowBackup="true"
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme" >
    <activity
        android:name="com.pfc.mipizzeria.MainActivity"
        android:configChanges="orientation|keyboardHidden|screenSize"
        android:label="@string/app_name" >
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />

            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>
</application>
```

Figura 41: Modificación de la sección <activity> de AndroidManifest.xml.

7.3.5. Instalación de jQuery Mobile.

Para instalar jQuery Mobile, también se deberá descargar jQuery, y se seguirán los pasos siguientes:

1. En *assets/www* se crearán los subdirectorios */javascript* y */css*.
2. Se descarga jQuery Mobile en http://jquerymobile.com/ , y se elige la opción de descargar la última versión estable.
   Entonces se iniciará la descarga de un archivo .zip que se descomprimirá en el lugar que se desee, y se seleccionarán las librerías de JQuery Mobile que se deseen utilizar. En este caso:
   - jquery.mobile-1.4.2.min.css
   - jquery.mobile-1.4.2.min.js.
3. Se instala jQuery Mobile copiando el archivo jquery.mobile-1.4.2.min.js en la subcarpeta *assets/www/javascript*, y el archivo jquery.mobile-1.4.2.min.css en la subcarpeta *assets/www/css*.
4. Se incluyen jQuery Mobile en la cabecera de la página.

index.html

```
<head>
<link rel="stylesheet" href="css/jquery.mobile-1.4.2.min.css" />
<script type="text/javascript" src="javascript/jquery.mobile-1.4.2.min.js"></script>
</head>
```

5. Se descargará la versión de jQuery deseada, que por cuestiones de compatibilidad con jQuery Mobile deberá ser de la versión de 1.8 a 1.10, o la versión 2.0:

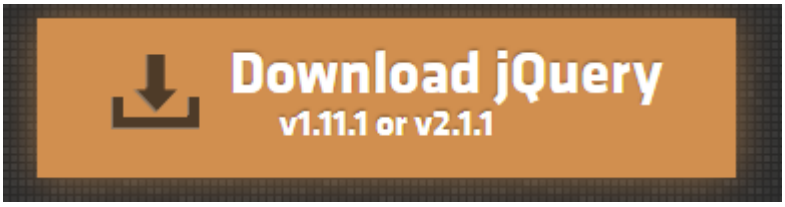


Figura 42: Enlace para descargar jQuery.

   - jQuery 1.11.1, en Download the compressed, production jQuery 1.11.1.
   - jQuery Migrate 1.2.1, el cual se obtendrá a través del enlace Download the compressed, production jQuery Migrate 1.2.1.
6. Se instala JQuery copiando los archivos jquery-1.11.min.js y jquery-migrate-1.2.1.min.js, en la subcarpeta *assets/www/javascript*.
7. Y para finalizar se incluye jQuery en la cabecera de la página.

index.html

```
<head>
<script type="text/javascript" src="javascript/jquery-1.11.1.min.js"></script>
<script type="text/javascript" src="javascript/jquery-migrate-1.2.1.min.js"></script>
</head>
```

7.3.6. Estructura de la App.

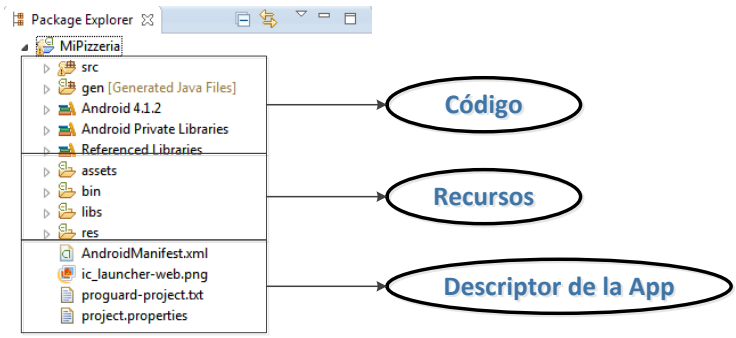


Figura 43: Estructura de la Aplicación en Eclipse.

Un proyecto creado para Android está compuesto por:

- Descriptor de la Aplicación.
- Código Fuente.
- Ficheros de Recursos.

7.3.6.1. Carpetas de Código.

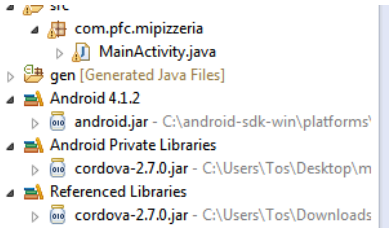


Figura 44: Carpetas de código de un proyecto Android.

Las carpetas de código son:

- src: Contiene el código fuente de la aplicación implementada en MainActivity.java.
- gen: Contiene el código generado de forma automática por el SDK, el cual nunca será modificado manualmente.
- Android 4.1.2: Es el código JAR, el API de Android según la versión seleccionada.
- Android Private Libraries y Referenced Libraries: Es el código JAR, el API de PhoneGap.

7.3.6.2. Carpetas de Recurso.

Las carpetas de recurso son:

- assets: Contiene los ficheros que podrán ser utilizados por la aplicación, en este caso todos los contenidos dentro de la subcarpeta /www. Que a la vez contendrá:

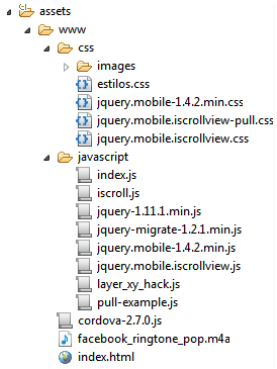


Figura 45: Carpeta assets.

  - /css: Es la subcarpeta donde se tendrán las hojas de estilo CSS, y las imágenes utilizadas por la app.
  - /javacript: Es la subcarpeta donde se tendrá el archivo index.js utilizado para manejar la aplicación, y los archivos JavaScript con el código fuente de las librerías JQuery, JQuery Mobile y iscrollview.
  - cordova-2.7.0.js: Es el archivo JavaScript con el código fuente de PhoneGap.
  - index.html: Es el documento HTML encargado de definir la estructura y componentes de las páginas que componen la app, su aspecto, y las referencias a los archivos JavaScript y hojas de estilo utilizadas.
  - facebook_ringtone_pop.m4a: Es el archivo de audio que se reproducirá cada vez que se elige una masa o se selecciona un ingrediente.
- res: Contiene los recursos utilizados por la aplicación y su interfaz de usuario.
  Está compuesto por las subcarpetas:

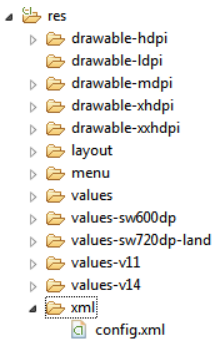


Figura 46: Carpeta res.

    - drawable: Contiene ficheros de imágenes y descriptores de imágenes. Aquí deberán incluirse los iconos y las imágenes de carga de aplicaciones de Cordova en las subcarpetas drawable.
    - layout: Contiene los ficheros XML con las vistas de la app.
    - menu: Contiene los ficheros XML con los menús de la aplicación.
    - values: Contiene los ficheros XML, que permiten cambiar los valores de las cadenas, los colores y los estilos, sin necesidad de modificar el código fuente.
    - xml: Contiene el archivo XML requerido por PhoneGap.
- libs: Contiene el fichero JAR externo fuente de PhoneGap.

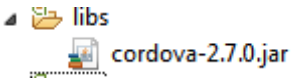


Figura 47: Carpeta libs.

7.3.6.3. Descriptores de la aplicación.

- AndroidManifest.xml: Es el fichero que describe la aplicación Android. En dónde se indica:
    - Actividades e Intentos.
    - Servicios y Proveedores de Contenido.
    - Permisos que requiere la aplicación.
    - Versión mínima de Android necesaria para poder ejecutar la app.
- project.properties: Es un fichero generado automáticamente por la SDK, que sirve para comprobar la versión del API cuando se instala la aplicación en el dispositivo.

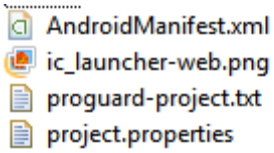


Figura 48: Descriptores de una App Android en Eclipse.

7.3.7. Características de PhoneGap implementadas en la App.

7.3.7.1. Eventos.

Los eventos son cualquier acción realizada en la aplicación que puede ser detectada por PhoneGap. Y que será detectada por el manejador de eventos *addEventListener (),* que se empleará de la forma siguiente:

```
element.addEventListener (event_type, function, useCapture)
```

Entre los diferentes eventos detectados por PhoneGap se encuentran:

- deviceready: Detecta cuando el dispositivo está listo. Este tiene que ser el primer evento en ser detectado, antes de cualquier otro, ya que una vez es ejecutado se puede llamar el API de PhoneGap.
- pause: Detecta cuando la app ha sido pausada. Se considera que una app ha sido pausada cuando deja de ejecutarse en primer plano, y pasa a ejecutarse en segundo plano, pero no cuando el usuario la ha cerrado.

- resume: Detecta cuando una app pausada pasa de ejecutarse de segundo a primer plano.
- menubutton: Detecta cuando el botón menú ha sido pulsado. Es soportado por dispositivos Android o BlackBerry.
- searchbutton: Detecta cuando el botón search ha sido pulsado. Es soportado por dispositivos Android.
- backbutton: Detecta cuando el botón back ha sido pulsado. Es soportado en dispositivos Android, BlackBerry o Windows Phone 7 y 8.
- startcallbutton: Detecta cuando se pulsa el botón de llamada. Es soportado por dispositivos BlackBerry.
- endcallbutton: Detecta cuando el botón de fin de llamada ha sido pulsado en dispositivos BlackBerry.
- volumedownbutton: Detecta cuando el usuario presiona el botón bajar volumen en el dispositivo. Es soportado por dispositivos BlackBerry.
- volumeupbutton: Detecta cuando el usuario presiona el botón subir volumen en dispositivos BlackBerry.
- online: Detecta cuando una app está conectada a Internet.
- offline: Detecta cuando una app no está conectada a Internet.
- batterycritical: Detecta cuando el nivel de batería ha llegado a un nivel crítico. Este evento pasa un objeto que contiene dos propiedades :
    - level: El porcentaje de nivel de batería (0-100).
    - isPlugged: Es un booleano que detecta si el dispositivo está conectado.

  Las app suelen utilizar *window.addEventListener* para detectar este tipo de evento.

  Es soportado por dispositivos iOS, Android, BlackBerry y Tizen.

- batterylow: Se ejecuta cuando la batería llega al umbral de bajo nivel. Este evento pasa un objeto que contiene dos propiedades :
    - level: El porcentaje de nivel de batería (0-100).
    - isPlugged: Es un booleano que detecta si el dispositivo está conectado.

  Es soportado por dispositivos iOS, Android, BlackBerry y Tizen.

- batterystatus: Se ejecuta cuando hay un cambio la batería en el nivel de la batería. Este evento pasa un objeto que contiene dos propiedades :
    - level: El porcentaje de nivel de batería (0-100).
    - isPlugged: Es un booleano que detecta si el dispositivo está conectado.

  Es soportado por dispositivos iOS, Android, BlackBerry, Windows Phone 7 y 8, y Tizen.

En el caso ejemplo presentado, para que la aplicación pueda trabajar con PhoneGap, se deberá esperar a que se ejecute el evento deviceready, y a partir de ese instante se ejecutarán cada una de las funciones y eventos implementados en la aplicación.

index.js

```
document.addEventListener("deviceready",onDeviceReady,false);

function onDeviceReady(){
```

```
        $(document).ready(function(){
                getHora();
                inicio();
        //se ejecutarán los eventos implementados por JQuery Mobile
});
}
function getHora(){
}
function inicio(){
}
```

7.3.7.2. Notificaciones.

Las notificaciones consisten en avisos visuales o sonoros que dan una información importante al usuario. Entre los diferentes tipos de notificaciones soportadas por PhoneGap se encuentran:

- Alertas: Se crearán utilizando el método *navigator.notificaton.alert ().*
- Diálogos de confirmación: Se crearán con el método *navigator.notification.dialog ().*
- Avisos: Se crearán con el método *navigator.notification.promtp ().*
- Notificaciones sonoras: Se crearán empleando el método *navigator.notification.beep ().*

Sin embargo, en el caso de estudio el utilizado el único tipo de notificaciones utilizadas son las alertas. Que serán creadas de dos formas distintas:

1. Mediante el método *alert ().*
2. Mediante el método *navigator.notification.alert ().*

Un ejemplo de uso del método *alert ()* es cuando el usuario no ha introducido los campos necesarios para realizar la petición.

index.js

```
document.addEventListener("deviceready",onDeviceReady,false);

function onDeviceReady(){
        $(document).ready(function(){
$("#pedido").on("tap",hacerPedido);
        });
}
function hacerPedido(){
        if($("#names").val()==" "||$("#names").val()=="Nombre"||$("#address1").val()==""||$("#address1").val()=="Dirección"||
          $("#address2").val()==" "||$("#address2").val()=="Población,Ciudad,Código Postal"||$("#email").val()==" "||
          $("#email").val()=="Su email"){

                alert("Rellene todos los campos");
        }
        else{
        // Se realiza la petición
        }
}
```

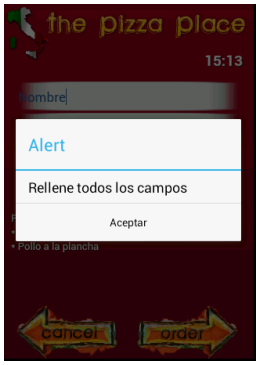


Figura 49: Alerta con alert ().

Mientras que un ejemplo de uso del método *navigator.notification.alert ()* es cuando al realizar la petición al servidor, este ha enviado con éxito el correo electrónico de confirmación.

A este método se le pasan los parámetros:

navigation.notification.alert (message, callback, [title], [button]);

- message: Es un string que contiene el mensaje con la información del aviso.
- callback: Es la función callback a la que se llama cuando la alerta es descartada.
- title: Este es un parámetro opcional, y consiste en el título de la alerta. Su valor por defecto es Confirm.
- button: Es un parámetro opcional, y consiste en el nombre que se le asignará al botón asociado a la alerta. Su valor por defecto es Ok.

index.js

```
if(response.mail==true){
        navigator.notification.alert(
                "Tu pedido ha sido enviado",
                reiniciarapp,
                "Éxito",
                "OK"
        );
}
else{
        alert("PHP ha fallado al enviar pedido al correo electrónico");

}
```

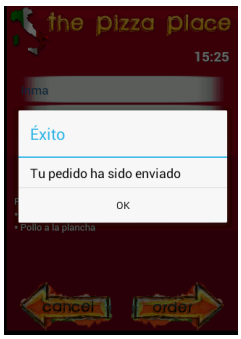


Figura 50: Alerta con navigator.notification.alert ().

7.3.7.3. Trabajar con Datos Locales.

Existen tres perspectivas diferentes que permiten trabajar con datos en PhoneGap:

- SQLite.
- FileSystem.
- LocalStorage.

Pero en esta aplicación lo que se necesita es una forma sencilla de almacenar un par de valores, así que se empleará un objeto *localStorage* que da acceso al interfaz local de almacenamiento en lugar de utilizar la base de datos o el sistema de archivos.

Para crear un objeto *localStorage* se invocará a *window.localStorage* que contendrá los métodos siguientes:

- key: Devuelve el nombre de la clave de la posición especificada.
- getItem: Devuelve el nombre identificado por su clave.
- setItem: Almacena el dato con la clave proporcionada.
- removeItem: Borra el dato identificado con la clave proporcionada.
- clear: Borra todas las parejas de valor/clave.

Este objeto se utilizará, por ejemplo, para guardar la masa seleccionada en la ventana crusts, y que esta pueda ser mostrada en la ventana toppings.

index.js
```
function irToppings(){
    //resto código
            var storage=window.localStorage;
            storage.setItem("gmasa",masas[i]);
            var masa=storage.getItem("gmasa");
            if(masa=="Hecha a Mano"){$("div.masag").addClass("hand");}
            if(masa=="Natural"){$("div.masag").addClass("natural");}
            if(masa=="De Pan"){$("div.masag").addClass("pan");}
            if(masa=="Rellena"){$("div.masag").addClass("stuffed");}
            if(masa=="Fina y Crujiente"){$("div.masag").addClass("thin");}
    //resto código
}
```

O cuando al volver a la ventana crusts, desde la ventana de selección de ingredientes toppings, se desee seleccionar de nuevo la masa, y por lo tanto, se deba borrar la masa que fue guardada anteriormente.

index.js
```
function volverCrusts(){
            var storage=window.localStorage;
            var masa=storage.getItem("gmasa");
            i=0;

            if(masa=="Hecha a Mano"){$("div.masas").removeClass("hand");}
            if(masa=="Natural"){$("div.masas").removeClass("natural");}
            if(masa=="De Pan"){$("div.masas").removeClass("pan");}
            if(masa=="Rellena"){$("div.masas").removeClass("stuffed");}
            if(masa=="Fina y Crujiente"){$("div.masas").removeClass("thin");}

            storage.removeItem("gmasa");
```

```
            $(".masa").html(masas[i]);
            $("div.masas").addClass("hand");
}
```

7.3.7.4. Multimedia.

El API *Media* permitirá reproducir y grabar audio en un dispositivo. Se puede reproducir archivos de audio almacenados en el dispositivo o almacenados en un servidor remoto.

Para poder reproducir audio se hará uso del objeto *Media* que permitirá guardar y reproducir audio en un dispositivo. Al que se le pasará los parámetros:

```
var media= new Media (src, mediaSuccess, [mediaError], [mediaSatus]);
```

- src: Es la URI que contiene el archivo de audio.
- mediaSuccesss: Es un parámetro opcional. Y es una función callback que se ejecuta después de que el objeto Media ha completado satisfactoriamente su reproducción, una acción stop, o se ha terminado de grabar el audio.
- mediaError: Es un parámetro opcional. Y consiste en una función callback que se ejecutará cuando ocurra un error. Esta función devolverá un objeto MediaError, con las propiedades siguientes:
    - code: Es el código del error que se ha producido. Sus valores pueden ser: MediaError.MEDIA_ERR_ABORTED, MediaError.MEDIA_ERR_NETWORK, MediaError.MEDIA_ERR_DECODE y MediaError.MEDIA_ERR_NONE_SUPORTED.
    - messsage: Es un mensaje de error que posee una descripción del error que se ha producido.
- mediaStatus: Es un parámetro opcional. Y consiste en una función callback que se ejecuta cuando se produce un cambio en el estado del objeto Media. Esta devolverá un objeto Media, con las propiedades siguientes: Media.MEDIA_NONE, Media.MEDIA_STARTING, Media.MEDIA_RUNNING, Media.MEDIA_PAUSED y Media.MEDIA_STOPPED.

Entre los diferentes métodos utilizados por el objeto *Media* para la reproducción de audio se encuentran:

- getCurrentPosition: Se utiliza para obtener la posición actual de un archivo de audio.
- getDuration: Devuelve la duración de un archivo de audio en segundos si es conocida, o el valor -1 si no se conoce la duración.
- play: Empieza o reanuda la reproducción de un archivo de audio.
- pause: Pausa la reproducción de un archivo de audio.
- stop: Para la reproducción de un archivo de audio.
- setVolume: Introduce el volumen de reproducción del archivo de audio. Y se le pasa el parámetro:

    ```
    media.setVolume (volume);
    ```

    - volume: Establece el volumen del reproductor de audio Sus valores irán de 0.0 a 1.0.

En el caso de estudio se reproducirá un sonido cada vez que se cambie la mase seleccionada, o cada vez que se seleccione o deseleccione un ingrediente de la lista.

index.js

```
document.addEventListener("deviceready",onDeviceReady,false);

function onDeviceReady(){
        $(document).ready(function(){
$("div.masas").on("tap",cambioMasa);
$("li").on("tap",seleccionIngredientes);
});
}
function cambioMasa(event){
        var media=new Media("/android_asset/www/facebook_ringtone_pop.m4a");
        media.play();
    //Código para cambiar masas
}
function seleccionIngredientes(event){
var media=new Media("/android_asset/www/facebook_ringtone_pop.m4a");
media.play();
var a=$(this).listview("option","disabled");
//Código para cambiar ingredientes
}
```

7.3.8. Características de jQuery Mobile implementadas en la app.

7.3.8.1. Estructura de UI, Navegación y eventos.

El diseño de una aplicación desarrollada mediante jQuery Mobile, consistirá en una serie de páginas web o ventanas con las que interactuara el usuario.

Para poder utilizar jQuery Mobile en cada una de estas páginas, se incluirán, las librerías siguientes dentro de la etiqueta <head> del documento HTML:

- El archivo jQuery Mobile CSS (jquery.mobile-x.x.x.min.css).
- La librería jQuery (jquery-x.x.x.min.js)
- La librería jQuery Mobile (jquery.mobile-x.x.x.min.js).

Hay que añadir, que JQuery Mobile proporcionas dos formas diferentes en las que se puede organizar la estructura de una página:

- Estructura Página Única: Se define añadiéndole a un div el atributo *data-role="page"*.
- Estructura Multipágina: JQuery Mobile permitirá integrar múltiples páginas dentro de un único documento HTML. Con lo que el navegador sólo deberá cargar una única página, lo que mejorará la experiencia de usuario.
  Además, en este tipo de páginas cuando se desee pasar de una página a otra, cada una de ellas serán identificadas mediante su identificador, y no mediante su documento HTML como en el caso anterior.

Cuando haya que decidir qué tipo de estructura utilizar se deberá tener en cuenta que:

- En los documentos multipágina hay mayor consumo de banda ancha al cargarse, pero menor tiempo de repuesta cuando se realice una petición al servidor.

- En los documentos de página única estos consumirán menos banda ancha de carga, pero el tiempo de respuesta de cada petición al servidor será mayor.

Por lo tanto, si se desea crear una aplicación en que se acceda a todas las páginas secuencialmente se utilizará la estructura multipágina. Pero cuando el acceso a todas las páginas no sea estrictamente necesario, se emplearán varios documentos de página única.

Según esta lógica la estructura más adecuada para esta aplicación será la multipágina, ya que en este caso es seguro que el usuario acceda a todas las páginas secuencialmente, y por lo tanto, cuando menor sea el tiempo de carga, mejor será la experiencia de usuario.

index.html

```
<body>
        <div data-role="page" id="crusts">
        </div>

        <div data-role="page" id="toppings">
        </div>

        <div data-role="page" id="details">
        </div>
</body>
```

Para poder navegar entre las ventanas se crearán enlaces y se utilizará en ello el atributo *data-role="none"*, para crear un enlace que sea representado mediante una imagen determinada. Y después se crearán una serie de eventos, que se asignarán a cada uno de los botones de enlace creados, para ejecutar cada una de las funciones asignadas a las ventanas.

Además se podrá asignar el tipo de transición ejecutada, mediante el atributo *data-transition*, al que se podrán asignar los valores:

- slide: Desplaza la nueva página de derecha a izquierda, y se regresa a la página anterior de izquierda a derecha.
- slideup: Desplaza la nueva página de abajo a arriba, y si se regresa a la página anterior de arriba a abajo.
- slidedown: Desplaza la nueva página de arriba abajo, y si se regresa a la página anterior de abajo a arriba.
- pop: Expande la nueva página. Es muy útil para diálogos y popups.
- fade: Desvanece la página anterior hasta mostrar la nueva página.
- flip: Da la vuelta la página anterior a la nueva página.
- none: No se ejecuta ningún efecto a la transición.

index.html

```
<body>
        <div data-role="page" id="crusts" data-position="fixed">
                <a href="#toppings" data-role="none" data-transition="none" class="btn"
id="it"><img src="css/images/toppings_next.png" /></a>

        </div>

        <div data-role="page" id="toppings" data-position="fixed">
                <a href="#crusts" data-role="none" data-transition="none" class="btn_iz"
id="vc"><img src="css/images/cancel.png" /></a>
                <a href="#details" data-role="none" data-transition="none" class="btn_der"
```

```
id="id"><img src="css/images/details.png" /></a>

                    </div>

                    <div data-role="page" id="details" data-position="fixed">
                              <a href="#toppings" data-role="none" data-transition="none" class="btn_iz"
id="vt"><img src="css/images/cancel.png" /></a>
                              <div id="pedido"></div>
                    </div>


          </body>
```

index.js

```
document.addEventListener("deviceready",onDeviceReady,false);

function onDeviceReady(){
          $(document).ready(function(){
                    getHora();
                    inicio();
                    $("#it").on("tap",irToppings);
                    $("#vc").on("tap",volverCrusts);
                    $("#id").on("tap",irDetails);
                    $("#vt").on("tap",volverToppings);
                    $("#pedido").on("tap",hacerPedido);
          });
}
function irToppings(){
}
function volverCrusts(){
}
function irDetails(){
}
function volverToppings(){
}
function hacerPedido(){
}
```

Sin embargo, los botones de enlace no serán la única forma de poder navegar entre las ventanas, también puede utilizarse el método *change ()*, al que se le pasarán los parámetros siguientes:

change (to, options);

- to: Es la dirección URL o referencia de la página a la que se desea navegar.
- options: Se utilizará para configurar la navegación.
    - role: Es el rol, o el tipo de página que se desea cargar. Su valor por defecto es el que se le asigno mediante el atributo data-role.
    - transition: Se utiliza para configurar el tipo de transición a la página elegida.

Este método se utilizará cuando se muestra el diálogo de confirmación al realizar un pedido con éxito, este se cierra y se abre la ventana crusts para poder realizar un nuevo pedido.

index.html

```
  <body>
                    <div data-role="page" id="crusts" >
                              <a href="#toppings" data-role="none" data-transition="none" class="btn"
id="it"><img src="css/images/toppings_next.png" /></a>
```

```
                    </div>

                    <div data-role="page" id="toppings" >
                            <a href="#crusts" data-role="none" data-transition="none" class="btn_iz"
id="vc"><img src="css/images/cancel.png" /></a>
                            <a href="#details" data-role="none" data-transition="none" class="btn_der"
id="id"><img src="css/images/details.png" /></a>

                    </div>

                    <div data-role="page" id="details">
                            <a href="#toppings" data-role="none" data-transition="none" class="btn_iz"
id="vt"><img src="css/images/cancel.png" /></a>
                            <div id="pedido"></div>
                    </div>


        </body>
```

index.js

```
document.addEventListener("deviceready",onDeviceReady,false);

function onDeviceReady(){
        $(document).ready(function(){
      //resto de funciones y eventos
      $("#pedido").on("tap",hacerPedido);
        });
}

function hacerPedido(){
        if($("#names").val()==" "||$("#names").val()=="Nombre"||
         $("#address1").val()==" "||$("#address1").val()=="Dirección"||
          $("#address2").val()==" "||$("#address2").val()=="Población,Ciudad,Código
Postal"||$("#email").val()==" "||$("#email").val()=="Su email"){
                alert("Rellene todos los campos");
        }
        else{
        $.ajax({
    //se le pasan los parámetros a la petición HTTP
        .done(function(data){ //se ha realizado la petición
                if(response.mail==true){
                        navigator.notification.alert(
                                "Tu pedido ha sido enviado",
                                reiniciarapp,
                                "Éxito",
                                "OK"
                        );
                        }
                else{
        alert("PHP ha fallado al enviar pedido al correo electrónico");

                }
        })
}
}
function reiniciarapp(){
        $( ":mobile-pagecontainer" ).pagecontainer( "change", "#crusts");
        volverCrusts();}
```

7.3.8.2. Componente de UI: Lista Masas.

Se utiliza para poder crear una lista de las masas que serán seleccionadas por el usuario, en que cada vez que se desee elegir la masa siguiente, se deberá realizar un clic sobre la masa actual, se seguirán los pasos siguientes:

En primer lugar se deberán crear la vista de las masas utilizando la etiqueta en el documento HTML, a la que se le asignarán una serie de clases diferentes. Y se añadirá una etiqueta con el nombre correspondiente a la masa seleccionada, utilizando para ello un array con los nombres de cada una de las masas que podrán ser utilizadas.

index.html

```
<div data-role="page" id="crusts" >
                    <div class="masas"></div>
                    <div class="puntos"></div>
                  <p class="masa">Hecha a Mano</p>
                    <a href="#toppings" data-role="none" data-transition="none" class="btn"
id="it"><img src="css/images/toppings_next.png" /></a>

 </div>
```

estilos.css

```
/*Crusts*/
.masa{
        position: fixed;
        top:170px;
        height:20px;
        left:30px;
        right:30px;
        font-family:Verdana;
        font-weight:bold;
        font-size:16pt;
        color:#ffffff;
        text-shadow: 1px 1px #333333;
        text-align:center;
}
/*Crusts Cambia*/
div.masas{
        position: fixed;
        top:190px;
        height:180px;
        left:30px;
        right:30px;
        background-repeat:no-repeat;
        background-position:center;
}
div.masas.hand{
        background-image:url("images/crust/hand.png");
}
div.masas.natural{
        background-image:url("images/crust/natural.png");
}
div.masas.pan{
        background-image:url("images/crust/pan.png");
}
div.masas.stuffed{
        background-image:url("images/crust/stuffedCrust.png");
}
div.masas.thin{
        background-image:url("images/crust/thinNcrispy.png");
}
```

```
/*Crusts Cambio Puntos*/
div.puntos{
	position: fixed;
	top:335px;
	height:40px;
	left:30px;
	right:30px;
	background-repeat:no-repeat;
	background-position:center;
}
div.puntos.p_mano{
	background-image:url("images/puntos/p_masa.png");
}
div.puntos.p_natural{
	background-image:url("images/puntos/p_natural.png");
}
div.puntos.p_pan{
	background-image:url("images/puntos/p_pan.png");
}
div.puntos.p_rellena{
	background-image:url("images/puntos/p_rellena.png");
}
div.puntos.p_fina{
	background-image:url("images/puntos/p_fina.png");
}
```

```
var masas=["Hecha a Mano","Natural","De Pan","Rellena","Fina y Crujiente"];
```
index.js

Después se ejecuta la función *inicio ()*, encargada de mostrar la primera masa que será elegida por defecto. Y se asigna al evento tap a la vista de las masas, para que cada vez que se ejecute un toque sobre ella cambie la masa elegida, y se muestra su imagen y su nombre.

```
document.addEventListener("deviceready",onDeviceReady,false);
```
index.js

```
function onDeviceReady(){
	$(document).ready(function(){
		inicio();
		$("div.masas").on("tap",cambioMasa);
		$("div.puntos").addClass("p_mano");
	});
}

var i;
var n;
var masas=["Hecha a Mano","Natural","De Pan","Rellena","Fina y Crujiente"];

function inicio(){
	i=0;
	$(".masa").html("Hecha a Mano");
	$("div.masas").addClass("hand");
}

function cambioMasa(event){
	var media=new Media("/android_asset/www/facebook_ringtone_pop.m4a");
	media.play();
	i++;
	n=i-1;
	if(i<=4){
		if(masas[n]=="Hecha a Mano"){	$(this).removeClass( "hand" );$(this).addClass( "natural"
```

```
);
                    $("div.puntos").removeClass("p_mano");
        $("div.puntos").addClass("p_natural");}
                    if(masas[n]=="Natural"){$(this).removeClass( "natural" );$(this).addClass( "pan" );
                    $("div.puntos").removeClass("p_natural");    $("div.puntos").addClass("p_pan");}
                    if(masas[n]=="De Pan"){$(this).removeClass( "pan" );$(this).addClass( "stuffed" );
                    $("div.puntos").removeClass("p_pan");
        $("div.puntos").addClass("p_rellena");}
                    if(masas[n]=="Rellena"){$(this).removeClass( "stuffed" );$(this).addClass( "thin" );
                    $("div.puntos").removeClass("p_rellena");    $("div.puntos").addClass("p_fina");}
                    $(".masa").html(masas[i]);
        }
        else{
                    i-=5;
                    $(this).removeClass( "thin" );
                    $(this).addClass( "hand" );
                    $("div.puntos").removeClass("p_fina");
                    $("div.puntos").addClass("p_mano");
                    $(".masa").html(masas[i]);

        }
}
```

Figura 51: Lista de Masas.

Y finalmente al ejecutar el botón para ir a la selección de ingredientes, el nombre de la masa elegida será guardado, y se mostrará la imagen de la masa en la página siguiente.

index.html

```
<div data-role="page" id="crusts" >
        <div class="masas"></div>
        <div class=”puntos”></div>
        <p class="masa">Hecha a Mano</p>
        <a href="#toppings" data-role="none" data-transition="none" class="btn" id="it"><img
src="css/images/toppings_next.png" /></a>

 </div>
```

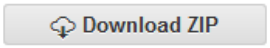
index.js

```
document.addEventListener("deviceready",onDeviceReady,false);

function onDeviceReady(){
        $(document).ready(function(){
                inicio();
                $("div.masas").on("tap",cambioMasa);
                $("#it").on("tap",irToppings);
});
```

```
}

function irToppings(){
	var storage=window.localStorage;
	storage.setItem("gmasa",masas[i]);
}
```

7.3.8.3. Componente de UI: Lista Deslizable.

En la página toppings, se creará una lista deslizable con la que se seleccionarán los ingredientes que se deseen añadir a la pizza.

Para ello se descargará iscrollview yendo a la sección Resources de la página principal de jQuery Mobile, y en el apartado 3rd party extensions, se elegirá el enlace al código de iscrollview. Este enlace llevará a su repositorio en GitHub, que contendrá tanto la documentación sobre su uso, como su enlace de descarga en 

.

Y se descomprime el archivo .zip, y siguiendo la ruta demo->sources, se copiaran los siguientes archivos en www/javascript:

- iscroll.js
- jquery.mobile.iscrollview.js
- layer_xy_hack.js
- pull-example.js

Y las hojas de estilo en www/css:

- jquery.mobile.iscrollview.css
- jquery.mobile.iscrollview-pull.css

Y después se añadirán las librerías de iscrollview en la cabecera del documento.

index.html
```
<head>
	<link rel="stylesheet" href="css/jquery.mobile.iscrollview-pull.css" />
	<link rel="stylesheet" href="css/jquery.mobile.iscrollview.css" />
	<script type="text/javascript" src="javascript/jquery.mobile.iscrollview.js"></script>
	<script type="text/javascript" src="javascript/layer_xy_hack.js"></script>
	<script type="text/javascript" src="javascript/pull-example.js"></script>
</head>
```

Ahora se creará una lista utilizando la etiqueta <ul>, a la que se le asigna los atributos *data-role="listview"*, y *data-inset="true"*, para crear una lista tipo recuadro con jQuery Mobile.

Pero para que esta lista sea deslizable se creará una vista a la que se le asignará el atributo *data-iscroll*, y dentro de esta vista se incluirá la lista.

index.html
```
<div data-role="page" id="toppings">


	<div class="myList" data-iscroll>
		<ul data-role="listview" data-inset="true" id="lingredientes">
```

```
                    </ul>
            </div>
</div>
```

```
.myList{                                                    estilos.css
            position:fixed;
            top:155px;
            max-height:120px;
            width:auto;
}
```

Ahora se procederá a implementar el código de la selección de ingredientes. Para ello primero se asigna a los elementos de la lista por defecto el atributo *disable: true* y la clase "no", para indicar que ninguno de los elementos de la lista han sido seleccionados. Todo esto se ejecutará en la función *inicio ()*.

```
li{                                                         estilos.css
            font-family:Verdana;
            font-weight:bold;
            font-size:14pt;
            color:#ffffff;
            text-shadow: 1px 1px #333333;
            text-align:left;
            width:340px;
            height:16px;
            max-height:16px;
            background-color:#8C0221;
            background-repeat:no-repeat;

}
li.no{
            background-image:url("images/checkbox_no.png");
}
li.yes{
            background-image:url("images/checkbox_yes.png");
}
```

```
document.addEventListener("deviceready",onDeviceReady,false);          index.html

function onDeviceReady(){
            $(document).ready(function(){
                        inicio();
});
}
function inicio(){
            i=0;
            $(".masa").html("Hecha a Mano");
            $("div.masas").addClass("hand");
            $("div.puntos").addClass("p_mano");
            $("li").addClass("no");
            $("li").listview({disabled:true});
}
```

Después se ejecuta la función *irToppings ()*, y se muestra la imagen de la masa actual que ha sido elegida, y se eliminan todas las imágenes de las masas e ingredientes que hayan podido ser elegidas con anterioridad.

index.html

```
<div data-role="page" id="crusts">

      <p class="fecha"></p>
      <div class="barra"></div>
      <p class="elige">1.Elige una Masa</p>

      <div class="masas"></div>
       <div class=”puntos”></div>
      <p class="masa">Hecha a Mano</p>
      <a href="#toppings" data-role="none" data-transition="none" class="btn" id="it"><img
      src="css/images/toppings_next.png" /></a>

</div>

<div data-role="page" id="toppings" >
       <div class="myList" data-iscroll>
            <ul data-role="listview" data-inset="true" id="lingredientes">
            </ul>
       </div>

      <div class="masag"></div>
</div>
```

estilos.css

```
div.masag{
            position: fixed;
            top:270px;
            width:216px;
            height:156px;
            left:30px;
            right:30px;
            background-repeat:no-repeat;
            background-position:center;
}
div.masag.hand{
            background-image:url("images/crust/hand.png");
}
div.masag.natural{
            background-image:url("images/crust/natural.png");
}
div.masag.pan{
            background-image:url("images/crust/pan.png");
}
div.masag.stuffed{
            background-image:url("images/crust/stuffedCrust.png");
}
div.masag.thin{
            background-image:url("images/crust/thinNcrispy.png");
}
```

index.js

```
document.addEventListener("deviceready",onDeviceReady,false);

function onDeviceReady(){
            $(document).ready(function(){
                        $("#it").on("tap",irToppings);
});
```

```
}
function irToppings(){
	numToppings=0;
	$("li").removeClass("yes");
	$("li").addClass("no");
	$("li").listview({disabled:true});

	$("div.masag").removeClass("hand");
	$("div.masag").removeClass("natural");
	$("div.masag").removeClass("pan");
	$("div.masag").removeClass("stuffed");
	$("div.masag").removeClass("thin");

	var storage=window.localStorage;
	storage.setItem("gmasa",masas[i]);
	var masa=storage.getItem("gmasa");

	if(masa=="Hecha a Mano"){$("div.masag").addClass("hand");}
	if(masa=="Natural"){$("div.masag").addClass("natural");}
	if(masa=="De Pan"){$("div.masag").addClass("pan");}
	if(masa=="Rellena"){$("div.masag").addClass("stuffed");}
	if(masa=="Fina y Crujiente"){$("div.masag").addClass("thin");}

	for(var j=0;j<toppings.length;j++){
		if(toppings[j].ids==1){
			toppings[j].ids=0;
			if(j==0){$("div.ingbacon").removeClass("bacon");}
			if(j==1){$("div.ingbeef").removeClass("beef");}
			if(j==2){$("div.inggrilled").removeClass("grilled");}
			// Aquí se quitan el resto de vistas de los ingredientes
		}
	}
}
```

Más adelante se define un array con los nombres de los ingredientes, y con una variable ids que indica con el valor 0 que el ingrediente no ha sido elegido, y el valor 1 que si lo ha sido. Además, se definen dos variables, una para establecer el número máximo de ingredientes elegidos, y otra para guardar el número de ingredientes elegidos.

index.html

```
var maxToppings=6;
var numToppings=0;
var toppings=[{title:"Bacon",ids:0},{title:"Ternera",ids:0},{title:"Pollo a la plancha",ids:0},
	{title:"Jamon",ids:0},{title:"Salami (Desmenuzado)",ids:0},{title:"Salami
(Cortado)",ids:0},{title:"Jalapeños",ids:0},
	{title:"Champiñones",ids:0},{title:"Olivas Negras",ids:0},{title:"Olivas Verdes",ids:0},
	{title:"Cebolla Roja",ids:0},{title:"Cebolla Blanca",ids:0},{title:"Pepperoni",ids:0},
	{title:"Chiles Banana",ids:0},{title:"Chiles Verdes",ids:0},{title:"Chiles Rojos",ids:0},
	{title:"Piña",ids:0},{title:"Cerdo",ids:0},{title:"Tomate Troceado",ids:0},{title:"Tomates Marinados",ids:0},
	{title:"Tomates Pera", ids: 0}];
```

Posteriormente se comienza con la implementación del código para la selección de ingredientes. Y para ello se le asignará a cada uno de los elementos de la lista el evento tap que ejecutará la función *seleccionIngredientes ().*

Para que la función funcione se deberá asignar un identificador único a cada uno de los elementos de la lista, para que cada vez que uno de estos sea seleccionado, se muestre su imagen correspondiente, se modifique el valor de su atributo *disabled* a false (si se selecciona)

o true (si se deselecciona), y se cambie el valor dela variable ids del array toppings a 1 (si se selecciona) y 0 (si se deselecciona).

Dentro de esta función también se deberá tener en cuenta que el número de ingredientes seleccionados en cada momento no exceda el número máximo de ingredientes permitidos y actuar en consecuencia.

index.html

```
<div data-role="page" id="toppings" >

        <p class="fecha"></p>
        <div class="barra"></div>
        <p class="elige">2.Elige los Ingredientes</p>

        <div class="myList" data-iscroll>
                <ul data-role="listview" data-inset="true" id="lingredientes">
                        <li id="#1">Bacon</li>
                        <li id="#2">Ternera</li>
                        <li id="#3">Pollo a la plancha</li>
                        <li id="#4">Jamón</li>
                        <!-- Se incluyen el resto de ingredientes en la lista-->
                </ul>
        </div>

        <div class="masag"></div>
        <div class="ingbacon"></div>
        <div class="ingbeef"></div>
        <div class="inggrilled"></div>
       <!--Aquí van el resto de vistas de los ingredientes-->
</div>
```

estilos.css

```
div.ingbacon{
        position: fixed;
        top:270px;
        width:216px;
        height:156px;
        left:30px;
        right:30px;
        background-repeat:no-repeat;
        background-position:center;
}
div.ingbacon.bacon{
        background-image:url("images/toppings/bacon_bits.png");
}
div.ingbeef{
        position: fixed;
        top:270px;
        width:216px;
        height:156px;
        left:30px;
        right:30px;
        background-repeat:no-repeat;
        background-position:center;
}
div.ingbeef.beef{
        background-image:url("images/toppings/beef.png");
}
div.inggrilled{
        position: fixed;
        top:270px;
        width:216px;
```

```
        height:156px;
        left:30px;
        right:30px;
        background-repeat:no-repeat;
        background-position:center;
}
div.inggrilled.grilled{
        background-image:url("images/toppings/grilled_chicken.png");
}
/*Aquí están el resto de las vistas de las masas*/
```

index.js

```
document.addEventListener("deviceready",onDeviceReady,false);

function onDeviceReady(){
        $(document).ready(function(){
                $("li").on("tap",seleccionIngredientes);
});
}
function seleccionIngredientes(event){
var media=new Media("/android_asset/www/facebook_ringtone_pop.m4a");
var a=$(this).listview("option","disabled");
        if(a==false){
                media.play();
                  numToppings-=1;
                  $(this).removeClass("yes");
                  $(this).addClass("no");
                  $(this).listview("option","disabled",true);

                  var id=$(this).attr("id");
                  if( id == "#1") {$("div.ingbacon").removeClass("bacon");toppings[0].ids=0;}
                  if( id == "#2") {$("div.ingbeef").removeClass("beef");toppings[1].ids=0;}
                  if( id == "#3") {$("div.inggrilled").removeClass("grilled");toppings[2].ids=0;}
                  //Se comprueban el resto de Ingredientes de la lista
        }
        else{

                  if(numToppings<maxToppings){
                        media.play();
                           numToppings+=1;
                           $(this).removeClass("no");
                           $(this).addClass("yes");
                           $(this).listview("option","disabled",false);

                           var id=$(this).attr("id");
                           if( id == "#1") {$("div.ingbacon").addClass("bacon");toppings[0].ids=1;}
                           if( id == "#2") {$("div.ingbeef").addClass("beef");toppings[1].ids=1;}
                           if( id == "#3") {$("div.inggrilled").addClass("grilled");toppings[2].ids=1;}
                           //Se comprueban el resto de Ingredientes de la lista
                  }
                  else{
                           alert ("Has Elegido demasiados ingredientes: " + numToppings + " son el máximo.");
                  }
        }
}
```

Figura 52: Lista Deslizable.

Y finalmente se guardará los ingredientes elegidos asignándole al botón details, el evento tap, que ejecutará la función *irDetails ().*

index.html

```
 <div data-role="page" id="toppings" >
        <a href="#crusts" data-role="none" data-transition="none" class="btn_iz" id="vc"><img
src="css/images/cancel.png" /></a>
        <a href="#details" data-role="none" data-transition="none" class="btn_der" id="id"><img
src="css/images/details.png" /></a>

 </div>

 <div data-role="page" id="details">

        <a href="#toppings" data-role="none" data-transition="none" class="btn_iz" id="vt"><img
src="css/images/cancel.png" /></a>
        <div id="pedido"></div>

 </div>
```

index.js

```
document.addEventListener("deviceready",onDeviceReady,false);

function onDeviceReady(){
        $(document).ready(function(){
                $("#id").on("tap",irDetails);
});
}
function irDetails(){
  var ingredientesG=new Array();

  for(j=0;j<toppings.length;j++){
        if(toppings[j].ids==1){
                ingredientesG.push(toppings[j].title);
        }
  }
  storage.setItem("gingredientes",JSON.stringify(ingredientesG));
  //Resto de Código
  }
```

7.3.8.4. Componente de UI: Etiqueta con Hora Actual.

Para poder mostrar en todas y cada una de las páginas o ventanas que componen la aplicación la hora actual, se creará una vista en cada una de las páginas y se ejecutará la función *getHora()*, que hallará la hora y minutos actúales.

index.js

```
  <body>
                    <div data-role="page" id="crusts" >

                              <p class="fecha"></p>
                    </div>

                    <div data-role="page" id="toppings" >

                              <p class="fecha"></p>
                    </div>

                    <div data-role="page" id="details" >

                              <p class="fecha"></p>
</div>


          </body>
```

estilos.css

```
/* Elementos de todas las paginas*/
div[data-role="page"]{
          background-color:#8C0221;
          background-image:url("images/bg_main.png");
          background-repeat:no-repeat;
          background-size:cover;
}
.fecha{
          font-family:Verdana;
          font-weight:bold;
          font-size:14pt;
          color:#ffffff;
          text-shadow: 1px 1px #333333;
  text-align:'right';
          height:20px;
          position:fixed;
  top:45px;
  right:13px;
}
```

index.js

```
document.addEventListener("deviceready",onDeviceReady,false);

function onDeviceReady(){
          $(document).ready(function(){
                    getHora();
                    inicio();
                    $("div.masas").on("tap",cambioMasa);
                    $("#it").on("tap",irToppings);
                    $("li").on("tap",seleccionIngredientes);
                    $("#vc").on("tap",volverCrusts);
                    $("#id").on("tap",irDetails);
});
}
function getHora(){
```

```
            var d=new Date();
            var h=d.getHours();
            var m=d.getMinutes();

            m=checkHora(m);
            $(".fecha").html(h+":"+m+" ");
            t=setTimeout(function(){getHora()},500);
}

function checkHora(i){
            if (i<10){
                        i="0" + i;
            }
            return i;
}
```

Figura 53: Etiqueta Hora Actual.

7.3.8.5. Componente de UI: Elemento de Formulario (TextField).

Los elementos de un formulario serán contenidos dentro de un elemento con la clase *ui-field-contain*. Sin embargo, este último será un atributo opcional.

Para definir una entrada de texto se empleará un input con el atributo *type="text"*.

En el caso presentado, estas entradas de texto se utilizarán para introducir los datos necesarios para completar la información.

index.html

```
<div data-role="page" id="toppings">
            <a href="#crusts" data-role="none" data-transition="none" class="btn_iz" id="vc"><img
src="css/images/cancel.png" /></a>
            <a href="#details" data-role="none" data-transition="none" class="btn_der" id="id"><img
src="css/images/details.png" /></a>

 </div>

 <div data-role="page" id="details">
            <p class="fecha"></p>
            <input type="text" name="name" id="names" value=" " data-role="none">
            <input type="text" name="name" id="address1" value=" " data-role="none">
            <input type="text" name="name" id="address2" value=" " data-role="none">
           <input type="text" name="name" id="email" value=" " data-role="none">
            <div class="ver_pedido"></div>
```

```
            <a href="#toppings" data-role="none" data-transition="none" class="btn_iz" id="vt"><img
src="css/images/cancel.png" /></a>
            <div id="pedido"></div>

 </div>
```

estilos.css

```
/*Details*/
input#names{
            color:#336699;
            font-family:Verdana;
            font-size:13pt;
            position: fixed;
            top:100px;
            left:10px;
            width:300px;
            height:40px;
            background-image:url("images/textfield.png");
            background-size:300px 40px;
            background-repeat:no-repeat;
            background-color:transparent;
            padding-left:8px;
            padding-right:8px;
            border-style:none;
            border-color:transparent;
}
input#address1{
            color:#336699;
            font-family:Verdana;
            font-size:13pt;
            position: fixed;
            top:140px;
            left:10px;
            width:300px;
            height:40px;
            background-image:url("images/textfield.png");
            background-size:300px 40px;
            background-repeat:no-repeat;
            background-color:transparent;
            padding-left:8px;
            padding-right:8px;
            border-style:none;
            border-color:transparent;
}
input#address2{
            color:#336699;
            font-family:Verdana;
            font-size:13pt;
            position: fixed;
            top:180px;
            left:10px;
            width:300px;
            height:40px;
            background-image:url("images/textfield.png");
            background-size:300px 40px;
            background-repeat:no-repeat;
            background-color:transparent;
            padding-left:8px;
            padding-right:8px;
            border-style:none;
            border-color:transparent;
}
input#email{
```

```
            color:#336699;
            font-family:Verdana;
            font-size:13pt;
            position: fixed;
            top:220px;
            left:10px;
            width:300px;
            height:40px;
            background-image:url("images/textfield.png");
            background-size:300px 40px;
            background-repeat:no-repeat;
            background-color:transparent;
            padding-left:8px;
            padding-right:8px;
            border-style:none;
            border-color:transparent;
}
```

index.js

```
document.addEventListener("deviceready",onDeviceReady,false);

function onDeviceReady(){
            $(document).ready(function(){
          $("#id").on("tap",irDetails);
});
}
function irDetails(){
            //Código del Pedido Realizado
            $("#names").val("Nombre");
            $("#address1").val("Dirección");
            $("#address2").val("Población,Ciudad,Código Postal");
           $("#email").val("Su email");
}
```

Figura 54: Formulario.

7.3.8.6. Componente de UI: Etiqueta con el Pedido Realizado.

Para poder mostrar en ventanas details, la información sobre el pedido de pizza, cuando se introduzcan todos los datos relativos al usuario, se creará una vista utilizando la etiqueta en la que se representará la masa e ingredientes seleccionados.

index.html

```
<div data-role="page" id="toppings">
        <a href="#crusts" data-role="none" data-transition="none" class="btn_iz" id="vc"><img
src="css/images/cancel.png" /></a>
        <a href="#details" data-role="none" data-transition="none" class="btn_der" id="id"><img
src="css/images/details.png" /></a>

 </div>

 <div data-role="page" id="details">
        <p class="fecha"></p>
        <input type="text" name="name" id="names" value=" " data-role="none">
        <input type="text" name="name" id="address1" value=" " data-role="none">
        <input type="text" name="name" id="address2" value=" " data-role="none">
        <input type="text" name="name" id="email" value=" " data-role="none">
        <div class="ver_pedido"></div>
        <a href="#toppings" data-role="none" data-transition="none" class="btn_iz" id="vt"><img
src="css/images/cancel.png" /></a>
        <div id="pedido"></div>

 </div>
```

estilos.css

```
div.ver_pedido{
        font-family:Verdana;
        font-size:10pt;
        color:#ffffff;
        text-shadow: 1px 1px #333333;
        text-align:left;
        height:160px;
        width:inherit;
        position:fixed;
        top:260px;
        left:10px;
        line-height:5px;
}
```

index.js

```
document.addEventListener("deviceready",onDeviceReady,false);

function onDeviceReady(){
        $(document).ready(function(){
       $("#id").on("tap",irDetails);
});
}
function irDetails(){
        var storage=window.localStorage;
        var masa=storage.getItem("gmasa");
        var ingredientesG=new Array();

  for(j=0;j<toppings.length;j++){
        if(toppings[j].ids==1){
                ingredientesG.push(toppings[j].title);
        }
  }
  storage.setItem("gingredientes",JSON.stringify(ingredientesG));

  var text="<p>Pizza "+masa+" con :</p>";
        var ing=JSON.parse(storage.getItem("gingredientes"));

  if(ing.length==0){
        text+="*Básica (Pizza de Queso) \n";
  }
```

```
    else{
            for(j=0;j<ing.length;j++){
                    text+="<p>* "+ing[j]+"</p>";
            }
    }
            $("div.ver_pedido").html(text);
      //Se añade el texto a los Inputs
}
```

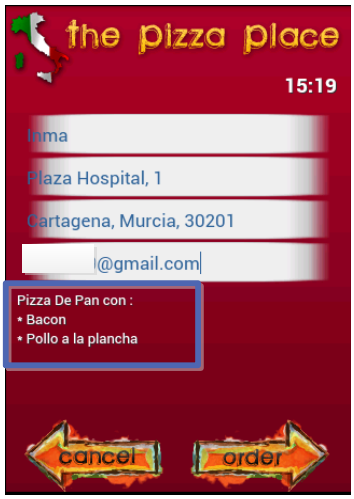


Figura 55: Etiqueta Pedido Realizado.

7.3.8.7. Trabajar con Datos Remotos vía Ajax.

Esta aplicación enviará el pedido a un servidor remoto, y para realizar este pedido se hará uso de Ajax.

AJAX (Asynchronous JavaScript and XML) permite la comunicación con un servidor o servicio web remoto sin la necesidad de recargar la página.

El método principal para realizar peticiones HTTP asíncronas Ajax es *$.ajax (),* a partir del cual se han definido otros métodos relacionados y especializados en tareas más concretas como *$.get ()* o *$.load ()*. Y al cuál se le pasan los parámetros siguientes:

```
$.ajax (url, opciones);
```

- url: Indica la URL del servidor al que fue enviada la petición.
- opciones: Sirven para configurar la petición Ajax.

Las opciones definidas en el parámetro opciones, pueden ser:

- async: Indica si la petición es asíncrona, y su valor por defecto es true.
- beforeSend: Define una función a la que se le pasa como argumento el objeto *XMLHttpRequest*, para modificarlo, antes de realiza la petición.
- always: Define la función que se ejecuta cuando se ha completado la petición, después de las funciones *success* y *error*, si han sido definidas. A esta función se le pasan como argumentos, el objeto *XMLHttpRequest*, como primer argumento, y el resultado de la petición, como segundo argumento.

- contentType: Define el valor de la cabecera Content-Type utilizada para realizar la petición. Su valor por defecto es *application/x-www-form-urlencoded.*
- data: Se utiliza para enviar parámetros al servidor. Puede ser o una cadena de texto parametro1=valor1&parametro2=valor2, o un array tipo clave/valor.
- dataType: Es el tipo de dato que se espera como respuesta. Sus posibles valores pueden ser:
    - xml: Devuelve un documento XML que puede procesarse vía JQuery.
    - html: Devuelve un documento HTML como texto.
    - script: Evalúa la respuesta como JavaScript y la devuelve como texto.
    - json: Evalúa la respuesta como JSON y devuelve un objeto JavaScript.
    - text: Devuelve un String de texto.
- fail: Define la función callback que es ejecutada cuando se produce un error en la petición. A esta función se le pasarán como parámetros:

  function (jqXHR jqXHR, String textStatust, String errorThrown)
    - jqXHR: Es el objeto XMLHttpRequest.
    - textStatus: Describe el tipo de error que se ha producido, sus valores pueden ser:
        - timeout.
        - error.
        - abort.
        - parseError.
    - errorThrown: Muestra la excepción producida en la petición sus valores pueden ser:
        - Not Found.
        - Internal Server Error.
- isModified: Considera que la petición es correcta, si y solo si, la respuesta recibida es diferente a la recibida anteriormente. Es un booleano, y su valor por defecto es false.
- processData: Indica si los datos pasados en la opción data son transformados en una cadena de texto.
- done: Define la función callback que se ejecuta, cuando la petición ha sido completada con éxito. La función recibe como parámetros:

  function (PlainObject data, String textStatus, jqXHR jqXHR);
    - data: Son los datos recibidos, con el formato indicado en la opción dataType.
    - textStatus: Es una cadena con el estado de la petición.
    - jqXHR: Es el objeto XMLHttpRequest.
- timeout: Indica el tiempo máximo, en milisegundos, en que la petición espera la respuesta del servidor antes de ser anulada.
- type: Es el tipo de petición realizada, sus valores pueden ser GET, que es su valor por defecto, o POST.

En el caso de la aplicación este realizará una petición HTTP empleando AJAX, cuando se complete el pedido de la pizza y este se envíe al servidor de MiPizzeria. Esta petición se realizará correctamente cuando el servidor envié el correo electrónico de confirmación. Y de forma incorrecta cuando no se introduzcan todos los datos de usuario necesarios, el correo

electrónico no sea enviado con éxito desde el servidor, o cuando se haya producido un error en la petición HTTP.

Y cuando esta petición se haya completado, se mostrará una notificación de confirmación, y se reiniciará la aplicación.

index.html

```
<body>
        <div data-role="page" id="crusts">

                <!—Aquí se añaden los componentes de esta página-->

        </div>

        <div data-role="page" id="toppings">

                <!-Aquí se añaden los componentes de esta página-->

        </div>

        <div data-role="page" id="details">

                <p class="fecha"></p>
                <input type="text" name="name" id="names" value=" " data-role="none">
                <input type="text" name="name" id="address1" value=" " data-role="none">
                <input type="text" name="name" id="address2" value=" " data-role="none">
              <input type="text" name="name" id="email" value=" " data-role="none">

                <div class="ver_pedido"></div>
                <a href="#toppings" data-role="none" data-transition="none" class="btn_iz"
id="vt"><img src="css/images/cancel.png" /></a>
                <div id="pedido"></div>

        </div>


    </body>
```

index.js

```
document.addEventListener("deviceready",onDeviceReady,false);

function onDeviceReady(){
    $(document).ready(function(){
        $("#pedido").on("tap",hacerPedido);
    });
}
function volverCrusts(){
    var storage=window.localStorage;
    var masa=storage.getItem("gmasa");
    i=0;

    if(masa=="Hecha a Mano"){$("div.masas").removeClass("hand");}
    if(masa=="Natural"){$("div.masas").removeClass("natural");}
    if(masa=="De Pan"){$("div.masas").removeClass("pan");}
    if(masa=="Rellena"){$("div.masas").removeClass("stuffed");}
    if(masa=="Fina y Crujiente"){$("div.masas").removeClass("thin");}

    storage.removeItem("gmasa");

    $(".masa").html(masas[i]);
    $("div.masas").addClass("hand");
}
```

```
function hacerPedido(){
	if($("#names").val()==" "||$("#names").val()=="Nombre"||$("#address1").val()==""||$("#address1").val()=="Dirección"||
	  $("#address2").val()==" "||$("#address2").val()=="Población,Ciudad,Código Postal"||$("#email").val()==" "||
	  $("#email").val()=="Su email"){

		alert("Rellene todos los campos");
	}
	else{
		var storage=window.localStorage;
		var masa=storage.getItem("gmasa");
		var ing=JSON.parse(storage.getItem("gingredientes"));
		if(ing.length==0){
			var a="Básica (Pizza de Queso)";
			ing.push(a);
		}
		var	params={names:$("#names").val(),address1:$("#address1").val(),address2:$("#address2").val(),
		crust:masa,toppings:JSON.stringify(ing),email:$("#email").val()};


	$.ajax({
		url:"http://IPServidor:80/submit_order.php",
	       //url:"http://10.0.2.2:80/submit_order.php",
		type:"POST",
		data:params
	})
	.done(function(data){
		var json=data;
		var response=JSON.parse(json);

		if(response.mail==true){
			navigator.notification.alert(
				"Tu pedido ha sido enviado",
				reiniciarapp,
				"Éxito",
				"OK"
			);
			}
		else{
			alert("PHP ha fallado al enviar pedido al correo electrónico");

		}
	})
	.fail(function(jqXHR, textStatus){
					alert("Error de red: "+textStatus);
	});
  }
}

function reiniciarapp(){
	$( ":mobile-pagecontainer" ).pagecontainer( "change", "#crusts");
	volverCrusts();
}
```

Nota: Cuando la aplicación se ejecute en el emulador, se conectará el servidor a partir de la dirección IP 10.0.2.2. Mientras que si se ejecuta en un dispositivo, la conexión se realizará a través de la IP de la máquina donde se encuentra el servidor, pero esto solo será válido cuando la máquina del servidor y el dispositivo estén conectados a la misma red WIFI.

7.3.9. Simulación de la Aplicación.

Para poder simular la aplicación en un dispositivo virtual se seguirán los pasos siguientes:

1. Se pulsa el botón  , y en la ventana que se abre se pulsa *New* y se crea un nuevo dispositivo virtual.
2. Se va a Run->Run Configurations, y se elige Target.
3. En Target, se elige el dispositivo virtual que se va a utilizar, para ello se elige "Automatically pick compatible device", y se selecciona el dispositivo virtual.
4. Se hace clic con el botón derecho sobre la carpeta del proyectos, y se elige Run As->Android Application.

7.3.10. Instalación en Dispositivo.

Para poder instalar la aplicación en un dispositivo HTC Desire se seguirán los pasos siguientes:

1. Se baja el HTC Driver en http://www.htc.com/us/support/.
2. En el dispositivo se marca la casilla que permite instalar aplicaciones de orígenes desconocidos, y se habilita la depuración Android.
3. Se conecta el dispositivo Android al ordenador mediante un cable USB.
4. Se inicia Eclipse.
5. Se va a Run->Run Configurations, y se va a Target.
6. En Target se pueden elegir entre los dispositivos disponibles, aquí se pulsa "Always promt to pick device" y se hace clic sobre el botón *Run*.
7. Se abre la ventana Android Device Chooser, en dónde aparecerá el teléfono en estado Online. Y en dónde se seleccionará este dispositivo y se pulsará Ok.

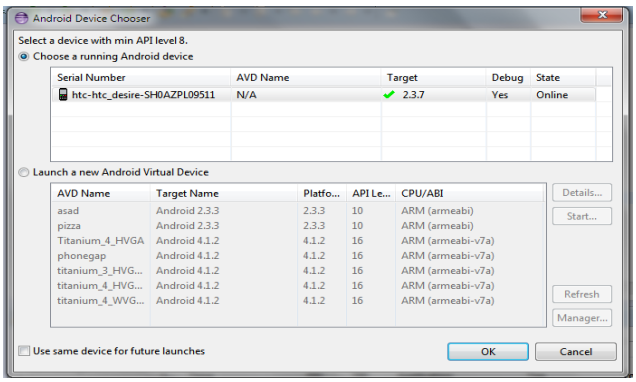


Figura 56: Ventana para seleccionar el dispositivo en que se ejecutará la aplicación en Eclipse.

7.4. Resultados.

7.4.1. Instalación.

Titanium Studio	Eclipse
Necesita una máquina con un S.O reciente, y al menos 4 GB de RAM.	Su instalación no está limitada por la memoria RAM o versión del S.O de la máquina.
Su instalador es capaz de instalar todo el Software necesario para el IDE automáticamente.	La instalación de plugins y resto de software necesario se realiza de forma manual.

7.4.2. Diferencias entre los Entornos de Desarrollo.

Titanium Studio	Eclipse
Se pueden acceder a documentación, tutoriales y ejemplos desde el Dashboard del entorno de desarrollo.	Se tendrá acceso a documentación y tutoriales a través del buscador web, y no a través de la herramienta.
Puede mostrar y detectar los errores de programación.	Sólo es capaz de detectar los errores de programación referentes al código Java. En cambio no es capaz de señalar los errores de código en HTML, CSS o JavaScript.
Posee un predictor de escritura, para autocompletar los métodos y variables.	Posee un predictor de escritura, para autocompletar los métodos y variables. Pero sólo funcionará con Java.
Podrá compilar código para todas las plataformas soportadas.	Sólo podrá compilar código para Android. Cuando se desee desarrollar para otras plataformas se podrá el compilador web multiplataforma PhoneGap Build disponible en https://build.phonegap.com/.

7.4.3. Creación de Proyecto.

Titanium Studio	Eclipse
El gestor permitirá elegir una serie de plantillas a partir de las cuales diseñar los proyectos.	El gestor permite elegir entre una serie de temas de Android desde las cuales diseñar el proyecto.
El gestor permitirá incluir la versión de Titanium SDK deseada.	Para poder utilizar PhoneGap el usuario deberá incluir la librería cordova.jar en su conjunto de librerías, y copiar cordova.js dentro de las carpetas del proyecto. Mientras que para utilizar jQuery Mobile solo deberá copiar sus archivos .css y .js en la carpeta del proyecto.
El gestor de proyectos permitirá elegir las plataformas para las que se desarrolla, entre Android, BlackBerry y iOS.	Eclipse sólo permitirá seleccionar la versión SDK Android que será utilizada

7.4.4. Código.

Titanium	PhoneGap y jQuery Mobile
Se utiliza una versión modificada de JavaScript, por lo que es un código más complejo.	Se utiliza HTML5, CSS y JavaScript para desarrollar las apps, lo que da facilita la programación.
Para poder acceder a las características de los dispositivos se emplean módulos proporcionados por Titanium. Como son Ti.Media o Ti.UI.	Para poder acceder a las características de los dispositivos se necesitará utilizar funciones de jQuery Mobile y PhoneGap, que funcionan como librerías externas.
El código resultante es relativamente corto, ya que se pueden configurar las funcionalidades, eventos y el UI de una aplicación dentro del mismo archivo.	El código resultante es bastante largo ya que utilizará: un documento HTML para definir la estructura de la aplicación, una hoja de estilos CSS para definir el aspecto, y un archivo JavaScript para definir las funciones y eventos de la aplicación.
El UI será definido por el programador desde cero.	Se tendrá acceso a las hojas de estilo proporcionadas por jQuery Mobile, para facilitar el diseño del UI.
Una parte del código será diferente para cada plataforma para la que se desarrolla.	Todo su código será válido para todas las plataformas.
Permite crear listas deslizables mediante de métodos internos.	Para poder crear una lista deslizable será necesario el uso de librerías externas, como iscroll.

8. Comparativas.

8.1. Titanium VS PhoneGap.

8.1.1. Plataformas Soportadas.

Plataformas	Titanium	PhoneGap	Observaciones
Android	✓	✓	
iOS	✓(*)	✓(*)	(*) Es necesario un ordenador con S.O. Apple
BlackBerry	✓	✓	
Windows Phone 7	✗	✓(*)	(*) Es necesario un ordenador con S.O. Windows 7 u 8
Windows Phone 8	✗	✓(*)	(*) Es necesario un ordenador con S.O. Windows 8
Otros	✓(*1)	✓(*2)	(*1) Mobile Web, (*2) Tizen y Bada.

8.1.2. APIs.

APIs		Titanium	PhoneGap	Observaciones
Notificaciones	Alertas	✓	✓	
	Diálogos	✓	✓	
	Avisos	✓	✓	
	Sonidos	✗	✓	
Vibración		✓	✓	
Eventos		✓	✓	
Gestos		✓(*)	✗	(*)Agitar
Información Dispositivo		✓	✓	
Conexión de Redes		✓	✓	
Acelerómetro		✓	✓	
Giroscopio		✓	✓	
Animaciones	Simples	✓	✗	
	2D/3D	✓	✗	
	Transiciones	✓	✗	
Datos Locales	Base de Datos SQLite	✓	✓	
	FileSystem	✓	✓	
	Otros	✓(*1)	✓(*2)	(*1) Propiedades, (*2) LocalStorage
Datos Remotos	Peticiones HTTP	✓	✗	
	Transferencia de Archivos	✓	✓	
	Datos JSON	✓	✗	
	Datos XML	✓	✗	
	Servicios SOAP	✓	✗	
Servicios de Localización	Geolocalización	✓	✓	
	Mapas Nativos	✓	✗	
	Brújula	✓	✓	
Acceso a Facebook		✓	✗	
Agenda de contactos		✓	✓	
Calendario		✓	✗	

Cámara	Captura Fotos	✓	✓
	Captura Video	✓	✓
	Acceso Cámara Frontal y Posterior	✓	✓
Galería de Fotos		✓	✓
Video	Reproducción	✓	✗
Audio	Reproducción	✓	✓
	Captura	✓	✓

8.1.3. Otros.

Otros	Titanium	PhoneGap	Observaciones
UI Nativo	✓	✗(*)	(*) Es necesario el uso de librerías como jQuery Mobile o Sencha Touch
Lenguaje Desarrollo	**JavaScript**	**JavaScript, HTML**	
Tipo App Resultante	**App Interpretada**	**App Híbrida**	
Licencia	**Libre y de Código Abierto**	**Libre y de Código Abierto**	
IDE Propio	✓(*1)	✗(*2)	(*1) Titanium Visual Studio, (*2) Se puede elegir entre Xcode, Eclipse, Microsoft Visual Studio o Apache Ant, según la plataforma para la que se desarrolle
Coste de Desarrollo	**$$(*1)**	**$$(*2)**	(*1) Las herramientas de desarrollo gratuitas, algunas licencias se pagan, (*2) Herramientas de desarrollo y licencias gratuitas
Soporte de Desarrollo	**Muy Bueno(*1)**	**Bueno(*2)**	(*1) Soporte IDE, Depuradores Integrados, (*2) Soporte algunos IDE , depuradores Chrome Devtools
Curva de Aprendizaje	**Muy Buena(*1)**	**Buena(*2)**	(*1) Acceso a tutoriales, video tutoriales, documentación APIs, consejos sobre buenas formas de programación, foros y soporte (*2) Acceso a tutoriales, documentación APIs, foros y soporte
Monetización	**Muy Bueno(*)**	**Muy Bueno(*)**	(*) Las apps desarrolladas son accesibles desde los App stores

8.2. Titanium VS jQuery Mobile.

8.2.1. Plataformas Soportadas.

Plataformas	Titanium	jQuery Mobile	Observaciones
Android	✓	✓	
iOS	✓(*)	✓(*)	(*) Es necesario un ordenador con S.O. Apple
BlackBerry	✓	✓	
Windows Phone 7	✗	✓(*)	(*) Es necesario un ordenador con S.O. Windows 7 u 8
Windows Phone 8	✗	✓(*)	(*) Es necesario un ordenador con S.O. Windows 8
Otros	✓(*1)	✓(*2)	(*1)Mobile Web, (*2) Bada, Meego y Tizen

8.2.2. APIs.

APIs		Titanium	jQuery Mobile	Observaciones
Notificaciones	Alertas	✓	✓	
	Diálogos	✓	✓	
	Avisos	✓	✓	
	Sonidos	✗	✗	
Vibración		✓	✗	
Eventos		✓	✓	
Gestos		✓(*)	✗	(*)Agitar
Información Dispositivo		✓	✗	
Conexión de Redes		✓	✗	
Acelerómetro		✓	✗	
Giroscopio		✓	✓	
Animaciones	Simples	✓	✓	
	2D/3D	✓	✗	
	Transiciones	✓	✓	
Datos Locales	Base de Datos SQLite	✓	✗	
	FileSystem	✓	✗	
	Otros	✓(*)	✗	(*) Propiedades
Datos Remotos	Peticiones HTTP	✓	✓	
	Transferencia de Archivos	✓	✗	
	Datos JSON	✓	✓	
	Datos XML	✓	✓	
	Servicios SOAP	✓	✗	
Servicios de Localización	Geolocalización	✓	✗	
	Mapas Nativos	✓	✗	
	Brújula	✓	✗	
Acceso a Facebook		✓	✗	
Agenda de contactos		✓	✗	
Calendario		✓	✗	
Cámara	Captura Fotos	✓	✗	
	Captura Video	✓	✗	
	Acceso Cámara Frontal y Posterior	✓	✗	
Galería de Fotos		✓	✗	
Video	Reproducción	✓	✗	
Audio	Reproducción	✓	✗	
	Captura	✓	✗	

8.2.3. Otros.

Otros	Titanium	jQuery Mobile	Observaciones
UI Nativo	✓	✓	
Lenguaje Desarrollo	**JavaScript**	**JavaScript, HTML**	
Tipo App Resultante	**App Interpretada**	**Web App**	
Licencia	**Libre y de Código Abierto**	**Libre y de Código Abierto**	
IDE Propio	✓(*1)	✗(*2)	(*1) Titanium Visual Studio, (*2) Se puede elegir entre Xcode, Eclipse, Microsoft Visual Studio o Apache Ant, según la plataforma para la que se desarrolle
Coste de Desarrollo	**$$(*1)**	**$$(*2)**	(*1) Las herramientas de desarrollo gratuitas, algunas licencias se pagan, (*2) Herramientas de desarrollo y licencias gratuitas
Soporte de Desarrollo	**Muy Bueno(*1)**	**Bueno(*2)**	(*1) Soporte IDE, Depuradores Integrados, (*2) Soporte algunos IDE, Depuradores: Chrome DevTools
Curva de Aprendizaje	**Muy Buena(*1)**	**Buena(*2)**	(*1) Acceso a tutoriales, video tutoriales, documentación APIs, consejos sobre buenas formas de programación, foros y soporte. (*2) Acceso a documentación APIs, demos, foros y soporte
Monetización	**Muy Bueno(*1)**	**Regular(*2)**	(*1) Las apps desarrolladas son accesibles desde los App stores (*2) Apps no son accesibles en las App stores (En Navegadores)

8.3. PhoneGap VS jQuery Mobile.

8.3.1. Plataformas Soportadas.

Plataformas	jQuery Mobile	PhoneGap	Observaciones
Android	✓	✓	
iOS	✓(*)	✓(*)	(*) Es necesario un ordenador con S.O. Apple
BlackBerry	✓	✓	
Windows Phone 7	✓(*)	✓(*)	(*) Es necesario un ordenador con S.O. Windows 7 u 8
Windows Phone 8	✓(*)	✓(*)	(*) Es necesario un ordenador con S.O. Windows 8
Otros	✓(*1)	✓(*2)	(*1)Bada, Meego y Tizen, (*2) Bada y Tizen

8.3.2. APIs.

APIs		jQuery Mobile	PhoneGap	Observaciones
Notificaciones	Alertas	✓	✓	
	Diálogos	✓	✓	
	Avisos	✓	✓	
	Sonidos	✗	✓	
Vibración		✗	✓	
Eventos		✓	✓	
Gestos		✗	✗	

Información Dispositivo		✗	✓	
Conexión de Redes		✗	✓	
Acelerómetro		✗	✓	
Giroscopio		✓	✓	
Animaciones	Simples	✓	✗	
	2D/3D	✗	✗	
	Transiciones	✓	✗	
Datos Locales	Base de Datos SQLite	✗	✓	
	FileSystem	✗	✓	
	Otros	✗	✓(*)	(*) LocalStorage
Datos Remotos	Peticiones HTTP	✓	✗	
	Transferencia de Archivos	✗	✓	
	Datos JSON	✓	✗	
	Datos XML	✓	✗	
	Servicios SOAP	✗	✗	
Servicios de Localización	Geolocalización	✗	✓	
	Mapas Nativos	✗	✗	
	Brújula	✗	✓	
Acceso a Facebook		✗	✗	
Agenda de contactos		✗	✓	
Calendario		✗	✗	
Cámara	Captura Fotos	✗	✓	
	Captura Video	✗	✓	
	Acceso Cámara Frontal y Posterior	✗	✓	
Galería de Fotos		✗	✓	
Video	Reproducción	✗	✗	
	Captura	✗	✓	
Audio	Reproducción	✗	✓	
	Captura	✗	✓	

8.3.3. Otros.

Otros	jQuery Mobile	PhoneGap	Observaciones
UI Nativo	✓	✗(*)	(*) Es necesario el uso de librerías como jQuery Mobile o Sencha Touch
Lenguaje Desarrollo	**JavaScript, CSS, HTML**	**JavaScript, CSS, HTML**	
Tipo App Resultante	**Web App**	**App Híbrida**	
Licencia	**Libre y de Código Abierto**	**Libre y de Código Abierto**	
IDE Propio	✗(*)	✗(*)	(*) Se puede elegir entre Xcode, Eclipse, Microsoft Visual Studio o Apache Ant, según la plataforma para la que se desarrolle
Coste de Desarrollo	$$(*)	$$(*)	(*) Herramientas de desarrollo y licencias gratuitas
Soporte de Desarrollo	**Bueno(*)**	**Bueno(*)**	(*) Soporte algunos IDE , depuradores Chrome Devtools

Curva de Aprendizaje	**Buena(*1)**	**Buena(*2)**	(*1) Acceso a documentación APIs, demos, foros y soporte (*2) Acceso a tutoriales, documentación APIs, foros y soporte
Monetización	**Regular(*1)**	**Muy Bueno(2)**	(*1)Apps no accesibles en los App stores (En Navegadores) (*2) Las apps desarrolladas son accesibles desde los App stores

8.4. Conclusiones y Posibles Trabajos Futuros.

Como se puede observar en cada una de las comparativas los puntos más importantes al elegir un framework para crear una aplicación son:

- Especificar en qué plataformas debe funcionar la aplicación: Si se desease realizar una aplicación que funcionase en todas las plataformas móviles sería más adecuado utilizar PhoneGap o jQuery Mobile. Mientras que si solo fuese necesario que funcionase para iOS, Android y BlackBerry, Titanium también sería adecuado.
- Especificar que recursos de los dispositivos se desean utilizar: Titanium permite tener acceso a todos los recursos de los dispositivos, tanto hardware como software. Mientras que PhoneGap tendría acceso a la mayoría de estos recursos, pero no a todos ellos. Ya que por ejemplo, no tendría acceso a los mapas nativos. Y jQuery Mobile no tendría acceso a ninguno de los recursos del hardware.
- Implementación de UI: Cuando se realizase la implementación del UI se debe tener en cuenta que mientras que con Titanium se puede implementar un UI con aspecto nativo, PhoneGap por sí mismo es incapaz. Ya que al utilizar HTML5, el aspecto del UI sería parecido al de una página web y no el de una app.
  Por lo tanto, PhoneGap necesita de otras herramientas para implementar el UI, como por ejemplo jQuery Mobile, el cual proporciona una serie de temas y plantillas para diseñar el UI.
- Lenguaje de Desarrollo: También se debe tener qué en cuenta que mientras que en PhoneGap y jQuery Mobile, las apps se desarrollan de igual forma que una página web, lo que disminuye el tiempo y coste de desarrollo, y además mejora la curva de aprendizaje. Titanium utiliza una versión modificada de JavaScript, lo que hace que se necesite cierto tiempo para aprender el lenguaje de programación, y los costes y tiempos de desarrollo aumenten.
- Entorno de Desarrollo: Titanium posee un IDE propio, Titanium Studio, lo que facilita su instalación, y acceso a las diferentes herramientas de desarrollo de apps. Mientras que PhoneGap y jQuery Mobile, necesitarán de diferentes IDE, dependiendo de la plataforma para la que se desarrolla, para implementar la aplicación.
- Monetización: Si se desea que las apps sean accesibles desde los App stores, y por lo tanto que los usuarios tengan acceso a su descarga y actualizaciones, y también recibir ciertos beneficios por su uso, se utilizarán PhoneGap o Titanium.

A partir de los cuales el futuro programador será capaz de elegir el framework y estrategia multiplataforma más adecuados para su aplicación.

Además, esta comparativa permitirá a los futuros usuarios de estos frameworks conocerlos en profundidad e instalarlos y utilizarlos correctamente. Es decir, le proporcionará una guía a partir de la cual poder realizar sus futuras apps.

Pero también servirá como punto de inicio a partir del cual conocer y estudiar otros frameworks como:

- RhoMobile: Permite crear apps empleando como lenguaje de programación Ruby y posee un IDE propio, RhoStudio, que podrá obtenerse en https://developer.motorolasolutions.com/community/rhomobile-suite/rhomobile-downloads. Dará como resultado apps interpretadas.
- Xamarin: Permite desarrollar aplicaciones válidas para distintas plataformas a partir del mismo código C#, posee un IDE propio, Xamarin Studio, capaz de compilar y ejecutar aplicaciones para iOS, Android y Windows Phone, y que puede obtenerse en http://xamarin.com/download. Dará como resultado una app utilizando un enfoque dirigido por plataformas.
- Sencha Touch: Permite desarrollar web apps utilizando como lenguaje de programación una versión de JavaScript. Puede ser descargado en http://www.sencha.com/products/touch.

En definitiva, este estudio proporciona una guía de como emplear los diferentes enfoques que se pueden utilizar para desarrollar apps multiplataforma, y de algunos frameworks que pueden ser utilizados. Con el objetivo de presentar una forma alternativa de desarrollo de apps diferente del enfoque nativo, aunque las apps resultantes no alcancen el rendimiento, aspecto o funcionalidad de una app nativa.

9. Bibliografía.

Estudios de mercado en 2013:

[1] http://www.gartner.com/newsroom/id/2610015

[2]http://www.merca20.com/mercado-de-apps-para-dispositivos-moviles-seguira-en-crecimiento/

[3]http://www.tech-thoughts.net/2013/05/global-smartphone-market-share-trends-android-iphone-windows-phone.html#.UnqKaCeaZ-h

[4] http://www.gartner.com/newsroom/id/2592315

[5]http://techcrunch.com/2013/09/19/gartner-102b-app-store-downloads-globally-in-2013-26b-in-sales-17-from-in-app-purchases/

Descripción de tipo de Aplicaciones:

[6] http://www.infoq.com/author/Peter-Friese

Tutoriales de HTML5, CSS y JavaScript:

[7] http://www.w3schools.com/default.asp

[8] http://www.w3.org/2002/03/tutorials.html#webdesign_htmlcss

[9] https://marketplace.firefox.com/developers/docs/mobile_developers

Aplicación Ejemplo:

[10] http://tutsplus.com/authors/ronnie-swietek

Titanium:

[11] http://www.appcelerator.com/

[12] https://wiki.appcelerator.org/display/guides/Home

[13] *Building Mobile Applications with Titanium,* https://wiki.appcelerator.org/display/guides/BNAPP+ebook

Node.js:

[14]http://nodejs.org/

Git:

[15] http://git-scm.com/

PhoneGap:

[16] http://phonegap.com/

jQuery Mobile:

[17] http://jquerymobile.com/

[18] http://the-jquerymobile-tutorial.org/index.php

jQuery:

[19] http://jquery.com/

XAMPP:

[20] https://www.apachefriends.org/es/index.html

HTC Sync:

[21] http://www.htc.com/es/support/