UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

Escuela Técnica Superior de Ingeniería Informática
Universidad Politécnica de Valencia

Diseño e Implementación de una App para intercambio de contenido multimedia con una aplicación de gestión de proyectos

TRABAJO DE FIN DE GRADO
Grado en ingeniería informática

***Alumno*: Claudio Díaz Sapiña**
***Tutor*: Patricio Letelier Torres**

Curso 2015-2016

Resumen

El presente trabajo de fin de grado sirve para poder utilizar la herramienta de gestión de desarrollo software TUNE-UP desde el terminal móvil. Está desarrollado de manera que resulte intuitivo y de fácil aprendizaje para cualquier tipo de usuario no especializado en gestión de desarrollo software.

El proyecto comienza con una introducción a la telefonía móvil y a los principales sistemas operativos actuales. A continuación se expone la herramienta TUNE-UP con su definición y características. Seguidamente se introduce la programación con Android, mediante la cual se desarrollará la aplicación, y se presenta una aplicación sencilla de ejemplo.

Veremos la definición, ventajas y desventajas del protocolo utilizado por la aplicación para la comunicación con el servidor, además de las partes y la estructura de la aplicación desarrollada (Diagrama de clases, mapa de la aplicación, interfaces...)

Palabras clave: Herramientas para gestión de proyectos, APP

Abstract

This thesis serves to use the management tool software development TUNE-UP from the mobile terminal. It is developed in a way that is intuitive and easy to learn for any user not specialized in software development management.

The project begins with an introduction to mobile telephony and the major current operating systems. Then the TUNE-UP tool is exposed with its definition and features. Then the Android programming is introduced, by which the application will be developed, and a simple sample application is presented.

We will see the definition, advantages and disadvantages of the application protocol used for communication with the server, addition to the parts and the structure of the developed application (class diagram, map application, interfaces ...)

Key words: Project management tools, APP

Índice

Dedicatoria

Dedicado a todas aquellas personas de mi familia, y compañeros, que tanto me han apoyado y animado durante estos años de estudio.

Agradecimientos

Agradezco a , Carlos del Fresno, su inestimable ayuda y colaboración en la creación y modificación de los métodos en el servidor invocados por esta versión para Android, así como el soporte a la hora de buscar soluciones en momentos puntuales.

Agradecer también a D. Patricio Letelier Torres, tutor de este proyecto, por su ayuda, paciencia y orientación.

1. Introducción

1.1. Motivación y Objetivos

La telefonía móvil es un mercado en continua expansión. A medida que la tecnología va avanzando, los terminales que llevamos en nuestros bolsillos se van haciendo más y más potentes, hasta el punto de que algunos de estos terminales, ya superan de largo la potencia de los ordenadores que poblaban los escritorios de muchos hogares y empresas hace no más de 15 o 16 años. Tecnologías como la aceleración 3D, el disponer de 4 (o incluso 8) núcleos, las redes móviles, y un largo etcétera, han convertido estos pequeños aparatos en auténticos y completos ordenadores.

Pensando en facilitar, organizar, y agilizar las tareas de desarrollo, se han creado herramientas llamadas en general aplicaciones de gestión de proyectos, que permiten aumentar la eficiencia de los equipos de desarrollo, mediante la compartición y sincronización de tareas entre los integrantes del equipo, control y gestión de tiempos, etc.

Una de estas herramientas es TUNE-UP, que ha sido desarrollada por el equipo Atikass del Departamento de Sistemas y Computación de la Universidad Politécnica de Valencia, y que como se detallará posteriormente en el Capítulo 3 de esta memoria, permite realizar todas estas funciones.

El objetivo de este TFG es diseñar y desarrollar una versión Android para interactuar con la aplicación de escritorio, y añadir la funcionalidad de subir imágenes a las unidades de trabajo de TUNE-UP, permitiendo seleccionar el sitio de Tune-up y la unidad a la que se desea subir la imagen. La utilidad de una versión Android de esta aplicación es bien clara. Esto permitiría tomar imágenes en cualquier lugar con nuestro dispositivo móvil y cargar dicha imagen a nuestra unidad de trabajo (Bocetos en reuniones). Esto hace que no sea necesario tener las imágenes en el ordenador y poder tomarlas con la cámara de cualquier móvil y dónde quiera que sea necesario, dando una potencia y versatilidad extra a esta aplicación.

.

1.2. Organización de la memoria

A continuación se presenta la estructura del proyecto:

- Capítulo 1: Introducción al proyecto.
  ·Este capítulo se realiza una breve introducción del trabajo de fin de grado, motivación, objetivos y organización de la memoria.

- Capítulo 2: Desarrollo de Aplicaciones Móviles.
  ·Este capítulo se dedica a introducir brevemente la telefonía móvil en el contexto actúa.

- Capítulo 3: TUNE-UP.
  ·Este apartado explica cómo funciona la herramienta versionada para móvil en este proyecto.

- Capítulo 4: Desarrollo con Android.
  ·Este apartado muestra como instalar un entorno para desarrollar en Android y una pequeña aplicación de ejemplo.

- Capítulo 5: Aplicación móvil Tune-up.
  ·Este apartado explica los servicios REST utilizados, y los beneficios y desventajas de su uso.

- Capítulo 6: Estructura de la aplicación.
  ·Se muestra la estructura de la aplicación (mapa de la aplicación, diagrama de clases...).

- Capítulo 7: Pruebas.
  ·Aquí se explica brevemente las pruebas de aceptación realizadas.

- Capítulo 8: Conclusiones
  ·Pequeña explicación de las conclusiones de la realización del proyecto.

- Capítulo 9: Referencias.
  ·Listado de cada una de las referencias utilizadas a lo largo de la memoria.

- Capítulo 10: Anexos.
  ·Este capítulo incluye un manual de usuario de la aplicación y detalla las pruebas realizadas.

2. Desarrollo de Aplicaciones Móviles

En el momento de crear o adaptar una versión ya existente de un programa, se encuentra a menudo ante una difícil elección, ya que existen en el mercado actual varios sistemas operativos distintos para estos aparatos, cada uno de ellos, con sus pros y sus contras, su enfoque particular y destinatarios finales, y siempre en continua evolución.

Hacer una correcta elección o incluso decantarse por varias simultáneamente, es sin duda un primer paso hacia el éxito de la aplicación, ya que de no ser así, podemos encontrarnos con que la misma no llega al tipo de usuario que nosotros teníamos pensado, o incluso, en el peor de los casos, que la plataforma que hayamos escogido desaparezca del mercado (ya sea por ser demasiado vieja y obsoleta, o por ser demasiado nueva y que no tenga la aceptación esperada), y por tanto, nuestra aplicación muera con ella.

El material expuesto en este apartado ha sido extraído de las fuentes : [3],[6],[7], [8] y [9] (Apartado 9 Referencias).

2.1. Situación del Mercado Actual

Actualmente, existen en el mercado de la telefonía móvil 2 Sistemas Operativos mayoritarios, que acaparan prácticamente el total del mercado: Android e iOS, quedando por detrás Windows Phone con un 3% del mercado.

Como se puede observar en la Figura 2.1, que está actualizada a fecha de 8 de Noviembre de 2015 , la situación actual del mercado, está repartida de la siguiente forma:

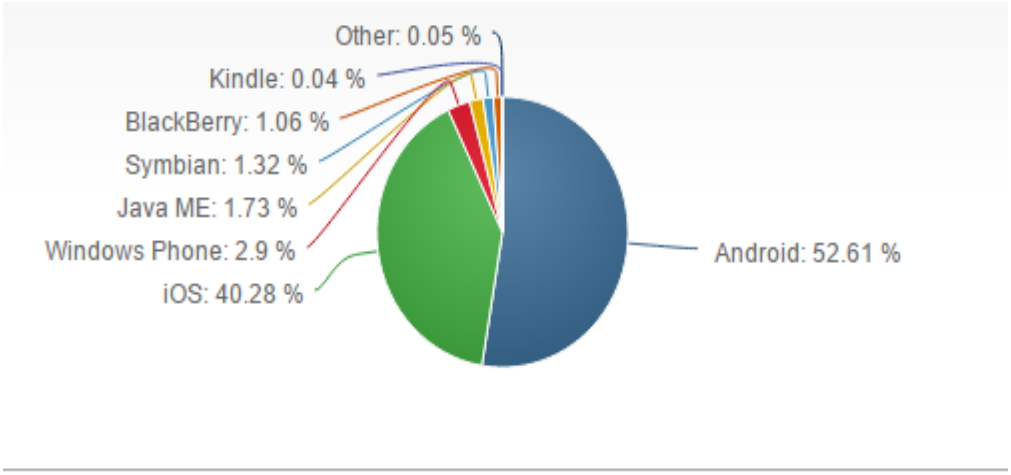


Figura 2.1 Situación mercado Sistemas Operativos

Como se puede apreciar en la figura 2.1, los sistemas operativos que predominan en el mercado son en primer lugar Android, con poco más de la mitad de mercado, y iOS con un 40%.

De todos estos sistemas operativos, hay un factor a tener en cuenta, y es que tanto Symbian, como iOS y RIM, son sistemas propietarios, es decir, que al no ser libres, sólo los fabricantes de estos smartphones pueden utilizarlos (Nokia, Sony, y unos cuantos fabricantes más que se detallarán a continuación para el caso de Symbian, Apple para iOS, y BlackBerry para el caso de RIM). A continuación pasaremos a enumerar las principales características e historia de los sistemas operativos más importantes del mercado.

iOS

De la mano de la veterana compañía Apple, y su interés por entrar en el mercado de la telefonía móvil, nació la primera versión de este sistema operativo que es el núcleo del primer iPhone (figura 2.4), lanzado el 29 de Junio de 2007. Estamos hablando por tanto de un sistema bastante más reciente que los comentados anteriormente, y pensado exclusivamente para su uso en dispositivos táctiles. El SDK de desarrollo, que no fue liberado hasta Marzo de 2008, se basa en Objective-C, y requiere el desembolso de una licencia de 99$ para su descarga y utilización. Si el desarrollador decide publicar la aplicación en el market de Apple, recibirá un 70% del importe que establezca.

Una de las características que lo hizo destacarse de sus competidores de la época, fue por el excelente uso de la tecnología multitouch, que permitía interactuar con el sistema mediante gestos como pellizcos, giros, etc. Como particularidad especial, cabe destacar de este sistema operativo que NO es compatible ni con Flash ni con Java, por motivos principalmente de licencias (aunque Steve Jobs, expresidente de la compañía, argumentaba que Flash no era lo suficientemente seguro, y que tenía un excesivo consumo de batería), lo cual perjudicaba enormemente la visualización de páginas web que utilicen estas tecnologías. No obstante, cuenta actualmente con soporte para el futuro HTML5, lo cual tratará de compensar estas carencias.

Figura 2.4 Primer dispositivo iPhone

Entre las características más importantes de su última versión (iOS 9.1), lanzada en Octubre de 2015, se encuentran las siguientes:

- **Múltiples aplicaciones** tales como : Asana, Slack, YouTube, Twitter, Instagram, Gmail, Hangouts, Chrome, Dark Sky, Amazon, SoundCloud, Spotify, and Xbox One Smartglass. All of them, including Chrome, have been behaving.
- **Gran velocidad** de procesamiento, así como interfaz de usuario rápida, fluida y estable.
- **Icloud Drive**: permite encontrar cualquier archivo guardado en iCloud.
- **Mail**: Sistema para adjuntar comentarios, dibujos, o hasta una firma a las fotos y documentos. Permite adjuntar archivos desde iCloud.
- **Maps** : En algunas ciudades permite ver todas las lineas y paradas del transporte público y mostrar una vista detallada del trayecto.
- **Quick Type** : Nueva barra de funciones rápidas, que permite dar formato a un texto, entre otras.

Windows Mobile

Este sistema operativo diseñado por Microsoft, y conocido actualmente en su última versión como Windows Phone, viene derivado del antiguo Windows CE, que era la versión del conocido sistema operativo para ordenadores de escritorio orientado a PDAs. Su primera versión orientada a smartphones, nació en el año 2003 bajo el nombre “Windows 2003 smartphone Edition”. Este es, de todos los sistemas anteriormente comentados junto con ANDROID, el único sistema operativo liberado a cualquier fabricante de telefonía móvil que quiera utilizarlo y que cumpla con los requisitos mínimos exigidos por su última versión, que son:

- Procesador Qualcomm Snapdragon S4 de doble núcleo o superior (hasta quad core)
- Mínimo 512 MB de RAM para teléfonos WVGA; mínimo 1 GB RAM para teléfonos 720p, WXGA o 1080p
- Mínimo 4GB de memoria interna
- GPS y A-GNSS mínimo; GLONASS depende de lo que decida el fabricante
- Soporte para microUSB 2.0
- Jack estéreo de 3,5mm para auriculares con soporte para tres botones
- Cámara trasera obligatoria, con autoenfoque y mínimo VGA (flash opcional)
- Acelerómetro, sensores de proximidad y luz, motor de vibración
- WiFi 802.11b/g y Bluetooth
- Hardware con soporte para DirecX
- Pantalla multitáctil capacitiva (mínimo cuatro puntos)

Esta disponibilidad de uso abierta a cualquier fabricante, debería en principio y en teoría, otorgarle una expansión superior al resto. No obstante, al menos de momento y tal y como se aprecia en el gráfico (Figura 2.1), no es así.

En cuanto al SDK, se pueden desarrollar dos tipos de aplicaciones para Windows Mobile: con código nativo o con código administrado. Llamamos código nativo al código C++ que utiliza directamente la API de Windows Mobile, y código administrado al que utiliza las clases del .NET Compact Framework con C# o VB.Net.

Android

Android es un sistema operativo basado en Linux que fue desarrollado por la empresa Android Inc (que fue comprada por Google en 2005). El lenguaje de programación utilizado por su SDK es principalmente Java, aunque tiene como opción la posibilidad de desarrollar en C++ utilizando el Android NDK (Native Development Kit). No obstante, Google no recomienda en ningún caso desarrollar una aplicación al 100% en este lenguaje, sino que se utilice junto al SDK para desarrollar pequeñas porciones de código que requieran un rendimiento crítico.

En cuanto a sus ventajas, nos encontramos ante un completo sistema operativo que, al estar basado en Linux, proporciona un alto nivel de abstracción y por tanto, un gran facilidad de acceso al hardware para el desarrollador. En la figura 2.5, podemos observar un diagrama del funcionamiento de este sistema operativo, que además cuenta con la ventaja de soporte completo de Flash (incluso en su última versión 10.2), y por supuesto Java, lo que le sitúa en una posición muy ventajosa frente al resto de sus rivales, que no soportan alguna (o incluso ninguna en algunos casos), de estas tecnologías ampliamente extendidas.

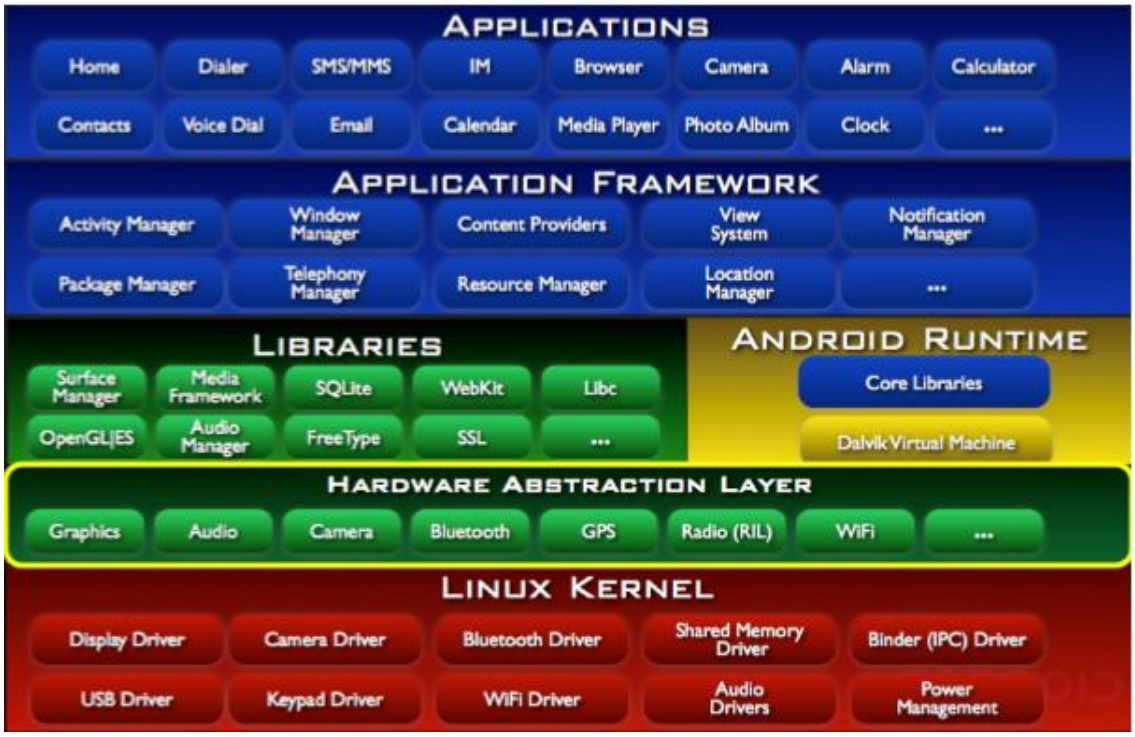


Figura 2.5 Diagrama de funcionamiento interno Android

Las características añadidas a este sistema operativo en su última versión para móviles (llamada Android 6.0 Marshmallow), que fue liberada el 5 de Octubre de 2015 son:

- Google Now on Tap: es la expansión de Google Now a todo nuestro dispositivo. Con una pulsación prolongada nos aparecerá una tarjeta con información referente a lo que está apareciendo en pantalla. Por ejemplo, si estamos leyendo un correo de un amigo que nos propone ir al cine a ver una película, al pulsar "Now on Tap" nos aparecerá la ficha de esa película.
- Soporte para huellas dactilares
- Android Pay
- Ahora Android realizará restauraciones y copias de seguridad de datos completas y automáticas de nuestras aplicaciones tras cambiar de dispositivo o tras restablecerlo de fábrica para continuar con todos nuestros datos y partidas
- Direct Share: una forma de compartir contenido más simplificada
- "Doze": nuevo sistema que intentará minimizar los wakelocks cuando el dispositivo no se está usando de forma activa
- Soporte oficial para tarjetas SD y USB
- Compatibilidad con lápices bluetooth
- Pantalla de bloqueo mejorada
- Controles de volumen simplificados
- Mejoras en el modo silencio y modo prioridad
- Opción experimental para modificar partes de la IU del sistema
- Direct Links: podemos vincular cada una de nuestras aplicaciones con direcciones URL, para que determinados enlaces siempre se abran con sus respectivas aplicaciones
- Explorador de archivos nativo

- Mejoras en el apartado de memoria RAM
- Mejoras en la selección de texto
- Soporte de Hotspot 2.0 versión 1
- Soporte para USB 3.1 Type-C
- Compatibilidad con pantallas 4K
- Administrador de permisos: Podemos decidir a qué permisos puede acceder cada aplicación, como los permisos de calendario, contactos, cámara, micrófono, SMS, sensores, teléfono y ubicación.

2.2. Generadores multiplataforma

Cada dispositivo móvil necesita una aplicación con un entorno de desarrollo distinto, un lenguaje de programación distinto. Eso obliga a hacer una gran inversión de aprendizaje de cada una de las tecnologías para abarcar el mayor número de dispositivos móviles.

Cuando queremos hacer una aplicación para los dispositivos iOS y para Android a la vez y no queremos programar dos veces, o simplemente sabemos Objetive-C pero no nos gusta Java o viceversa, la mejor solución es emplear frameworks multiplataforma, "write once, run everywhere".(Escríbelo una vez, ejecútalo dónde sea)

El desarrollo de aplicaciones multiplataforma implica reducción de costes y tiempo de desarrollo. La curva de aprendizaje de este tipo de frameworks de desarrollos es menor que el de desarrollos nativos. Las aplicaciones nativas son más caras porque requieren incremento del tiempo de desarrollo, cada software requiere un lenguaje de programación distinto.

Algunas herramientas para desarrollo multiplataforma:
Phonegap, Titanium Appcelerador y Ansca Corona, que nos permiten construir aplicaciones usando lenguajes como Javascript y Lua, pero estas requieren Mac y Xcode.

Un ejemplo; Phonegap:

Se puede emplear en 7 diferentes plataformas móviles (iPhone/iPad, Android, WebOS, W7, Black Berry, Palm y Symbian). Debido a que las funcionalidades disponibles para PhoneGap son más limitadas respecto a las disponibles en plataforma nativa, se requieren menos habilidades de implementación sobre aplicaciones móviles dentro del equipo de desarrollo.

Ventajas:

- Existe mucha documentación acerca de PhoneGap e incluso la propia web proporciona muchos ejemplos.
- Es fácil de desarrollar y proporciona gran libertad a los que tienen conocimientos de HTML y JavaScript.
- Es gratis, soporte de pago. Licencia BSD.

Inconvenientes:

- Dado que PhoneGap está diseñado para soportar diferentes plataformas, no contará con las características nuevas de cada plataforma tan pronto como estas se publiquen.
- En algunos casos es necesario usar el sistema operativo de la plataforma. Por ejemplo empaquetar aplicaciones Windows Phone sólo es posible con el sistema operativo Windows. Lo mismo ocurre con iOS, es necesario usar un Mac.
- Dependiendo de la plataforma se necesita un sistema diferente, para Android se requiere el uso de Eclipse y para iOS el uso de Xcode. Se desarrolla una aplicación híbrida, por lo que el rendimiento no es como el de una aplicación nativa.

3. TUNE-UP

En este capítulo se presenta la metodología que apoya el proceso de desarrollo, y que guarda todos los datos que posteriormente explotaremos para mostrar la información relevante en el cuadro de mandos.

El material presentado en este capítulo ha sido extraído de la siguiente fuente [1].

3.1 ¿Qué es TUNE-UP?

TUNE-UP es una metodología que incorpora aspectos ágiles y tradicionales con un sentido marcadamente pragmático. TUNE-UP se caracteriza fundamentalmente por combinar los siguientes elementos:

- **Modelo iterativo e incremental** para el desarrollo y mantenimiento del software. El trabajo se divide en unidades de trabajo que son asignadas a versiones del producto. Se realizan ciclos cortos de desarrollo, entre 3 y 6 semanas, dependiendo del producto.

- **Workflows flexibles** para la coordinación del trabajo asociado a cada unidad de trabajo. Los productos, según sus características, tienen disponibles un conjunto de workflows los cuales se asignan a cada una de las unidades de trabajo. Cada unidad de trabajo sigue el flujo de actividades del workflow para completarla. Bajo ciertas condiciones se permite saltar hacia adelante o hacia atrás en el workflow, así como cambios de agentes asignados e incluso cambio de workflow. Por ejemplo, las típicas situaciones de re-trabajo en desarrollo de software ocasionadas por detección de defectos se abordan con saltos atrás no explícitos en el workflow.

- **Proceso de desarrollo dirigido por las pruebas de aceptación (Test-Driven)**. La definición de una unidad de trabajo es básicamente la especificación de sus pruebas de aceptación acordadas con el cliente. A partir de allí, todo el proceso gira en torno a ellas, se estima el esfuerzo de implementar, diseñar y aplicar dichas pruebas, se diseñan e implementan y luego se aplican sobre el producto para garantizar el éxito de la implementación.

- **Planificación y seguimiento continuo centrados en la gestión del tiempo**. En todo momento debe estar actualizado el estado de las versiones, de las unidades de trabajo, y del trabajo asignado a los agentes. El jefe del proyecto puede actuar oportunamente con dicha información, tomando decisiones tales como: redistribuir carga de trabajo entre agentes, cambiar los plazos de la versión, mover unidades de trabajo entre versiones, etc.

- **Control de tiempos**. Los agentes registran el tiempo que dedican a la realización de las actividades, el cual se compara con los tiempos estimados en cada una de ellas, detectando oportunamente desviaciones significativas. Esto permite a los agentes gestionar más efectivamente su tiempo, mejorar sus estimaciones y ofrecer al jefe del proyecto información actualizada del estado de la versión.

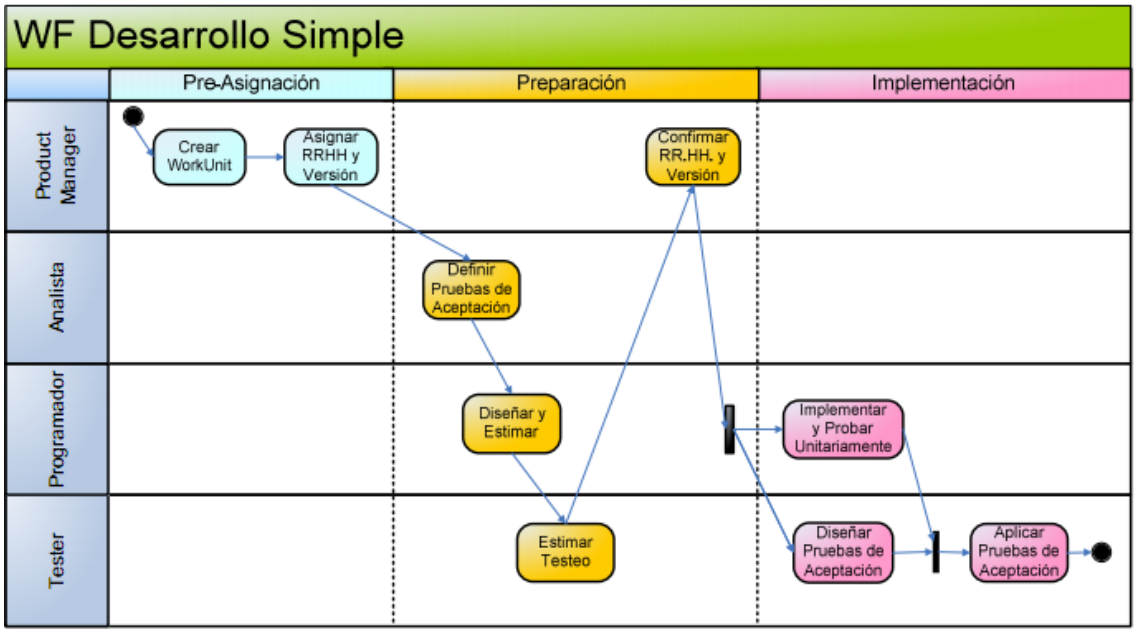


Figura 3.1: WF de desarrollo simple para unidades de trabajo

La Figura 3.1 ilustra un workflow simple para el desarrollo de una unidad de trabajo. Un workflow, en general, incluye actividades asociadas a tres fases del proceso:

- **Pre-Asignación**: actividades realizadas hasta la asignación de los RRHH y versión. La unidad de trabajo ha sido identificada pero aún no tiene una prioridad como para asignarle los RRHH y versión.

- **Preparación**: se pueden realizar cuando la unidad de trabajo ha alcanzado cierta prioridad y se le han asignado los RRHH y una versión. Se comienza a trabajar en la preparación de la unidad de trabajo, y debería concluirse antes del inicio de la versión objetivo en la cual se implementará. Incluye el análisis, las revisiones y estimaciones para su implementación.

- **Implementación**: se realizan durante la versión objetivo. Incluye la implementación, aplicación de pruebas e implantación. Las actividades de cada workflow pueden variar significativamente dependiendo de factores tales como la cantidad y especialización de agentes participantes, validaciones o negociaciones predeterminadas con el cliente, etc. Cada unidad de trabajo en una iteración del proyecto podría tener su propio workflow. Sin embargo, en la práctica basta con disponer de un reducido conjunto de workflows que permitan cubrir los tipos de unidades de trabajo que se presentan en el proyecto.

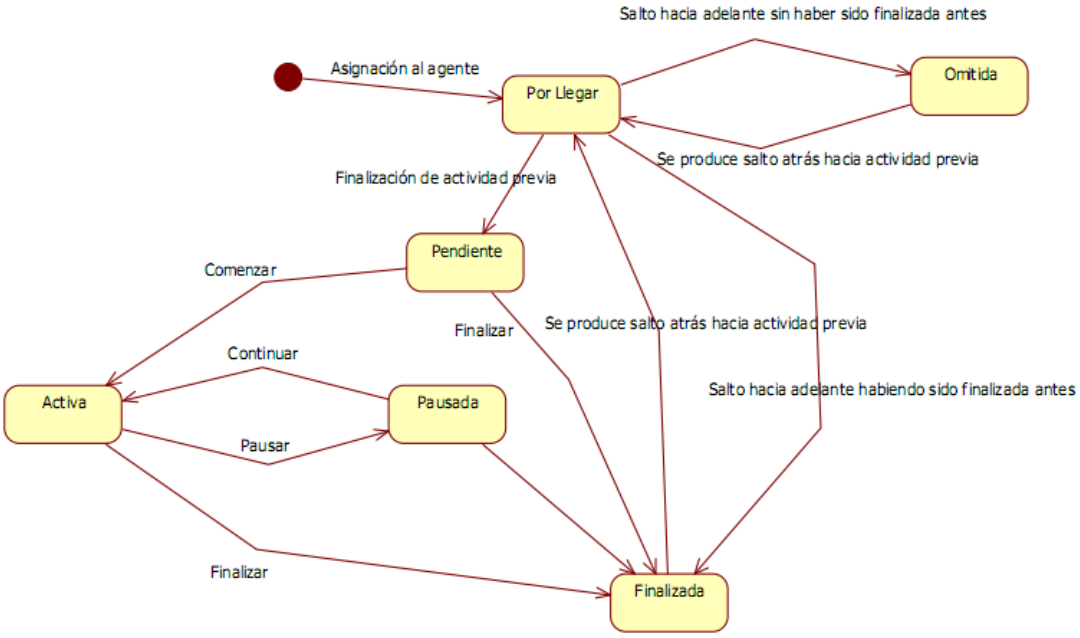


Figura 3.2: Posibles estados de una unidad de trabajo en una actividad

La Figura 3.2 muestra los posibles estados (Por Llegar, Pendiente, Activa, Pausada, Finalizada y Omitida) en los que se puede encontrar una actividad asociada a una unidad de trabajo.

3.2 TUNE-UP Process Tool

TUNE-UP Process Tool es una herramienta de apoyo para la aplicación efectiva de la metodología TUNE-UP. La herramienta está formada por tres módulos principales: Planificador Personal, Gestor de Unidades de Trabajo, y Planificador de Versiones. A continuación se describe cada uno de estos módulos.

Planificador Personal:

El Planificador Personal (PP) presenta el trabajo que tiene un agente. Cuando un agente inicia su jornada laboral, accede al PP (Figura 3.3) para ver el trabajo que tiene asignado actualmente, y seleccionar qué va a realizar de acuerdo a sus prioridades. El PP ofrece una variedad de facilidades de filtrado y ordenamiento de información, además de datos de tiempos, para que el agente pueda determinar su elección de acuerdo a sus prioridades. La tabla de la izquierda de la figura resume las contabilizaciones de las unidades de trabajo según la actividad y estado en el que se encuentran, y la tabla de la derecha, muestra información de dichas unidades de trabajo incluyendo: producto, versión, descripción de la unidad de trabajo, tiempos calculados, estado de la actividad actual dentro del workflow, etc.

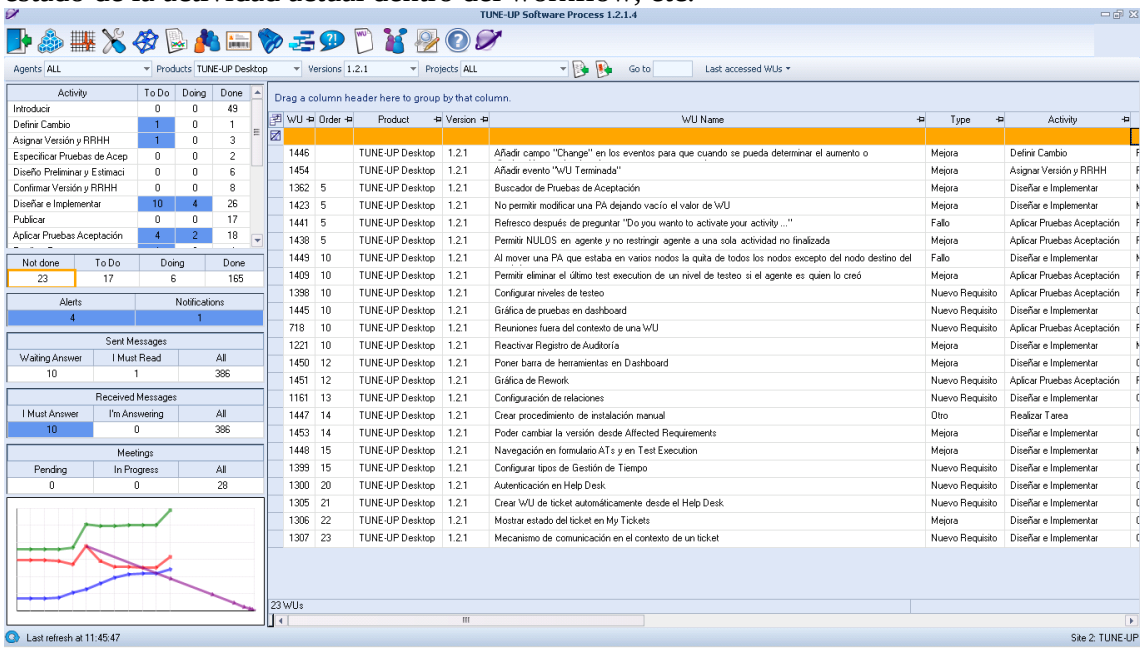


Figura 3.3: Planificador personal de TUNE-UP

Gestor de Unidades de Trabajo:

Cuando el agente decide la unidad de trabajo que va a realizar, accede con ella al Gestor de Unidades de Trabajo (GUT) (Figura 3.4) como apoyo esencial para realizar su tarea. En la parte superior se observan los datos generales de la unidad de trabajo, y en la parte inferior, un conjunto de pestañas con la funcionalidad que ofrece el GUT. Entre la funcionalidad disponible tenemos la posibilidad de activar/pausar/finalizar la actividad, mandar peticiones, añadir/consultar documentación asociada a la unidad de trabajo, ver los tiempos registrados en dicha unidad, etc.

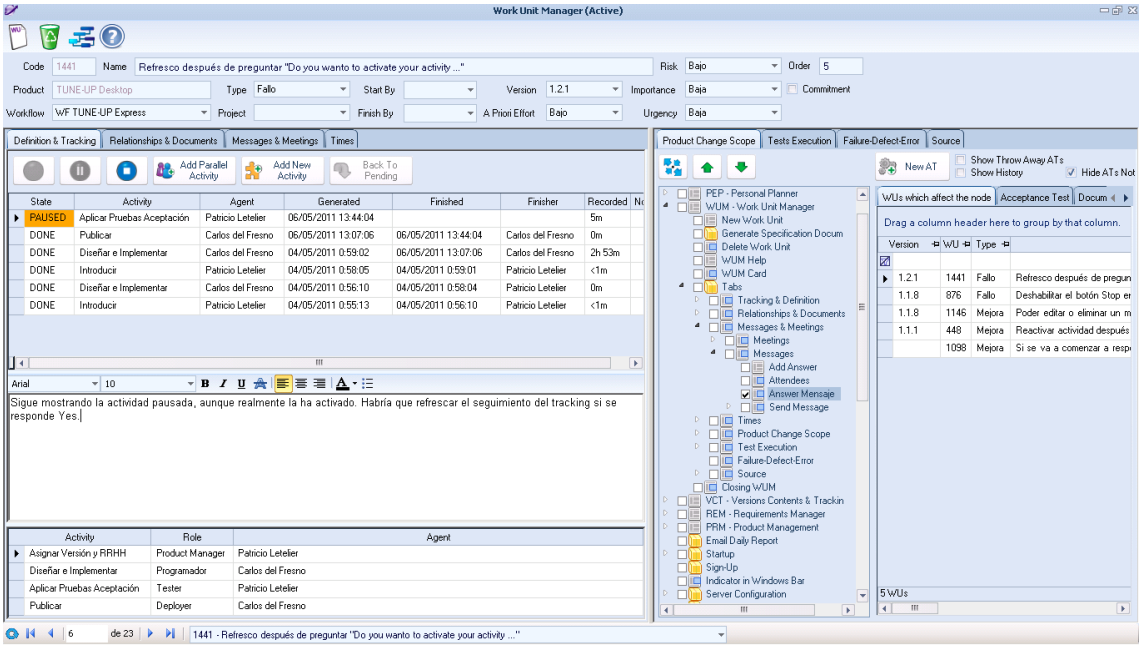


Figura 3.1: Gestor de unidades de trabajo

Planificador de Versiones:

El Planificador de Versiones (PV) (Figura 3.5.) permite gestionar los productos, sus versiones, los workflows disponibles para cada producto, los agentes por defecto asignados a las actividades de los workflows y realizar el seguimiento continuo del estado actual de las versiones.

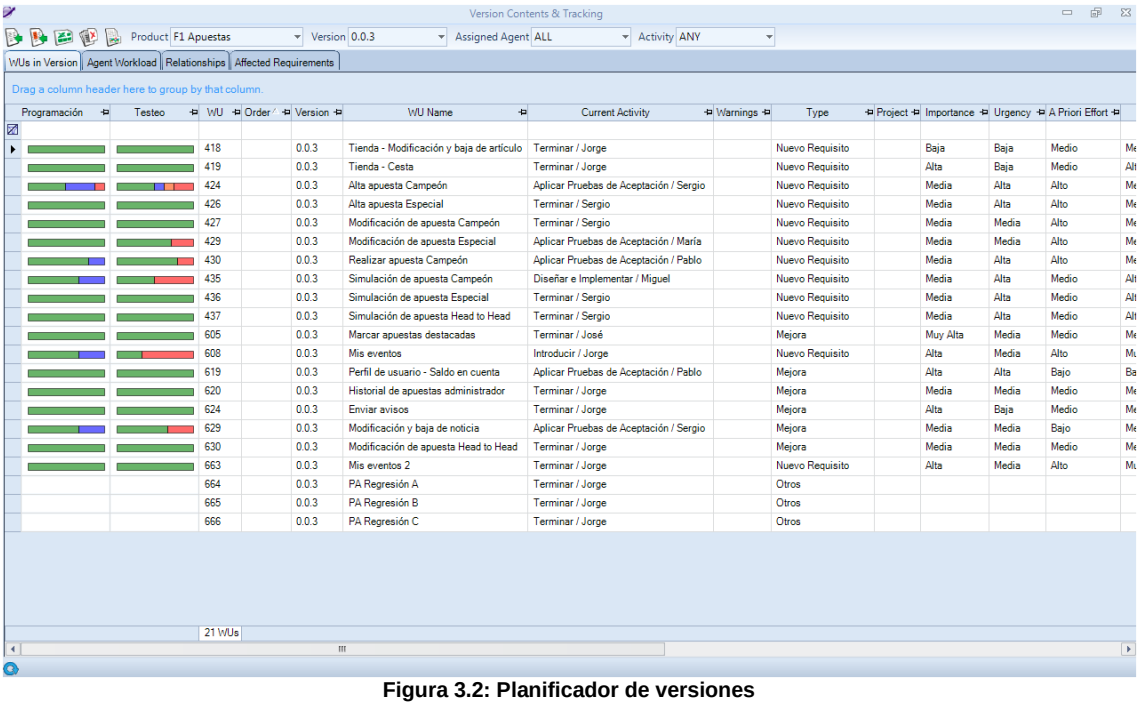


Figura 3.2: Planificador de versiones

4. Desarrollo con Android

En esta parte de la memoria pasaremos a explicar y detallar algunos aspectos esenciales de cómo funciona internamente este sistema operativo, para entender su lógica de funcionamiento tanto a nivel interno, como de cara al usuario. También detallaremos la instalación del SDK, así como una pequeña primera aplicación de ejemplo.

4.1 Entendiendo Android

Dos son las cuestiones fundamentales acerca del funcionamiento de este sistema operativo que deben detallarse: Lo que se entiende por el ciclo de vida de una aplicación, y el funcionamiento de la pila de aplicaciones abiertas de Android.

Antes de entrar en materia, cabe destacar que una aplicación de Android, puede estar compuesta por hasta 4 componentes básicos distintos (aunque también puede tener solo uno o varios de ellos), que son:

- **Activities**: Una Activity supone una única pantalla que contiene una interfaz de usuario. Sirve (como es obvio), para que el usuario pueda interactuar con nuestra aplicación.

- **Services**: Este componente hace referencia a lo que comúnmente se entiende por un servicio. Es decir, una parte del código que se ejecuta de manera transparente al usuario y normalmente en 2º plano, tanto si nuestra aplicación está abierta, como si no.

- **Content Providers**: Un Content Provider (o proveedor de contenidos), administra como su propio nombre indica una serie de datos. A través de un proveedor de contenidos, nuestra aplicación puede compartir datos con el resto de aplicaciones del sistema (si lo permitimos y lo definimos de esta manera), y guardarlos en el sistema de ficheros, en una base de datos SQLite, en la web, etc. Por ejemplo, invocando al proveedor de contenidos que almacena los contactos del teléfono, podemos recibir los mismos en nuestra aplicación, y mostrarlos, manipularlos, etc.

- **Broadcast Receiver**: Un receptor de mensajes sirve para que nuestra aplicación reciba diferentes mensajes que transmita el sistema operativo, y por tanto, pueda actuar en consecuencia. Estos mensajes pueden ser desde una notificación de que se ha extraído la tarjeta de memoria, a que se ha desactivado la WIFI, encendido la pantalla, perdido la cobertura del GPS…, etc.

Una vez aclarado esto, podemos pasar a detallar el funcionamiento de la pila de aplicaciones de Android. Para ello, es necesario mencionar una particularidad más de este sistema: todo terminal que lleve el sistema operativo Android, tendrá cuatro botones "físicos" (que pueden estar integrados dentro de la pantalla táctil), y que tienen las siguientes funciones:

- **HOME**: Al pulsar este botón, volvemos al escritorio principal, y cualquier aplicación que estuviéramos utilizando en ese momento, queda relegada a segundo plano (pero no se interrumpe su ejecución). Se puede pulsar en cualquier momento.

- **MENU**: Esta tecla tiene una función similar al botón derecho de un ordenador de escritorio. Pulsando menú, si la aplicación que está en ejecución en ese momento tiene definida alguna acción especial, mostrará diferentes opciones en función de su situación actual. Puede pulsarse en cualquier momento aunque si la aplicación no tiene definida ninguna acción especial, la pulsación se ignorará.

- **BACK**: Este es sin duda el botón clave que afectará al funcionamiento de la pila que vamos a detallar y que se muestra en la Figura 4.1. Al pulsar BACK, se destruye completamente la actividad que tengamos abierta en ese momento (recordemos que una aplicación puede estar compuesta de una o varias actividades, por lo que al pulsar BACK, no tiene por que cerrarse completamente la aplicación), y se vuelve a la actividad superior de la pila, que detallaremos posteriormente.

- **SEARCH**: Este es un simple botón de búsqueda que nos permitirá realizar búsquedas por Internet, inicialmente a través del buscador de Google.

Android dispone de una pila de aplicaciones, donde va almacenando todas las tareas (que recordemos que pueden estar compuestas de varias actividades como la del ejemplo), que el usuario va ejecutando y no cierra. Esta pila, es una cola LIFO, por lo que al destruir la actividad activa en ese momento, se rescatará la que esté en la cima de la pila y se pondrá de nuevo en ejecución.

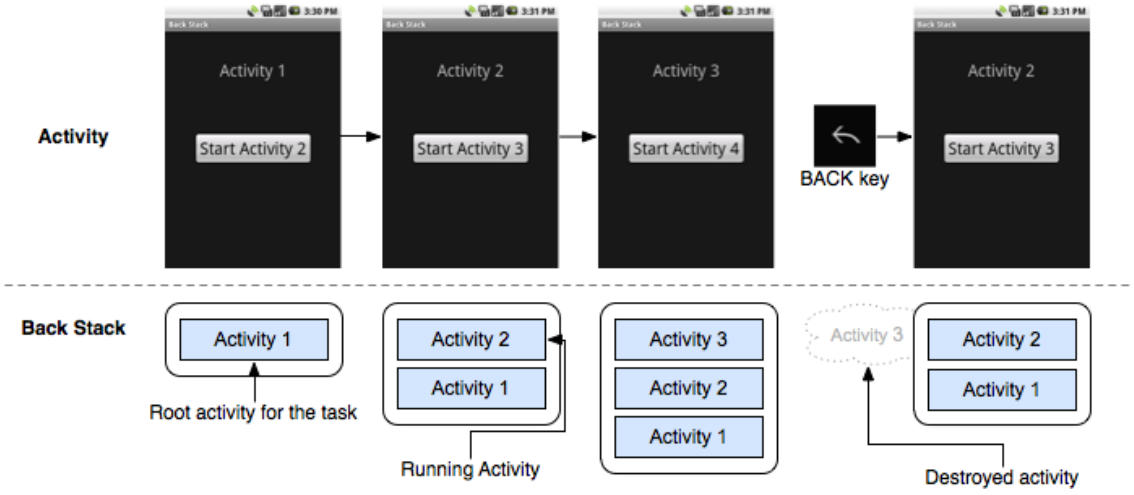


Figura 4.1 Funcionamiento de la pila de Android con una sola tarea.

Si tenemos varias tareas en ejecución (Figura 4.2), el funcionamiento es similar, solo que cada actividad se guarda dentro del espacio asignado en la pila a su tarea correspondiente, y al pulsar BACK, se vuelve a una actividad de esa misma tarea, lo cual evita la confusa situación que resultaría el cambiar a una actividad de una tarea totalmente diferente a la que estuviéramos ejecutando en ese momento. Por último, cabe mencionar que Android puede destruir automáticamente cualquier tarea que esté en la pila si el sistema llega a una situación de escasez de recursos, matando así el proceso que los ocupa y liberando los mismos para el sistema o apertura de nuevas aplicaciones que los requieren. Este proceso de destrucción automático se realiza siguiendo un protocolo de actuación que no merece la pena mencionar en este punto.

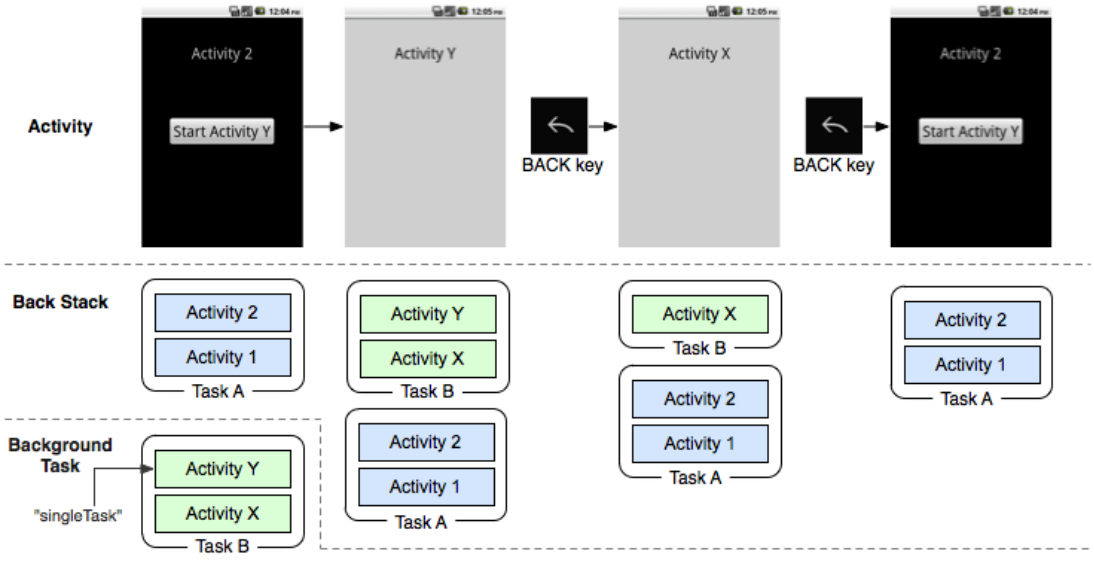


Figura 4.2 Funcionamiento de la pila Android con varias tareas.

En cuanto al ciclo de vida, es necesario observar el gráfico de la Figura 4.3. En ella, podemos observar los diferentes métodos por los que nuestra aplicación va pasando durante todo su ciclo de vida. De todos ellos, los más fundamentales y utilizados son los que detallaremos a continuación:

- **onCreate():** Este es el primer método que nuestra aplicación ejecutará. En este método colocaremos todo lo que queremos que nuestra aplicación realice en el momento de su creación. Normalmente, se suele utilizar para inicializar variables, definir el "layout" o plantilla que va a utilizar nuestra aplicación, recuperar los identificadores de los distintos botones, campos de texto, etc, para luego hacer uso de ellos en la aplicación, etc.

- **onResume()**: Toda actividad puede perder el foco en cualquier momento (bien sea porque el usuario ha pulsado la tecla HOME, se ha recibido una llamada de teléfono, una notificación, etc). Cuando ésta vuelve a tener el control se ejecutará onResume(). En este método por tanto, colocaremos todo aquello que queramos ejecutar cuando la aplicación vuelve a estar activa.

- **onPause()**: En este método colocaremos todas aquellas acciones que queramos realizar justo antes de que nuestra aplicación pierda el foco y pase a segundo plano. Aquí normalmente se suelen situar operaciones en las que se salvan determinados estados de la aplicación y que no queremos que se pierdan.

- **onDestroy()**: Cuando una aplicación finaliza, se ejecuta este método, por lo que aquí debemos colocar todas aquellas operaciones que queramos realizar justo antes de que nuestra aplicación finalice completamente (por ejemplo, finalizar un servicio asociado a la misma, etc).

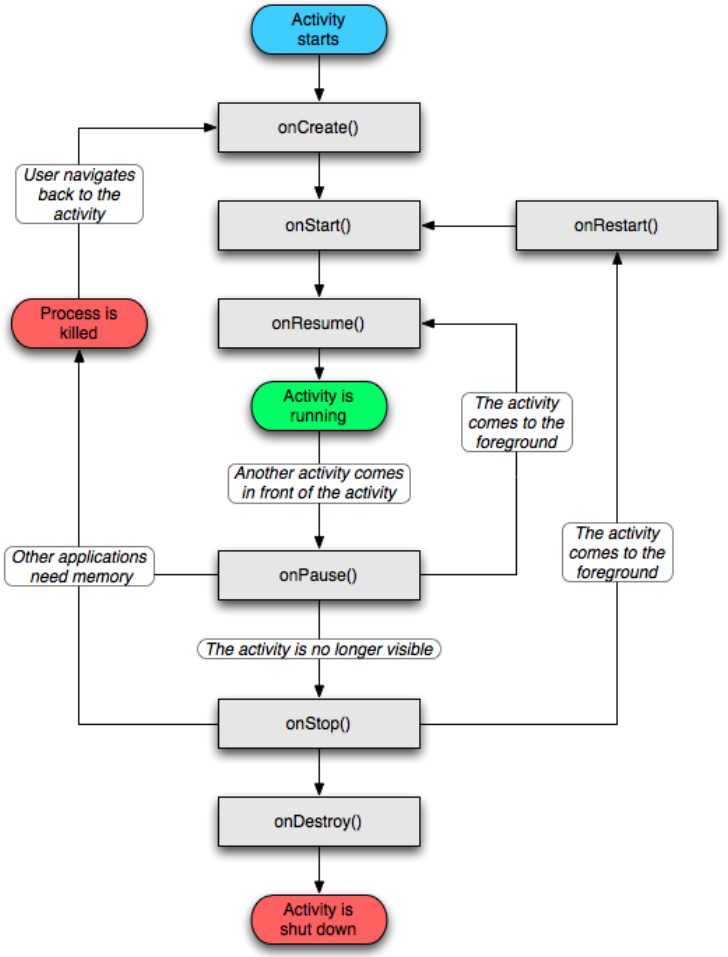


Figura 4.3 Ciclo de vida de una aplicación Android

4.2 Instalando el SDK de Android

Una vez entendidos todos estos aspectos del sistema operativo, pasaremos a instalar su Kit de desarrollo, el cual nos permitirá empezar a desarrollar y crear así nuestra primera aplicación.

Este tutorial de instalación del SDK y desarrollo de la primera aplicación que viene a continuación, ha sido obtenido desde [2]

Paso 1. Descarga e instalación de Eclipse:

Si aún no tienes instalado Eclipse, puedes descargar la versión 3.5 desde http://www.eclipse.org/downloads/packages/release/galileo/sr2 Recomiendo descargar por ejemplo la versión *Eclipse IDE for Java Developers*. La instalación consiste simplemente en descomprimir el zip en la ubicación deseada.

Paso 2. Descargar el SDK de Android:

El SDK de la plataforma Android se puede descargar desde http://developer.android.com/sdk/index.html. Una vez descargado, de nuevo bastará con descomprimir el zip en cualquier ubicación.

Paso 3. Descargar el plugin Android para Eclipse:

Google pone a disposición de los desarrolladores un plugin para Eclipse llamado *Android Development Tools* (ADT) que facilita en gran medida el desarrollo de aplicaciones para la plataforma. Podéis descargarlo mediante las opciones de actualización de Eclipse, accediendo al menú "*Help / Install new software…*" e indicando la URL de descarga "https://dl-ssl.google.com/android/eclipse/". Se debe seleccionar e instalar el paquete completo *Developer Tools*, formado por *Android DDMS* y *Android Development Tools*.

Paso 4. Configurar el plugin ADT:

En la ventana de configuración de Eclipse (Figura 4.4), se debe acceder a la sección de Android e indicar la ruta en la que se ha descomprimido el SDK (paso 2).

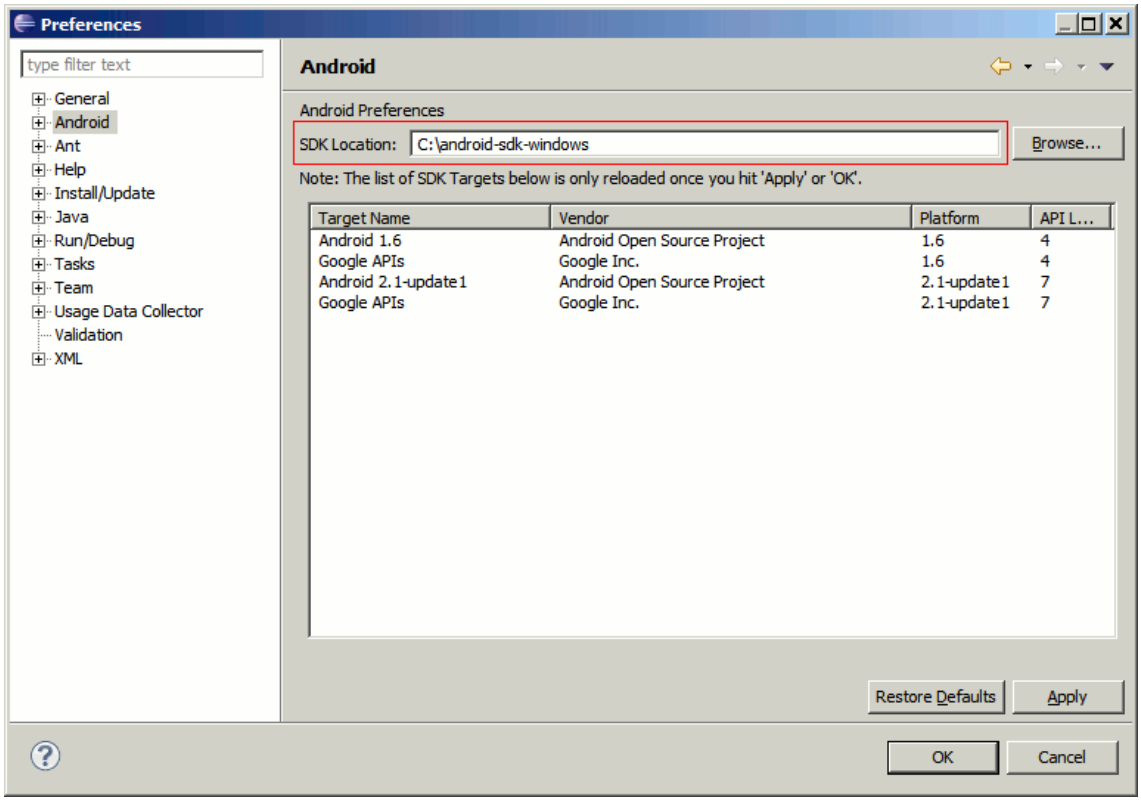


Figura 4.4 Apartado Android de la ventana de configuración de Eclipse

Paso 5. Descargar los *targets* necesarios:

Además del SDK de Android comentado en el paso 2, también debemos descargar los llamados *SDK Targets* de Android (Figura 4.5), que no son más que las librerías necesarias para desarrollar en cada una de las versiones concretas de Android. Así, si queremos desarrollar por ejemplo para Android 1.6 tendremos que descargar su *target* correspondiente. Para ello, desde Eclipse debemos acceder al menú "*Window/Android SDK and AVD Manager*", y en la sección *Available Packages* seleccionar e instalar todos los paquetes deseados.

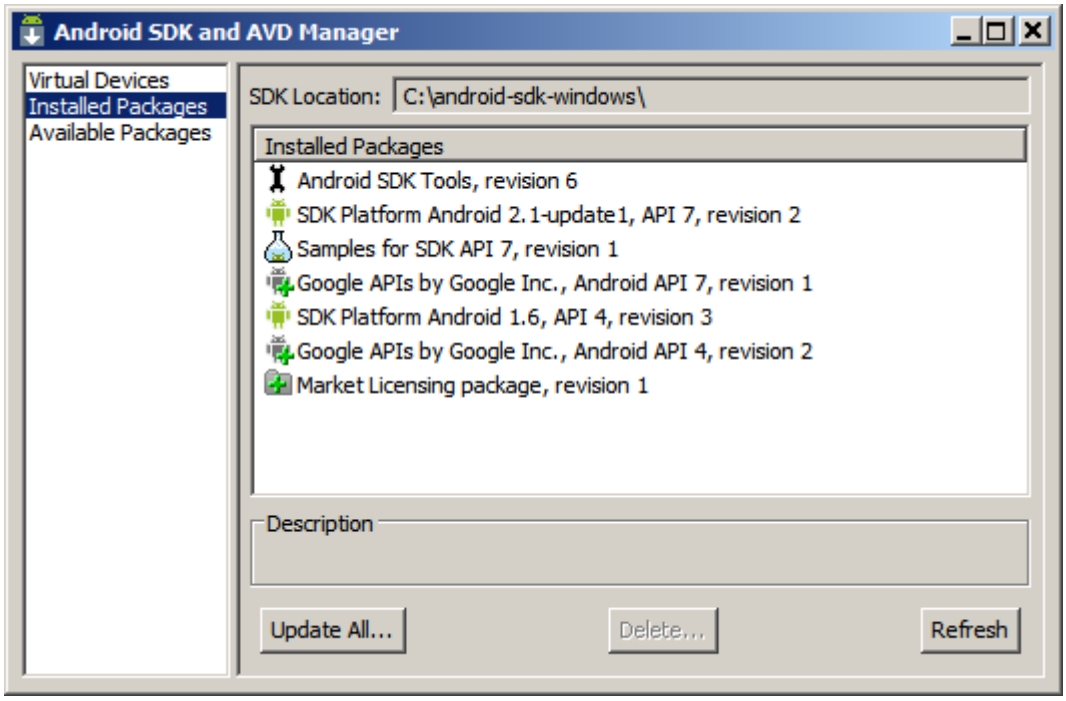


Figura 4.5 Instalación y configuración de los Targets

Paso 6. Configurar un AVD:

A la hora de probar y depurar aplicaciones Android no tendremos que hacerlo necesariamente sobre un dispositivo físico, sino que podremos configurar un emulador o dispositivo virtual (*Android Virtual Device*, o AVD) donde poder realizar fácilmente estas tareas (Figura 4.6). Para ello, volveremos a acceder al *AVD Manager*, y en la sección *Virtual Devices* podremos añadir tantos AVD como se necesiten (por ejemplo, configurados para distintas versiones de Android).

Para configurar el AVD tan sólo tendremos que indicar un nombre descriptivo, el target de Android que utilizará, y las características de hardware del dispositivo virtual, como por ejemplo su resolución de pantalla, el tamaño de la tarjeta SD, o la disponibilidad de GPS.

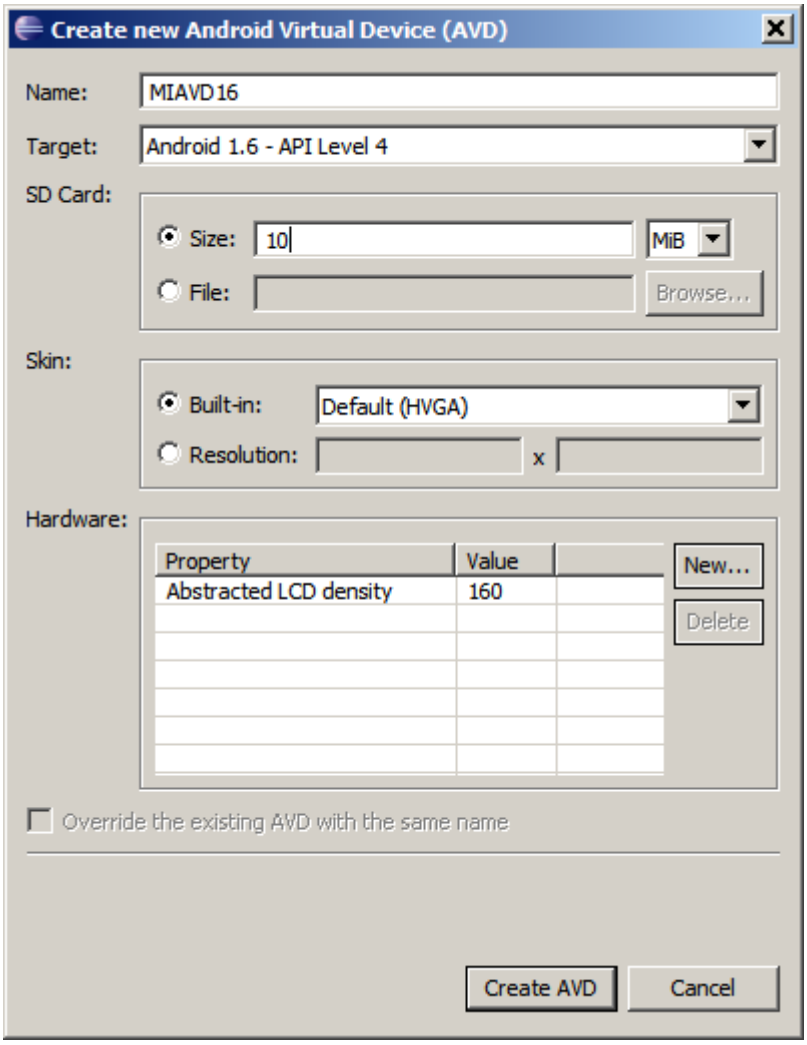


Figura 4.6 Creación de un Android Virtual Device

4.3 Desarrollo de una aplicación de Ejemplo

Después de instalar nuestro entorno de desarrollo para Android y comentar la estructura básica de un proyecto y los diferentes componentes software que podemos utilizar ya es hora de empezar a escribir algo de código. Empezaremos por escribir una aplicación sencilla.

La aplicación constará de dos pantallas (Figura 4.7). Por un lado la pantalla principal, posteriormente la pantalla de login de la aplicación, que solicitará un nombre al usuario y contraseña y una segunda pantalla en la que se mostrará un mensaje de saludo al usuario. Sencillo, pero aprenderemos muchos conceptos básicos, que para empezar no está mal.

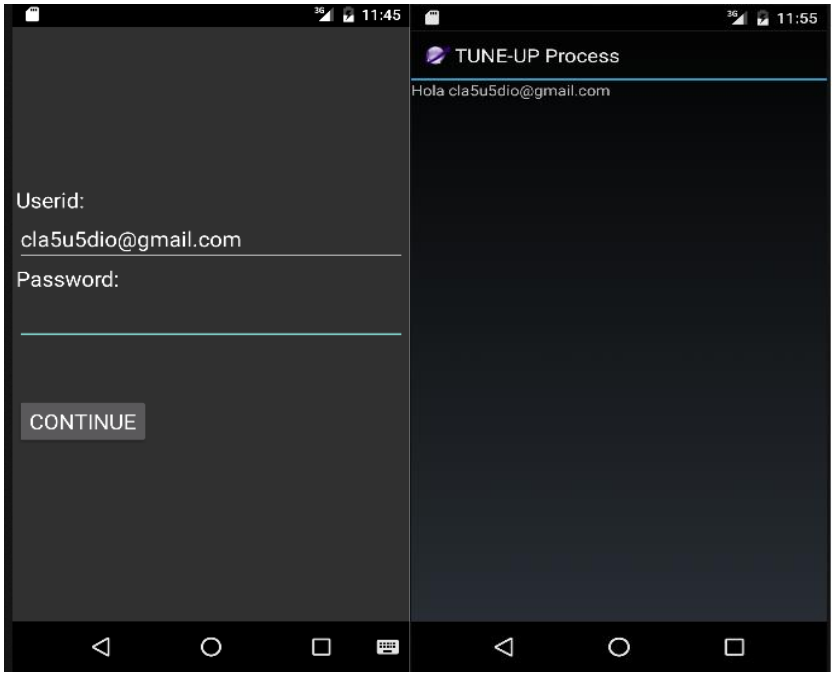


Figura 4.7 Primera aplicación de ejemplo

En primer lugar vamos a crear un nuevo proyecto Android (Figura 4.8). Llamaremos al proyecto "*HolaUsuario*", daremos un nombre a la aplicación e indicaremos que se cree una actividad llamada "*HolaUsuario*".

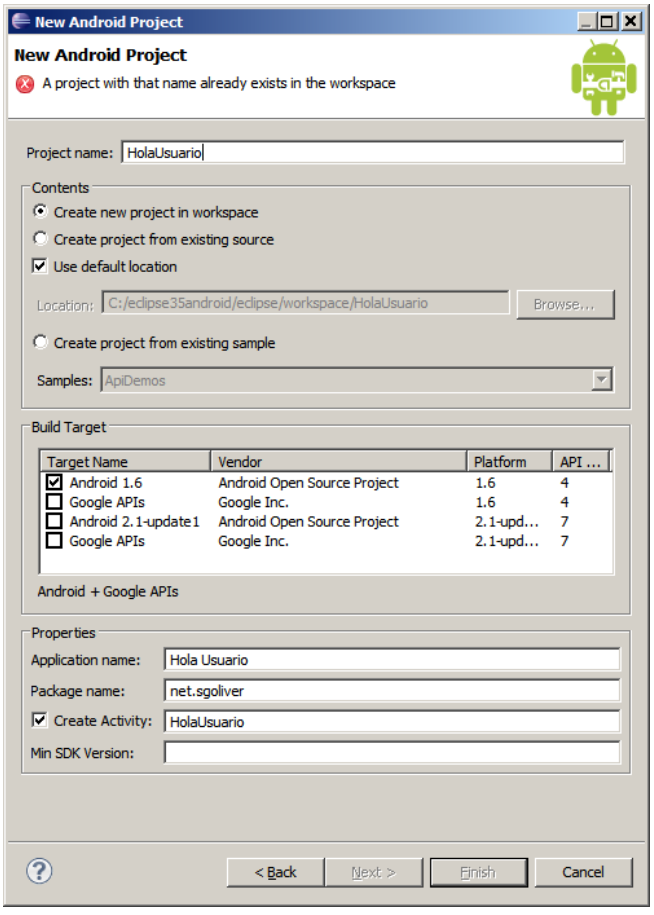


Figura 4.8 Pantalla de creación de un proyecto

Esto nos crea la estructura de carpetas del proyecto y todos los ficheros necesarios de un *Hola Mundo* básico, es decir, una sola pantalla donde se muestra únicamente un mensaje fijo. Lo primero que vamos a hacer es diseñar nuestra pantalla principal modificando la que Eclipse nos ha creado por defecto. ¿Pero dónde y cómo se define cada pantalla de la aplicación? En Android, el diseño y la lógica de una pantalla están separados en dos ficheros distintos. Por un lado, en el fichero /res/layout/main.xml tendremos el diseño puramente visual de la pantalla definido como fichero XML y por otro lado, en el fichero /src/paquetejava/HolaUsuario.java, encontraremos el código java que determina la implementación del código correspondiente a la pantalla.

Vamos a modificar en primer lugar el aspecto de la ventana principal de la aplicación añadiendo los controles (views) que vemos en la primera captura de pantalla. Para ello, vamos a sustituir el contenido del fichero main.xml por el siguiente:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent" >
    <TextView android:text="@string/user"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content" />
    <EditText android:id="@+id/TxtPass"
    android:layout_height="wrap_content"
    android:layout_width="fill_parent" />
    <TextView android:text="@string/password"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content" />
    <EditText android:id="@+id/TxtNombre"
    android:layout_height="wrap_content"
    android:layout_width="fill_parent" />
    <Button android:id="@+id/BtnLogin"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/login" />
</LinearLayout>
```

En este XML se definen los elementos visuales que componen la interfaz de nuestra pantalla principal y se especifican todas sus propiedades. Expliquemos un poco lo que vemos en el fichero:

Lo primero que nos encontramos es un elemento LinearLayout. Los *layout* son elementos no visibles que determinan cómo se van a distribuir en el espacio los controles que incluyamos en su interior. En este caso, un LinearLayout distribuirá los controles uno tras otro y en la orientación que indique su propiedad android:orientation.

Dentro del layout hemos incluido 5 controles: dos etiquetas (TextView), dos cuadros de texto (EditText), y un botón (Button). En todos ellos hemos establecido las siguientes propiedades:

- android:id. ID del control, con el que podremos identificarlo más tarde en nuestro código. Vemos que el identificador lo escribimos precedido de "@+id/". Esto tendrá como efecto que al compilarse el proyecto se genere automáticamente una nueva constante en la clase R para dicho control.

- android:text. Texto del control. El texto de un control se puede especificar directamente o bien utilizar alguna de las cadenas de texto definidas en los recursos del proyecto (fichero *strings.xml*), en cuyo caso indicaremos su identificador precedido del prefijo "@string/".
- android:layout_height y android:layout_width. Dimensiones del control con respecto al layout que lo contiene. Esta propiedad tomará normalmente los valores "wrap_content" para indicar que las dimensiones del control se ajustarán al contenido del mismo, o bien "fill_parent" para indicar que el ancho o el alto del control se ajustará al ancho o alto del layout contenedor respectivamente.

Con esto ya tenemos definida la presentación visual de nuestra ventana principal de la aplicación. De igual forma definiremos la interfaz de la segunda pantalla, creando un nuevo fichero llamado *frmmensaje.xml*, y añadiendo esta vez tan solo una etiqueta (TextView) para mostrar el mensaje personalizado al usuario. Veamos cómo quedaría nuestra segunda pantalla:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
android:layout_width="wrap_content"
android:layout_height="wrap_content">

    <TextView android:id="@+id/TxtMensaje"
    android:layout_height="wrap_content"
    android:layout_width="fill_parent"
    android:text="$mensaje"></TextView>
</LinearLayout>
```

Una vez definida la interfaz de las pantallas de la aplicación deberemos implementar el código de la misma. Como ya hemos comentado, el código de la aplicación se definirá en ficheros java Independientes (una clase activity por cada pantalla de la aplicación). Para la pantalla principal ya tenemos creado un fichero por defecto llamado *HolaUsuario.java*. Empecemos por comentar su código por defecto:

```
public class HolaUsuario extends Activity {
/** Called when the activity is first created. */
@Override
public void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.main);
}
    }
```

Como ya vimos en un apartado anterior, las diferentes pantallas de una aplicación Android se definen mediante objetos de tipo Activity. Por tanto, lo primero que encontramos en nuestro fichero java es la definición de una nueva clase HolaUsuario que extiende a Activity. El único método que sobre escribiremos de esta clase será el método OnCreate, llamado cuando se crea por primera vez la actividad. En este método lo único que encontramos en principio, además de la llamada a su implementación en la clase padre, es la llamada al método setContentView(R.layout.main). Con esta llamada estaremos indicando a Android que debe establecer como interfaz gráfica de esta actividad la definida en el recurso R.layout.main, que no es más que la que hemos especificado en el fichero /res/layout/main.xml. La clase R es una clase que se genera de manera automática al compilar el proyecto, y que asigna a cada recurso (una imagen, un layout de una activity, un color, etc), una constante numérica. En principio vamos a

crear una nueva actividad para la segunda pantalla de la aplicación análoga a ésta primera, para lo que crearemos una nueva clase FrmMensaje que extienda de Activity y que implemente el método onCreate indicando que utilice la interfaz definida en R.layout.frmmensaje.

```
public class FrmMensaje extends Activity {
@Override
public void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.frmmensaje);
}
    }
```

Como vemos, el código incluido por defecto en estas clases lo único que hace es generar la interfaz de la actividad. A partir de aquí nosotros tendremos que incluir el resto de la lógica de la aplicación. Y vamos a empezar con la actividad principal HolaUsuario, obteniendo una referencia a los diferentes controles de la interfaz que necesitemos manipular, en nuestro caso sólo los cuadros de texto y el botón.

Para ello utilizaremos el método findViewById() de Android, indicando el Id de cada control, definidos como siempre en la clase R:

```
final EditText txtNombre = (EditText)findViewById(R.id.TxtNombre);
final EditText txtPass = (EditText)findViewById(R.id.TxtPass);
final Button btnLogin = (Button)findViewById(R.id.BtnLogin);
```

Una vez tenemos acceso a los diferentes controles, ya sólo nos queda implementar las acciones a tomar cuando pulsemos el botón de la pantalla. Para ello implementaremos el evento onClick de dicho botón, veamos cómo:

```
btnHola.setOnClickListener(new OnClickListener() {
@Override
public void onClick(View arg0) {
Intent intent = new Intent(HolaUsuario.this, FrmMensaje.class);
Bundle bundle = new Bundle();
bundle.putString("NOMBRE", txtNombre.getText().toString());
bundle.putString("PASSWORD", txtPass.getText().toString());
intent.putExtras(bundle);
startActivity(intent);
}
    });
```

La comunicación entre los distintos componentes y aplicaciones en Android se realiza mediante *intents*, por lo que el primer paso será crear un objeto de este tipo. Existen varias variantes del constructor de la clase Intent, cada una de ellas dirigida a unas determinadas acciones, pero en nuestro caso particular vamos a utilizar el *intent* para llamar a una actividad desde otra de la misma aplicación, para lo que pasaremos al constructor una referencia a la propia actividad llamadora (HolaUsuario.this), y la clase de la actividad llamada (FrmMensaje.class).

Si quisiéramos tan sólo mostrar una nueva actividad ya tan sólo nos quedaría llamar a startActivity() pasándole como parámetro el *intent* creado. Pero en nuestro ejemplo queremos también pasarle cierta información a la actividad, concretamente el nombre y el password que introduzca el usuario en el cuadro de texto. Para hacer esto vamos a crear un objeto Bundle, que puede contener una lista de pares clave-valor con toda la información a pasar entre las actividades. En nuestro caso sólo añadiremos dos datos de

tipo String mediante el método putString(*clave*, *valor*). Tras esto añadiremos la información al *intent* mediante el método putExtras(bundle).

Finalizada la actividad principal de la aplicación pasamos ya a la secundaria. Comenzaremos de forma análoga a la anterior, ampliando el método onCreate obteniendo las referencias a los objetos que manipularemos, esta vez sólo la etiqueta de texto. Tras esto viene lo más interesante, debemos recuperar la información pasada desde la actividad principal y asignarla como texto de la etiqueta. Para ello accederemos en primer lugar al *intent* que ha originado la actividad actual mediante el método getIntent() y recuperaremos su información asociada (objeto Bundle) mediante el método getExtras().

Hecho esto tan sólo nos queda construir el texto de la etiqueta mediante su método setText(texto) y recuperando el valor de nuestra clave almacenada en el objeto Bundle mediante getString(clave).

```
public class FrmMensaje extends Activity {
@Override
public void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.frmmensaje);
TextView txtMensaje = (TextView)findViewById(R.id.TxtMensaje);
Bundle bundle = getIntent().getExtras();
txtMensaje.setText("Hola " + bundle.getString("NOMBRE"));
}
    }
```

Con esto hemos concluido el código de las dos pantallas de nuestra aplicación y tan sólo nos queda un paso importante para finalizar nuestro desarrollo. Como indicamos en uno de los artículos anteriores, cualquier aplicación Android utiliza un fichero especial en formato XML (*AndroidManifest.xml*) para definir, entre otras cosas, los diferentes elementos que la componen.

Por tanto, todas las actividades de nuestra aplicación deben quedar convenientemente recogidas en este fichero. La actividad principal ya debe aparecer puesto que se creó de forma automática al crear el nuevo proyecto Android, por lo que debemos añadir tan sólo la segunda. Para este ejemplo nos limitaremos a incluir la actividad en el XML, más adelante veremos que opciones adicionales podemos especificar.[4]

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="net.sgoliver"
    android:versionCode="1"
    android:versionName="1.0">
    <application android:icon="@drawable/icon" android:label="@string/app_name">
    <activity android:name=".HolaUsuario"
    android:label="@string/app_name">
    <intent-filter>
    <action android:name="android.intent.action.MAIN" />
    <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
    </activity>
    <activity android:name=".FrmMensaje"></activity>
    </application>
```

```
    <uses-sdk android:minSdkVersion="4" />
</manifest>
```

5.Aplicación TUNE-UP móvil.

5.1 Introducción.

Se ha tomado como base la aplicación móvil ya existente del alumno Luis Alejandro Ruiz Ramírez, ya que ésta aplicación, aunque anticuada, tendrá la misma funcionalidad añadiendo la posibilidad de enviar imágenes a la UT deseada desde la galería o desde la cámara y con una interfaz actualizada a las posibilidades que ofrece android actualmente.

Además, la aplicación realizaba las llamadas a los servicios web mediante el protocolo SOAP, y en esta aplicación se ha realizado la sustitución de dicho protocolo por el protocolo REST, que será explicado a continuación junto a sus ventajas y desventajas.

La información expuesta en el siguiente apartado ha sido extraída de la fuente [5] y [11].

5.2 SERVICIOS REST.

Según la definición puramente teórica, REST es un estilo de arquitectura que abstrae los elementos de dicha arquitectura dentro de un sistema hypermedia distribuido . Pero esto de por sí no nos dice demasiado a no ser que nos dediquemos al estudio teórico de este tipo de cosas. Utilizando palabras llanas, podríamos decir que REST es un conjunto de principios, o maneras de hacer las cosas, que define la interacción entre distintos componentes, es decir, las reglas que dichos componentes tienen que seguir. El protocolo más usado que cumple esta definición, es el protocolo HTTP.

Esto quiere decir, por extensión, que toda aplicación web bajo el protocolo HTTP es a su vez una aplicación REST. Sin embargo, como veremos más abajo, eso no implica en absoluto que todas las aplicaciones web sean servicios web RESTful, ya que estas tienen que cumplir una serie de requisitos para ser consideradas tales. Existen, además, otros tipos de métodos para implementar servicios web, que seguramente te suenen, como RPC, SOAP o WSDL. Sin embargo, el uso de dichos mecanismos no se suele recomendar en favor de RESTful, ya que RESTful es mucho más fácil de entender e implementar. No obstante, como siempre, la decisión sobre implementar una tecnología u otra depende en gran medida de las características del proyecto en el que estemos implicados, por lo que es recomendable siempre hacer un análisis concienzudo del proyecto y las tecnologías disponibles para decantarnos por una o por otra. A continuación continuo con más detalles sobre REST que te pueden ayudar a elegir esta metodología por encima de otras.

5.2.1. Reglas de la arquitectura REST

REST define una serie de reglas que toda aplicación que pretenda llamarse REST debe cumplir. Como veremos, estas reglas se nos dan ya dadas si vamos a usar el protocolo HTTP, en cualquiera de sus implementaciones (por ejemplo, Apache). No obstante, conviene repasar estos conceptos para tener una visión más profunda, y veremos más adelante que nos ayudará bastante a la hora de implementar nuestros programas:

Arquitectura cliente-servidor: consiste en una separación clara y concisa entre los 2 agentes básicos en un intercambio de información: el cliente y el servidor. Estos 2 agentes deben ser independientes entre sí, lo que permite una flexibilidad muy alta en todos los sentidos.

Stateless: esto significa que nuestro servidor no tiene porqué almacenar datos del cliente para mantener un estado del mismo. Esta limitación es sujeto de mucho debate en la industria, incluso ya empiezan a usarse tecnologías relacionadas que implementan el estado dentro de la arquitectura, como WebSockets. Como sabemos, HTTP también cumple esta norma, por lo que estamos acostumbrados ya a hacer uso de protocolos stateless.

Cacheable: esta norma implica que el servidor que sirve las peticiones del cliente debe definir algún modo de cachear dichas peticiones, para aumentar el rendimiento, escalabilidad, etc. Una vez más, HTTP implementa esto con la cabecera "Cache-control", que dispone de varios parámetros para controlar la cacheabilidad de las respuestas.

Sistema por capas: nuestro sistema no debe forzar al cliente a saber por qué capas se tramita la información, lo que permite que el cliente conserve su independencia con respecto a dichas capas.

Interfaz uniforme: esta regla simplifica el protocolo y aumenta la escalabilidad y rendimiento del sistema. No queremos que la interfaz de comunicación entre un cliente y el servidor dependa del servidor al que estamos haciendo las peticiones, ni mucho menos del cliente, por lo que esta regla nos garantiza que no importa quien haga las peticiones ni quien las reciba, siempre y cuando ambos cumplan una interfaz definida de antemano.

· Para que una aplicación sea REST al 100%, tendrá que implementar 4 principios básicos, y pondremos esto en relación a cómo HTTP implementa dichos principios:

Identificación de recursos: toda aplicación REST debe poder identificar sus recursos de manera uniforme. HTTP implementa esto usando las llamadas URIs (Uniform resource identifier). Esta es la URL que usamos tradicionalmente, y aunque hay una diferencia sutil entre URLs y URIs , diremos que toda URL es una URI a su vez. Esta identificación del recurso no incluye la representación del mismo, cosa que veremos a continuación.

Recursos y representaciones: visto que todo recurso debe tener una identificación (URI), REST define también la manera en que podemos interactuar con la representación del mismo, ya sea para editarlo o borrarlo, directamente del servidor. Estas representaciones se dejan a instancias de la implementación final del programa, pero HTTP define distintas cabeceras de tipos, y un contenido en la respuesta, por lo que nuestras aplicaciones pueden enviar el contenido en el formato que quieran, siempre y cuando este contenido contenga la información necesaria para poder operar con el objeto en el caso de que tengamos permiso para hacerlo.

Mensajes autodescriptivos: cuando hacemos peticiones a un servidor, éste debería devolver una respuesta que nos permita entender sin lugar a duda cual ha sido el resultado de la operación, así como si dicha operación es cacheable, si ha habido algún error, etc. HTTP implementa esto a través del estado y una serie de cabeceras. El cómo se usan estos estados y cabeceras depende por entero de la implementación de nuestro programa, en otras palabras, REST no fuerza el contenido de dichos elementos, por lo que el programa que se ejecuta en el servidor, y que en última instancia accede a los recursos y opera con ellos, tiene la responsabilidad de devolver estados y cabeceras que se correspondan con el estado real de la operación realizada. Esto es importante tenerlo en cuenta, ya que, desgraciadamente, un gran número de aplicaciones y servicios web no respetan esta regla (por lo tanto no pueden ser considerados REST), lo que nos lleva a tener que realizar todo tipo deworkarounds y cosas por el estilo. En la segunda parte de esta serie veremos un caso práctico de un servicio web que no respeta esta norma, y daremos varias soluciones posibles.

HATEOAS: por último, y algo que la mayoría de servicios web no cumplen, es la necesidad de incluir en las respuestas del servidor toda aquella información que necesita el cliente para seguir operando con este servicio web. En otras palabras, el cliente no tiene porqué saber que cuando obtenemos, por ejemplo, un objeto cualquiera, tenemos además las opciones de modificarlo, o eliminarlo. El servidor debe enlazar a estas operaciones en la respuesta a dicha petición. De esta manera, lo único que necesita saber un cliente sobre una aplicación REST, es el punto de entrada (endpoint). Además nos garantiza más independencia entre el cliente y el servidor. Desgraciadamente, este último principio no se implementa en la mayoría de APIs que usamos hoy en día, por lo que, siendo estrictos, podríamos decir que la mayoría de servicios web no son 100% RESTful. No obstante, esto es una limitación menor que en prácticamente ningún caso supone un problema.

Así pues, sabiendo que un servicio web RESTful hace referencia a un servicio web que implementa la arquitectura REST, podemos ya dar una definición concisa, lo cual nos dejará claro cómo tenemos que implementarlo en nuestras aplicaciones. Un servicio web RESTful contiene lo siguiente:

URI del recurso. Por ejemplo: *http://api.servicio.com/recursos/casas/1* (esto nos daría acceso al recurso “Casa” con el ID “1″)

El tipo de la representación de dicho recurso. Por ejemplo, podemos devolver en nuestra cabecera "Content-type: application/json", por lo que el cliente sabrá que el contenido de la respuesta es una cadena en formato JSON, y podrá procesarla como prefiera. El tipo es arbitrario, siendo los más comunes **JSON, XML** y **TXT** y el usado en esta aplicación para tratar los datos que envía el servidor de la aplicación de escritorio ha sido **JSON**.

Operaciones soportadas: HTTP define varios tipos de operaciones (verbos) , que pueden ser **GET, PUT, POST, DELETE, PURGE**, entre otros. Es importante saber para que están pensados cada verbo, de modo que sean utilizados correctamente por los clientes.

Hipervínculos: por último, nuestra respuesta puede incluir hipervínculos hacia otras acciones que podamos realizar sobre los recursos. Normalmente se incluyen en el mismo contenido de la respuesta, así si por ejemplo, nuestra respuesta es un objeto en formato JSON, podemos añadir una propiedad más con los hipervínculos a las acciones que admite el objeto.

Ejemplo de petición a servicio web github:

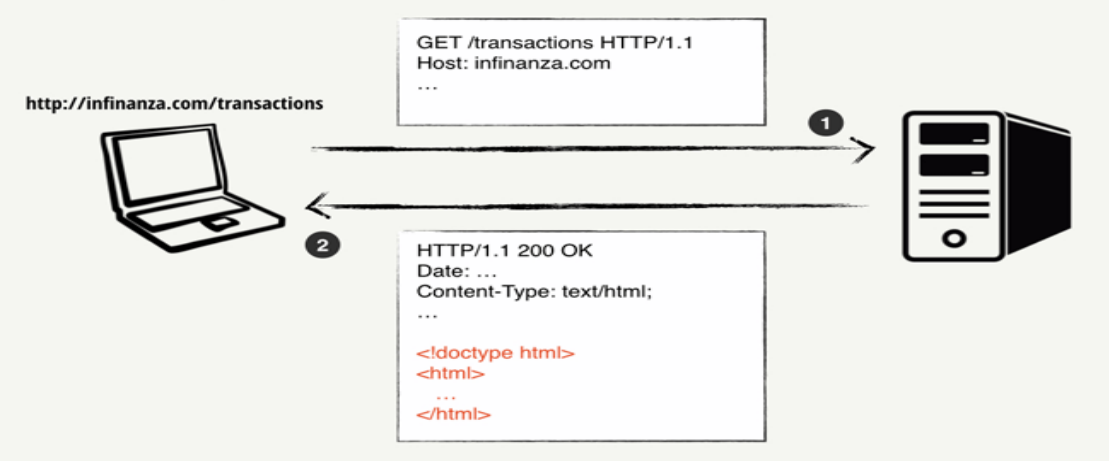


Figura 5.1.- Ejemplo de llamada REST

El formato de los objetos utilizados en el envío y recepción con los servicios web es JSONObject, que son una colección desordenada de pares <nombre,valor>, siendo el nombre la "etiqueta" que identifica cada atributo del objeto.

5.2.2. Beneficios-Desventajas Transformación SOAP-REST.

En la aplicación se han transformado los métodos de SOAP a REST, a continuación se expone el por qué de este cambio.

Beneficios de SOAP:

- Diseñado para tratar con entornos distribuidos.
- Es el standard de los servicios web, tiene mejor soporte y mejores herramientas.
- Facilidad para la extensibilidad.

Desventajas de SOAP:

- Conceptualmente mas difícil, más pesado que REST.
- Más detallado, por tanto, más extenso.
- Más difícil de desarrollar, requiere herramientas.

Beneficios de REST:

- Mucho más simple de desarrollar que SOAP.
- Más fácil de aprender, menos dependencia de las herramientas.
- Más conciso, no precisa de capa de mensajes adicional.
- Más cercano en diseño y filosofía a la WEB.

Desventajas de REST:

- Falta de standards de soporte para la seguridad, política, mensajería fiable, etc. Por tanto los servicios que tienen requerimientos más sofisticados son más difíciles de desarrollar.
- Atado al modelo HTTP de transporte.
- Asume una modelo de comunicación punto a punto, por tanto no se usa en computación distribuida.

Al ser más sencillo el desarrollo con REST y no necesitar la comunicación distribuida en la aplicación, se ha optado por trasladar los métodos al protocolo REST, que no requiere tanto código ni herramientas para su utilización.[5]

6.- Estructura de la aplicación.

6.1 Arquitectura global de la aplicación

Tal y como comentábamos anteriormente, TUNE-UP es una aplicación que basa su funcionamiento en el modelo Cliente-Servidor. La idea es que todos los datos y procesos realizados con los mismos se hagan en el servidor, y el cliente se utilice simplemente para interactuar con el usuario y mostrar los resultados que el servidor nos envía.

Desde la aplicación se invocará a los servicios web, que realizarán las tareas y manejo de datos correspondientes, y en el cliente únicamente se introducirán los datos y ordenes deseadas, como muestra la figura 5.2.

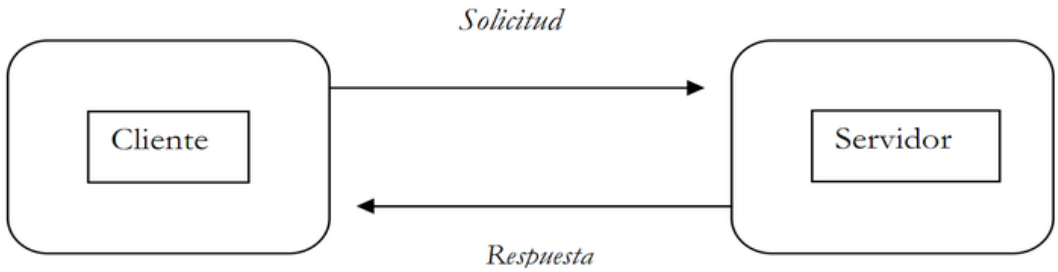


Figura 5.2.- Arquitectura global de la aplicación

De esta manera se consigue principalmente una perfecta coherencia en los datos, ya que todos los clientes trabajan sobre la misma base de datos alojada en el mismo servidor, y como ventaja adicional en el dispositivo Android (que no tenemos que olvidar que no tiene la potencia de un PC de escritorio actual), conseguimos un mayor rendimiento, ya que todas los accesos, búsquedas, y cálculos, son hechos por el servidor y el terminal sólo se limita a mostrar los resultados.

6.2 Estructura de Clases de la Aplicación

A continuación vamos a enumerar las diferentes clases de las que está compuesta la aplicación, así como el mapa general de la misma (Figura 5.3). No obstante, debe tenerse en cuenta de que esta aplicación es una versión de un programa de mucha mayor envergadura ya existente para PCs de escritorio, por lo que el diseño de muchas de las clases (sobre todo aquellas referentes a los objetos que se manejan), son también versiones más reducidas de las mismas clases que se utilizan en la aplicación de Windows, y están creadas para que se asemejen lo máximo posible (tanto en estructura y diseño, como en comportamiento), a sus análogas de versión de escritorio.

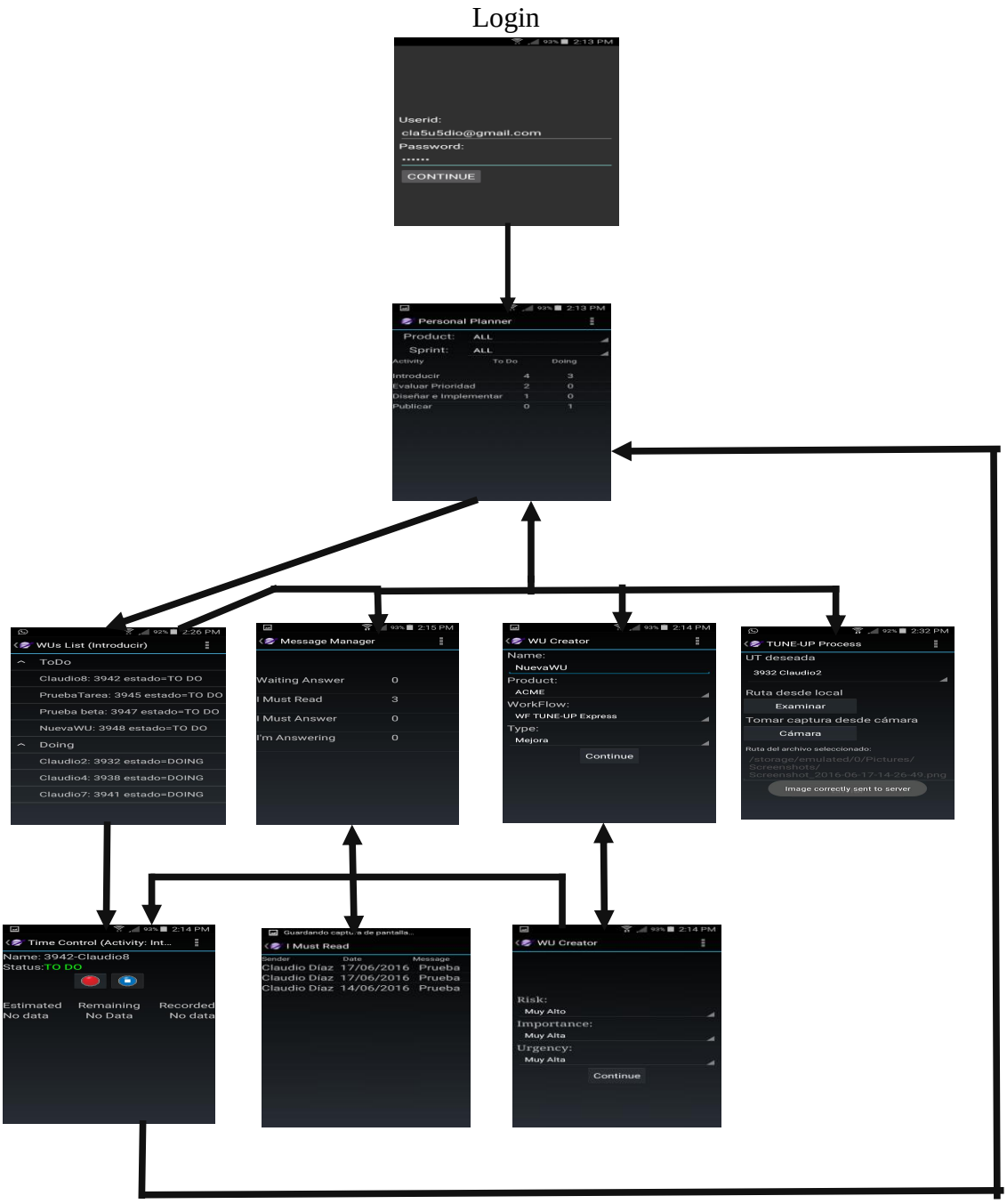


Figura 5.3 Mapa de la aplicación

Como se puede ver en la figura, la aplicación TUNE-UP para Android, está compuesta de 9 pantallas diferentes, por lo que para empezar, serán necesarias 9 clases para cada una de ellas, que son las siguientes:

- *Login.java*: Esta clase contiene todos los métodos necesarios y su implementación para hacer el login de la aplicación al servidor. La interfaz de usuario de dicha clase (UI), se compone de dos campos de texto editable (EditText), un control de tipo Spinner para la selección del sitio, y un botón para aceptar la selección y comenzar el proceso de validación del login.

- *Pep.java*: Esta clase corresponde a la implementación de los métodos necesarios para mostrar el planificador personal (o Personal Planner), principal de TUNE-UP, donde se nos mostrarán las diferentes categorías y el número de tareas pendientes y comenzadas, así como los controles necesarios para aplicar los filtros de búsqueda por producto y versión. La UI de dicha clase (UI), se compone de 2 controles de selección para los filtros de tipo "Spinner", y un elemento Lista para mostrar los resultados.

- *WUsList.java*: Esta clase corresponde con la visualización del listado de tareas pendientes y en progreso de ejecución (a la que accedemos una vez hemos seleccionado una categoría del Personal Planner. La UI de esta clase se compone de una lista expandida, que consta de dos grupos (TO DO, y DOING para las tareas pendientes y en progreso respectivamente), y la lista de UTs anidada en la categoría correspondiente.

- *TimeControl.java*: Esta clase contiene la implementación de la pantalla de control de tiempos de la UT que hemos seleccionado en la interfaz de usuario anterior (UT's List). A través de ella se puede comenzar, reanudar, o incluso finalizar la contabilización de tiempo de la UT anteriormente mencionada. La UI de esta clase cuenta con dos botones (uno para comenzar y pausar la contabilización de tiempo, y otro de "Stop" para dar por finalizada la contabilización del mismo), aparte de unos campos de texto para mostrar información acerca del tiempo acumulado, el restante, el estimado, el nombre de la UT y su estado.

- *NewWU.java*: A través de esta clase tenemos implementada la interfaz gráfica que nos permite crear nuevas unidades de trabajo (UTs). La UI se compone de un campo de texto editable para seleccionar el nombre, y 3 controles de selección desplegable de tipo Spinner para seleccionar el Producto, WorkFlow, y tipo de UT. Además de un botón que nos permite aceptar todos estos parámetros y pasar a la pantalla siguiente (NewUT2).

- *NewWU2.java*: Esta clase implementa el código que permite al usuario seleccionar ciertos parámetros de la UT recién creada, como son la importancia, la urgencia, y el riesgo. La UI se compone de tres desplegables de tipo Spinner que nos mostrarán los diferentes tipos anteriormente mencionados, y un botón para aceptar esos cambios.

- *Message.java*: Esta clase corresponde a la implementación de la pantalla principal del sistema de mensajes de la aplicación. A través de esta pantalla, podemos ver la lista de los diferentes tipos de mensajes que tenemos disponibles en la aplicación (dentro de las categorías de recibidos y enviados), así como el número pendiente que tenemos para cada una. La UI se compone de una simple lista por categorías, donde el usuario podrá seleccionar cada una de ellas.

- *MessageDetails.java*: Esta clase representa la interfaz gráfica que muestra los detalles de cada mensaje. Al igual que la pantalla anterior, la UI se trata de una simple lista.

- SendImage.java: Esta clase representa la interfaz gráfica que permite el envío de imágenes a una UT en el servidor de TUNE-UP, está formada por un Spinner que contiene las UT asociadas al agente logueado, dos botones para abrir el explorador de archivos o la cámara y un boton para enviarla.

Hasta aquí hemos enumerado y descrito, las clases de la aplicación que corresponden a las diferentes interfaces de usuario. No obstante, en la aplicación existen otras clases que son necesarias para el uso de objetos, métodos, y diferentes variables de entorno del sistema, que describiremos a continuación:

- *Entorno.java*: Esta clase, es una clase estática donde se guardan ciertas variables de entorno que serán utilizadas por el resto de clases de la aplicación, como son el identificador del agente "logueado", el identificador del sitio, y ciertos aspectos de la ubicación del servidor (dirección IP, ubicación web de los métodos, etc).

- *Comun.java*: Esta clase contiene métodos que pueden ser necesarios en varias partes de la aplicación, como puede ser el cambio de formatos de fecha, el método de descarga de mensajes (que es usado tanto por el PEP como por el servicio), así como los métodos que se encargan de transformar el objeto REST que recibimos del servidor, a los distintos objetos que manejamos en nuestra aplicación (Agente, Sitio, Producto,etc).

- *Agente.java*: Esta clase nos permite usar el objeto Agente, que tiene los siguientes atributos:

    - int IdAgente
    - String Nombre
    - Boolean Activo
    - String Email
    - String Contraseña
    - Boolean Administrador
    - Boolean PedirPassword

- *MensajesRecibidosVista.java*: Esta clase nos permite utilizar el objeto que hace referencia a los mensajes, y consta de los siguientes atributos:
    - int IdMensaje
    - int IdAgenteEmisor
    - int IdAgenteReceptor
    - String NombreAgenteEmisor
    - String NombreAgenteReceptor
    - String Respuesta
    - boolean Leido
    - String FechaEnvio
    - String FechaRespuesta
    - String Estado
    - Integer tipo
    - String NombreVersion
    - String NombreProducto
    - Integer IdUT
    - String NombreUT

- *MessageService.java*: Esta es una clase muy especial ya que se trata de la clase que contiene el servicio que se ejecuta en segundo plano mientras está activa la aplicación. Este servicio, que será descrito más adelantem cuenta con dos tareas programas (o TimerTask), que se encargan de gestionar la recepción de nuevos mensajes en la aplicación, así como la notificación al usuario.

- *MisActividadesAgente.java*: Esta clase nos permite utilizar y manipular el objeto Actividades, que son mostradas en el "Personal Planner". Cuenta con los siguientes atributos:

    - int IdActividad
    - String NombreActividad
    - Integer Posicion
    - Integer NumSeguimientosPendiente
    - int IdColorPendiente
    - Integer NumSeguimientosProgreso
    - int IdColorProgreso
    - Integer NumSeguimientosFinalizados

- *MisSeguimientos.java*: Esta clase nos permite utilizar el objeto Seguimiento, que será necesario para mostrar la lista de unidades de trabajo. Cuenta con los siguientes atributos:

    - int IdSeguimiento
    - int IdUT
    - String NombreUT
    - String Estado
    - String NombreActividad

- *Producto.java*: Esta clase nos permite manipular el objeto Producto, que será necesario a la hora de crear una nueva UT. Cuenta con los siguientes atributos:

    - int IdProducto
    - String Nombre
    - boolean Activo

- *RegistroTiempo.java*: Esta clase nos permite manipular el objeto de Registro de Tiempos, que será necesario en la pantalla de TimeControl. Cuenta con los siguientes atributos:

    - int IdRegitroTiempo
    - String FechaInicio
    - String FechaFin
    - int IdAgente
    - int IdSeguimiento
    - double TiempoRegistrado
    - String Observacion

- *Sitio.java*: Esta clase representa al objeto Sitio, que será necesario al hacer el login a la aplicación. Cuenta con los siguientes atributos:

    - int IdSitio
    - String Nombre
    - int IdTipoSitio
    - boolean Activo
    - String FechaCreacion

- *TiempoActividad.java*: Esta clase representa al objeto TiempoActividad, que será necesario en la pantalla de TimeControl para mostrar características adicionales del registro de tiempos de una UT. Cuenta con los siguientes atributos:

    - Integer IdUT
    - Integer IdActividad
    - Double TiempoEstimadoInicial
    - Double TiempoEstimadoActual
    - Double TiempoRegistradoTotal

- *Tipo.java*: Esta clase se utiliza en la aplicación para albergar al objeto “Tipo”, a la hora de crear una nueva UT y elegir el tipo de la misma (Mejora, Fallo, Nuevo Requisito, etc) Sus atributos son:

    - int IdTipoUT
    - String Nombre
    - int Posicion

- *WorkFlow.java*: Esta clase representa al objeto WorkFlow, que será necesario a la hora de crear una nueva UT. Sus atributos son los siguientes:
    - int IdWorkflow
    - String Nombre
    - boolean Activo

- *Urgencia.java:* Esta clase representa al objeto Urgencia, el cuál define el nivel de urgencia de una unidad de trabajo al ser creada.

    - int IdTipoUrgencia
    - String Nombre
    - int Posicion

- Riesgo.java: Esta clase representa al objeto Riesgo, el cuál define el nivel de riesgo de una unidad de trabajo al ser creada.

    - int IdTipoRiesgo
    - String Nombre
    - int Posicion

- Importancia.java: Esta clase representa al objeto Importancia, el cuál define el nivel de importancia de una unidad de trabajo al ser creada.

    - int IdTipoImportancia
    - String Nombre
    - int Posicion

- Version.java: Esta clase se utiliza para representar al objeto Versión, el cual define un sprint de un producto.

    - String idVersion;
    - String Nombre;
    - String Descripcion;
    - String FechaEstIni;
    - String FechaEstFin;
    - String FechaRealIni;
    - String FechaERealFin;
    - int IdProducto;
    - boolean Entregado;

A continuación podemos ver un diagrama de clases simplificado (Figura 5.4) para hacernos una idea de cómo se relacionan todas estas clases entre sí. En este diagrama faltan tan sólo la clase común (que es un simple contenedor de métodos al que acceden todas las clases, los accesos de la mayoría de clases que representan la IU a la clase Entorno) y la clase Message, que no tiene relación alguna con el resto de clases ya que es una simple lista con 4 valores numéricos.

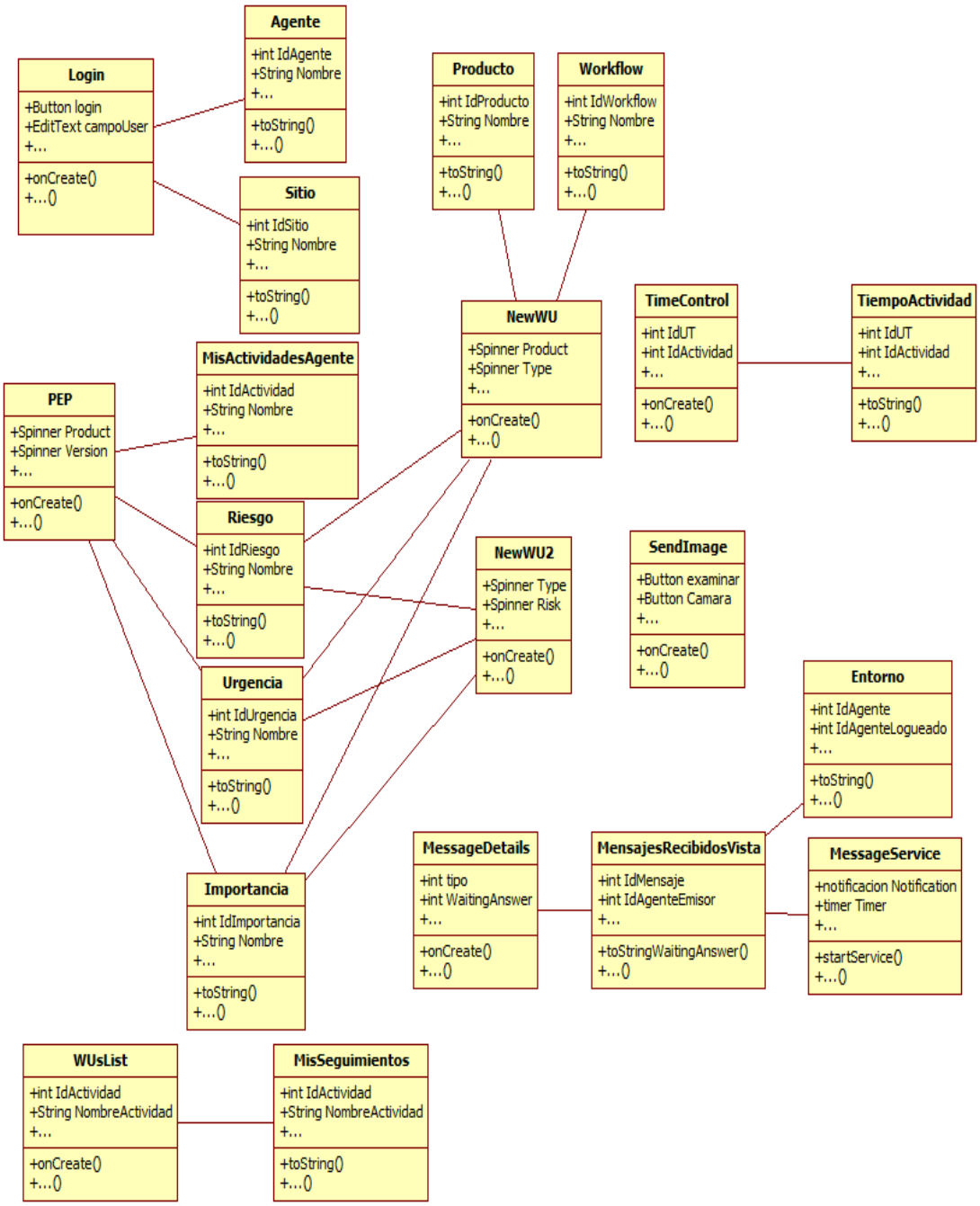


Figura 5.4.- Diagrama de clases de la aplicación

6.3 Diseño de la Interfaz de Usuario

Como hemos visto en el apartado anterior, la versión de TUNE-UP para Android cuenta con 9 pantallas diferentes, en las que, como es obvio, cada una de ellas tiene su propio interfaz de usuario diferente al resto. A continuación pasaremos a ver y a analizar cada una de ellas, empezando por la pantalla inicial del Login (Figura 5.3), con los diferentes desafíos y dificultades que se han ido presentando durante el desarrollo de la misma.

Hay que tener en cuenta que por razones obvias de potencia del terminal y sobre todo, de espacio de pantalla, se ha buscado siempre eliminar información que no sea de carácter esencial (ya que para eso tenemos la versión de escritorio), y que la cantidad final resultante de la misma sea lo más práctica y fundamental posible.

6.3.1 Refactorización de la interfaz de usuario:

Debido a que la aplicación original es muy antigua, la interfaz de usuario que utilizaba ha quedado obsoleta, y ha sido necesaria una refactorización para adaptarla a los nuevos recursos que han aparecido para el desarrollo android, ya que en el entorno de desarrollo utilizado (AndroidStudio) muchos de los elementos que utilizaba la aplicación habían quedado anticuados y/o no se podían utilizar en la versión actual de android.

Entre los cambios de la refactorización encontramos la adición de una barra de acciones (ActionBar) en todas las pantallas, la cuál permite una navegación entre actividades mucho más intuitiva y cómoda. También se ha modificado la fuente y su tamaño para una mejor interacción con el usuario, ya que en la anterior versión, resultaba costoso leer alguna información de la aplicación, como las unidades de trabajo en la lista de unidades de trabajo, o las categorías de los mensajes.

Tras la refactorización, las interfaces de la aplicación quedaron de la siguiente manera:

6.3.1.1 **Interfaz del Login**

5.3 Interfaz de Usuario del Login a la aplicación

Esta primera interfaz de usuario que nos muestra la figura 5.3, ofrece lo justo y necesario para que podamos completar el proceso de autentificación en el servidor de TUNE-UP. Para ello, necesitaremos proporcionarle al mismo nuestro nombre de usuario, la contraseña correspondiente, y finalmente el sitio dentro del servidor de TUNE-UP donde queremos autentificarnos.

Para realizar esta interfaz, se pretendió imitar el proceso de autentificación de la versión de escritorio, el cual, inicialmente pide el nombre de usuario y el password, dejando oculto tal y como muestra la figura 5.4, el desplegable que nos permite elegir el sitio (ya que inicialmente, hasta que el usuario no se identifica, el servidor no puede mostrar la lista de sitios porque ésta es personal para cada usuario).

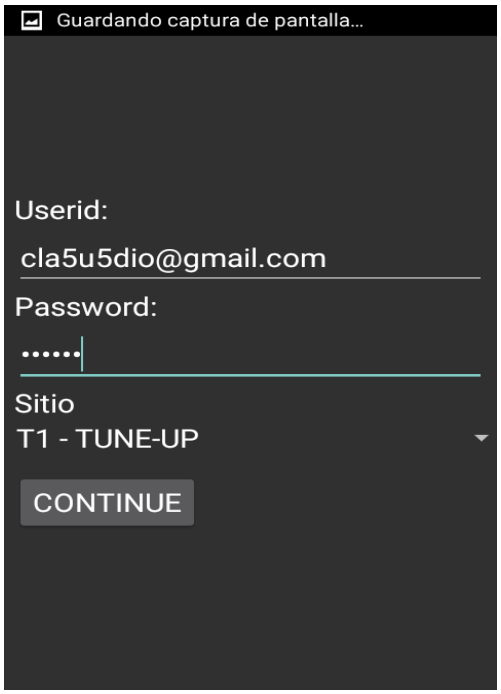


Figura 5.4 Proceso de Login

Para conseguir este "efecto", teníamos dos opciones: o bien diseñar el interfaz sin el desplegable de los sitios junto con su etiqueta, y añadirlos después dentro del código(opción por la que se optó), o bien tenerlos en la interfaz desde el principio, pero definiendo su atributo de visibilidad como oculto, y cambiando después el valor de dicho atributo una vez estuviese rellenado con datos. Por tanto, en la clase login, definimos el spinner como INVISIBLE al iniciar la aplicación

```
    SpinnerSitio.setVisibility(View.INVISIBLE);
    TextoSite.setVisibility(View.INVISIBLE);
```

Y después, los hacemos visibles en el momento que nos interesa de la siguiente forma:

```
if (ListaSitios.size() > 1) {
            if(SpinnerSitio.isShown()==false || sitio ==null) {
                  TextoSite.setVisibility(View.VISIBLE);
                  SpinnerSitio.setVisibility(View.VISIBLE);
            }else{
                   if(!Password.equals("")) {
                       Entorno.IdSitio = sitio.IdSitio;
                       entrar();
                   }else toastcamposvacios.show();
                                  }
```

6.3.1.2 Interfaz del Personal Planner

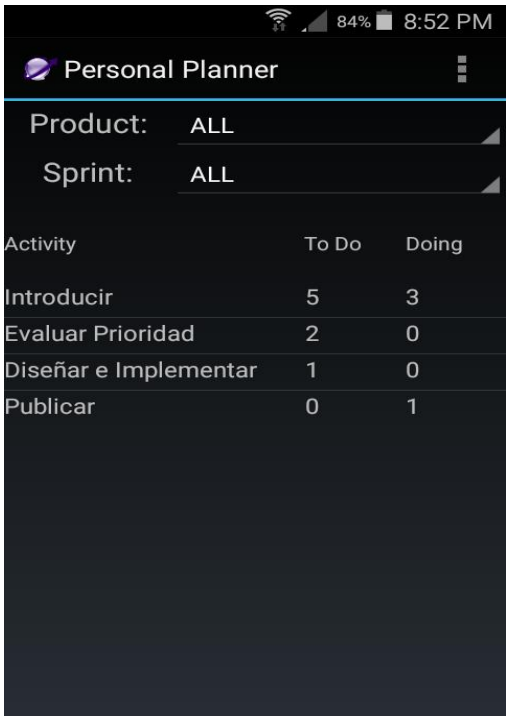


Figura 5.5 Interfaz de Usuario del Personal Planner

La figura 5.5, nos muestra la siguiente pantalla correspondiente a la interfaz de usuario del planificador personal. Esta interfaz, simplemente cuenta con dos controles desplegables tipo Spinner que nos servirán para aplicar el filtro (junto con sus dos etiquetas que muestran los textos de product y versión), una cabecera de la tabla compuesta por tres etiquetas también para indicar la Activity, y los textos To Do y Doing, y finalmente un ListView que será el encargado de mostrar los datos.

Cabe destacar que cuando trabajamos con listas, sean del tipo que sean (simples como la que nos ocupa, o expandibles o desplegables como las que nos mostrará la siguiente pantalla), es necesario crearles también un layout o plantilla xml (como si fuera una interfaz de usuario independiente), para luego asignarle dicho layout en su adaptador que será el encargado de plasmar esa información, dentro del espacio reservado para el ListView del layout de la aplicación principal. Esto, al igual que prácticamente cualquier cosa relacionada con los layouts en Android, puede hacerse mediante código java, o bien creándolo previamente y guardándolo como un fichero xml aparte dentro de nuestro proyecto como hemos optado en nuestra aplicación.

Lo normal, es que este layout simplemente contenga información detallada acerca de cómo mostrar cada una de las filas. Como podemos ver en el contenido del fichero filapep.xml que es el que nos ocupa, su contenido simplemente son 3 campos de texto o etiquetas (que corresponderán al nombre de la actividad, el número de tareas en To Do, y el número de tareas en Doing):

```
<LinearLayout android:id="@+id/listado_filas" android:layout_width="fill_parent"
   android:layout_height="fill_parent" android:orientation="horizontal"
   xmlns:android="http://schemas.android.com/apk/res/android">
```

```
  <TextView android:text="TextView" android:id="@+id/texto1"
     android:layout_width="fill_parent" android:layout_height="wrap_content"
     android:layout_weight="1" />
  <TextView android:text="TextView" android:id="@+id/texto2"
     android:layout_width="fill_parent" android:layout_height="wrap_content"
     android:layout_weight="2" />
  <TextView android:text="TextView" android:id="@+id/texto3"
     android:layout_width="fill_parent" android:layout_height="wrap_content"
     android:layout_weight="2" />
</LinearLayout>
```

Una vez definido el fichero, simplemente tenemos que asignarle el mismo al adaptador que crearemos dentro de nuestro código, que es quien se encargará realmente de mostrar la lista invocando al constructor de su superclase con los parámetros Context (contexto de la Activity principal en la que está ubicada la lista), el fichero que contiene el Layout para las filas (en nuestro caso el fichero filapep.xml que está referenciado automáticamente en la clase R), y el vector que contiene los datos de la tabla:

```
public class Mi_Adaptador extends ArrayAdapter<MisActividadesAgente> {
        Activity context;

        public Mi_Adaptador(Activity context) {

              super(context, R.layout.filapep, listaActividades);
```

6.3.1.3 Interfaz del listado de UTs

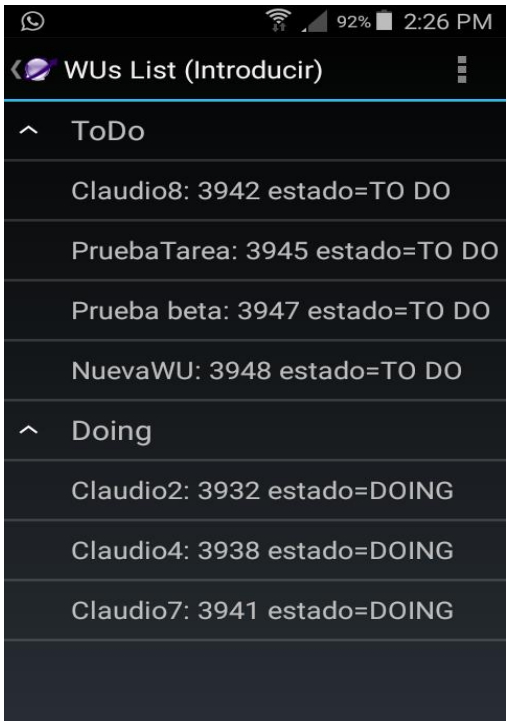


Figura 5.6 Interfaz de Usuario de la Lista de UTs

La figura 5.6, nos muestra la interfaz de usuario correspondiente a la lista de UTs de la actividad que hemos seleccionado en la pantalla anterior (nuestro planificador personal). El único elemento de este interfaz es una lista expandible, que cuenta con dos grupos "padre" (To Do y Doing), y las actividades correspondientes "hijas" clasificadas según su tipo.

Esta interfaz tiene como particularidad, que no se ha definido un layout para ella mediante un fichero xml como es lo usual. La explicación es sencilla: Como el único elemento de este interfaz es la propia lista expandible, y cualquier lista necesita su propio layout (que puede crearse mediante código), nos podemos ahorrar la definición de un fichero extra xml y crear la plantilla mediante código, con lo que optimizamos el espacio que ocupa nuestra aplicación en disco. Para ello utilizaremos el método getExpandableListView() de la clase ExpandableListView de Android (fuera de la clase adaptador que crearemos para mostrar los datos de nuestra lista), que nos servirá para crear el elemento ListView que antes teníamos que definir en el xml. Una vez hecho esto, y dentro de la clase adaptador que utilizaremos para mostrar la información), definiremos características comunes a "padres" e "hijos" con el método getGenericView(). Como podemos ver en nuestro caso, simplemente creamos un campo de texto con tamaño de letra 17 y alineación centrada en vertical, con un espacio de 40 pixels para cada fila. Después, para los objetos "padre", podemos personalizarlos en el método getGroupView, y con el método getChildView haremos lo mismo para los elementos "hijos".

```
ExpandableListView Lista = getExpandableListView();

public TextView getGenericView() {
            // Parametros generales del Layout
            AbsListView.LayoutParams lp = new AbsListView.LayoutParams(
                    ViewGroup.LayoutParams.FILL_PARENT, 40);

            TextView textView = new TextView(UTsList.this);
            textView.setLayoutParams(lp);
            textView.setTextSize(17);
            // Centramos el texto Verticalmente
            textView.setGravity(Gravity.CENTER_VERTICAL | Gravity.LEFT);
            // Seleccionamos la posicion inicial de escritura
            textView.setPadding(36, 0, 0, 0);
            return textView;
        }

public View getGroupView(int groupPosition, boolean isExpanded, View convertView,
                ViewGroup parent) {
            TextView textView = getGenericView();
            textView.setText(getGroup(groupPosition).toString());
            return textView;
        }

public View getChildView(int groupPosition, int childPosition, boolean isLastChild,
                View convertView, ViewGroup parent) {
            TextView textView = getGenericView();
            textView.setText(getChild(groupPosition, childPosition).toString());
            return textView;
        }
```

Finalmente, para conseguir que la lista se muestre totalmente desplegada por defecto para una mayor comodidad para el usuario, simplemente hubo que introducir estas líneas:

```
Lista.expandGroup(0);
Lista.expandGroup(1);
```

6.3.1.4 Interfaz de la pantalla de control de tiempos

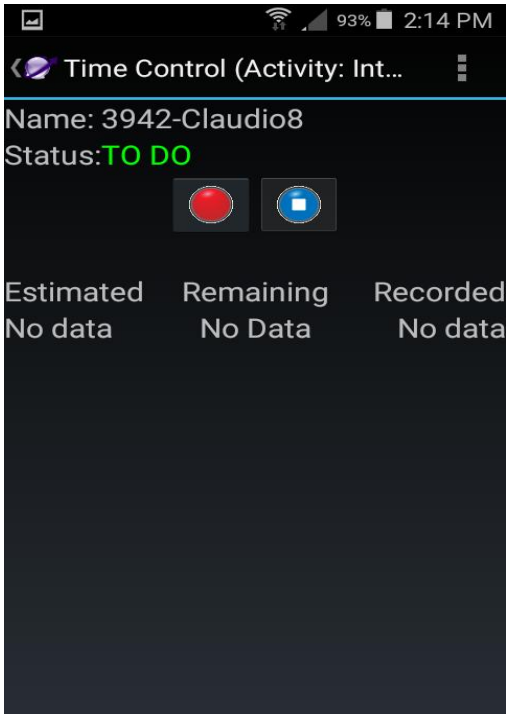


Figura 5.7 Interfaz de Usuario de la Lista de UTs

Como se puede observar en la figura 5.7, la pantalla de control de tiempos está compuesta básicamente por dos etiquetas superiores que sirven de cabecera, donde se muestran el nombre de la UT seleccionada y su estado actual, dos botones de tipo "ImageButton" (se diferencian de un botón estándar básicamente en que pueden asociarse a una imagen y adaptarse a su tamaño), y tres etiquetas inferiores que nos muestran datos acerca de los tiempos ya grabados para la UT en cuestión.

A diferencia de la versión de escritorio de TUNE-UP, que cuenta con 3 botones para los funciones de REC, PAUSA, y STOP, en la versión Android por estética y por razones de espacio, se ha utilizado el mismo botón para las funciones de REC y PAUSA. Para ello, es necesario cambiar el atributo correspondiente a la imagen mediante código en el momento preciso (es decir, cuando el usuario pulse el botón). Para ello, se creó el método SwitchImagenBoton(), que se invoca en el método onClickListener() del botón, que simplemente comprueba la imagen actual y llama al método CambiarImagenBotón (que es el que hace el trabajo en si), en función de la misma, tal y como vemos a continuación:

```
public void SwitchImagenBoton(){
        if (Imagen==ImagenPausa)
        {
            CambiarImagenBoton(ImagenPlay);
        }
        else{
            CambiarImagenBoton(ImagenPausa);
            lanzarNotificacion();
        }
    }

public void CambiarImagenBoton(int ImagenDeseada){
        if (ImagenDeseada==ImagenPausa){
            BotonPlayPause.setImageResource(R.drawable.pausar);
            Imagen=ImagenPausa;
        }
        else{
            BotonPlayPause.setImageResource(R.drawable.empezar);
            Imagen=ImagenPlay;
            borrarNotificacion();
        }
    }
```

Otro aspecto que causó un inconveniente también relacionado con el cambio de imagen de los botones, fue el hecho de que este cambio no se hacía de manera inmediata y la imagen se quedaba "bloqueda" hasta que el terminal finalizaba la negociación con el servidor. Todo esto incluso aunque el método SwitchImagenBoton() anteriormente comentado, se invocaba antes de hacer la solicitud al servidor tal y como se muestra a continación:

```
BotonPlayPause.setOnClickListener(new View.OnClickListener() {

            @Override
            public void onClick(View arg0) {
                if (HayConexionServidor()){
                    BotonStop.setEnabled(true);
                    SwitchImagenBoton();
                    if (CronoPausado) {
                        TextoEstado.setTextColor(Color.YELLOW);
                        TextoEstado.setText("PAUSED");
                    }
                    else{
                        TextoEstado.setTextColor(Color.RED);
                        TextoEstado.setText("ACTIVE");
                    }
                    SolicitarInicioPausaAlServidor();
                }
                else{
                    MostrarErrorConexion();
                }
            }
        });
```

Para solucionar este problema, lo que se hizo fue implementar el método SolicitarInicioPausaAlServidor() para que hiciera todo el trabajo de negociación con el servidor en un hilo, dejando libre al hilo principal para que hiciera el cambio de imagen tal y como se muestra a continuación:

```
public void SolicitarInicioPausaAlServidor(){

        final ProgressDialog dialog = ProgressDialog.show(this, "",
                "Requesting to server...", true);
        final Handler handler = new Handler() {
            public void handleMessage(Message msg) {

                        MostrarTiempos();
                        dialog.dismiss();
                }
            };
            Thread hilo = new Thread() {
                public void run() {
                    EmpezarPausarTiempo();
                    DescargarTiempos();
                    handler.sendEmptyMessage(0);
                    }
            };
        hilo.start();
    }
```

6.3.1.5 Interfaz del la pantalla de creación de UTs

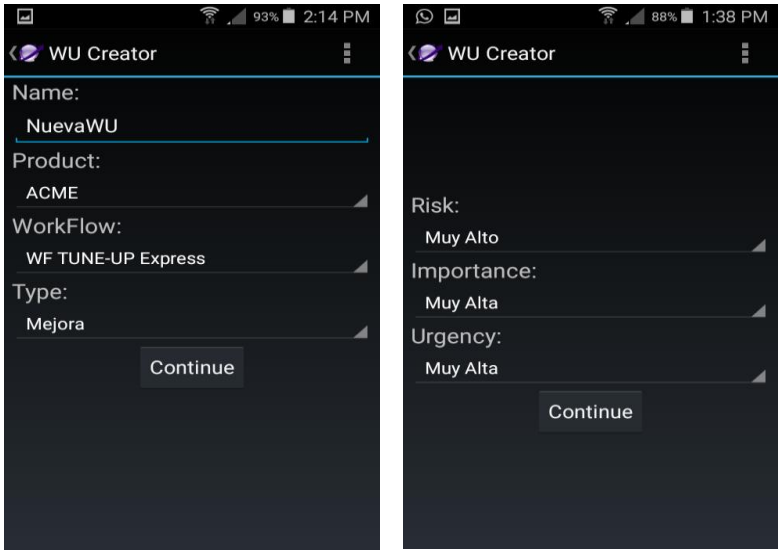


Figura 5.9 Interfaz de la pantallas de creación de UTs

Como podemos ver en la figura 5.9, todo lo referente a la creación de UTs de nuestra aplicación se hace en realidad en dos pantallas distintas, aunque debido a las similitudes entre ambas (básicamente solo se diferencian en la existencia de un campo de texto adicional para seleccionar el nombre en la primera de ellas), pasaremos a verlas dentro del mismo apartado.

Como acabamos de comentar, básicamente se tratan de interfaces que contienen controles desplegables de tipo Spinner casi en exclusiva, por lo que a nivel de implementación no hubo grandes complicaciones en este caso.

6.3.1.6 Interfaz del Administrador de Mensajes

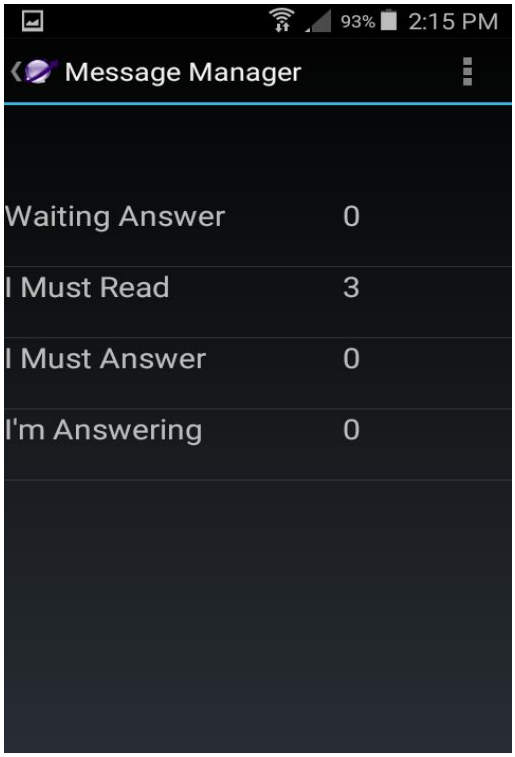


Figura 5.10 Administrador de Mensajes

Tal y como podemos observar en la figura 5.10, la interfaz del administrador de mensajes es una interfaz muy simple que básicamente contiene un elemento ListView, completamente similar al que vimos en la interfaz del planificador personal.

El único desafío o reto que supuso la creación de esta interfaz, fue la clasificación de los mensajes en 4 grupos distintos diferenciados como se pueden ver en la figura (Waiting Answer, I Must Read, I Must Answer, I'm Answering). Para ello, lo que hacemos es recorrer el vector donde están guardados estos mensajes, y rellenamos el atributo tipo de la clase MensajesRecibidosVista (que es la clase que define al objeto mensaje en si), con un número que va del 0 al 3 dependiendo del tipo de mensaje que sea y que coincide con la posición de la lista de clasificación (es decir, 0 para Waiting Answer, 1 para I Must read…,etc). Todo esto se hace en el método CalcularNumerosMensajes(), que está definido de la siguiente forma:

```
public void CalcularNumerosMensajes(){
        for (int i=0; i<Entorno.Mensajes.size();i++){
            MensajesRecibidosVista aux;
            aux=Entorno.Mensajes.get(i);
            if (aux.IdAgenteEmisor == Entorno.IdAgente &&
aux.FechaRespuesta.equals("")){
                TotalMensajesCategorias[0]++;
                Entorno.Mensajes.get(i).tipo=WaitingAnswer;
            }
            else{
                if (aux.IdAgenteEmisor == Entorno.IdAgente &&
!(aux.FechaRespuesta.equals("")) && aux.Leido == false){
                    TotalMensajesCategorias[1]++;
                    Entorno.Mensajes.get(i).tipo=IMustRead;
                }
                else{
                    if (aux.IdAgenteReceptor == Entorno.IdAgente &&
aux.Estado.equals("PENDING") && aux.FechaRespuesta.equals("")){
                        TotalMensajesCategorias[2]++;
                        Entorno.Mensajes.get(i).tipo=IMustAnswer;
                    }
                    else{
                        if (aux.IdAgenteReceptor == Entorno.IdAgente &&
!(aux.Estado.equals("PENDING")) && aux.FechaRespuesta.equals("")){
                            TotalMensajesCategorias[3]++;
                            Entorno.Mensajes.get(i).tipo=IMAnswering;
                        }
                    }
                }
            }
        }
    }
```

6.3.1.7 Interfaz de los detalles de mensajes

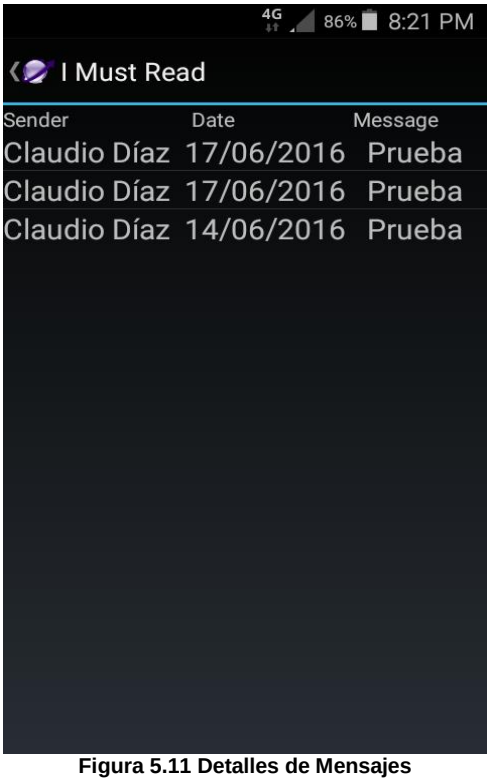


Figura 5.11 Detalles de Mensajes

Como podemos apreciar en la figura 5.11, el interfaz de usuario referente a los detalles de los mensajes no es muy diferente al de la figura 5.10 (el administrador general de mensajes). Al igual que el otro, es simplemente un elemento ListView con 3 columnas, una para identificar al Agente que envía (o recibe) el mensaje según el caso, la fecha, y el texto. No obstante, este interfaz tiene una particularidad respecto al anterior y es que al pinchar sobre una fila (es decir, como si seleccionáramos un mensaje), aparecerá un cuadro de diálogo con más detalles acerca de ese mensaje (el mensaje completo y su fecha, la actividad a la que hace referencia, la UT, etc).

En cuanto a la aparación del cuadro de diálogo con los detalles del mensaje, lo que se hizo fue utilizar la clase AlertDialog de Android, construyéndolo y lanzándolo dentro del método OnListItemClick() de la siguiente forma:

```
@Override
    public void onListItemClick(ListView l, View v, int position, long id){

        MensajesRecibidosVista aux=MensajesTipo.get(position);
        AlertDialog.Builder DetallesMensaje=new AlertDialog.Builder(this);
        DetallesMensaje.setTitle("Message
Details").setMessage(aux.DetallesMensaje());
         DetallesMensaje.setPositiveButton("Back",new
DialogInterface.OnClickListener() {
            public void onClick(DialogInterface dialog, int whichButton) {
            }
      });
         DetallesMensaje.show();

    }
```

6.3.1.8 Interfaz del envío de imágenes a una UT

Figura 5.12 Envio de imagen a UT

Como se puede apreciar en la imagen, la interfaz está formada por 3 botones, uno para abrir el explorador, otro para abrir la cámara, y otro para enviar la imagen. Dispone además de un campo de texto para introducir la UT al a que enviar la imagen, y dos RadioButton para seleccionar desde donde tomar la imagen y un mensaje en formato Toast para confirmar que la imagen se ha enviado correctamente al servidor.

La mayor complicación de esta interfaz fue lograr abrir un explorador de archivos pulsando un boton, y se consiguió llamando a otra actividad mediante un intent, que ejecuta éste código:

```
Intent intent = new Intent();
                intent.setType("image/*");
                intent.setAction(Intent.ACTION_GET_CONTENT);//
                startActivityForResult(Intent.createChooser(intent,           "Select
File"),request_code);
```

6.4 Diseño del Servicio

TUNE-UP para Android cuenta con un servicio que trabaja en segundo plano siempre y cuando tengamos la aplicación principal activa, que se encarga de comprobar cada 15 minutos si tenemos mensajes nuevos en el servidor.

Android tiene dos formas de trabajar con servicios: la vinculación y la iniciación. Un servicio iniciado está más relacionado con el aspecto de una tarea de fondo, es decir, una "aplicación" que se encarga de hacer trabajo de fondo de manera transparente para el usuario, sin interaccionar con el mismo, y sin una conexión concreta a otra actividad o aplicación. La vinculación en cambio, implica el uso de un objeto ServiceConnection para conectar y obtener una referencia remota, permitiendo acceder así a un objeto remoto e invocar métodos definidos desde una actividad. Puede incluso interactuar con la interfaz de la aplicación, enviando mensajes de alerta al usuario, etc. Además, un servicio puede vincularse a varias aplicaciones a la vez, permitiendo así compartir resultados con distintas aplicaciones.

En nuestro caso, optamos por un servicio iniciado, dedicado única y exclusivamente a nuestra aplicación. Por esto, y por el hecho de que no necesitábamos interactuar con la interfaz de usuario (la única función parecida de nuestro servicio es la notificación de nuevos mensajes y esto no forma parte de la interfaz de TUNE-UP, sino del propio sistema operativo), se eligió esta opción.

Entrando en detalles acerca del funcionamiento del servicio, y si analizamos el código del mismo, veremos que está compuesto por dos tareas programadas (TimerTask de JAVA). Una que se activa cada 15 minutos, descarga los mensajes y comprueba si existen nuevos (activando una variable booleana en este caso), y otra tarea que se activa una vez por minuto, que comprueba el valor de esta variable booleana y si detecta que es cierta, significará que la tarea principal ha encontrado nuevos mensajes. Todo esto se puede ver a continuación en el siguiente fragmento de código:

```
private void startservice() {
        timer.scheduleAtFixedRate( new TimerTask() {
        public void run() {
            if (HayConexionInternet()){
                Mensajes=comun.DescargarMensajes();
                if(HayMensajesNuevos()){
                    Entorno.Mensajes=Mensajes;
                }
            }
        }
        }, 0, 60000 * 15);;
                    }, 0, 60000 * 1);;
```

El por qué de esta forma de trabajo fue por lo siguiente: Debido al funcionamiento de Android, y por razones fundamentalmente de ahorro de energía, cuando el teléfono entra en modo suspensión (es decir, se apaga la pantalla porque el terminal detecta que el usuario no está interactuando con él), se desactivan muchos procesos y servicios internos del sistema, para poder poner al procesador en un modo de bajo consumo.

Finalmente, y una vez ya hemos comentado todos los aspectos destacables acerca del servicio y su funcionamiento, ya tan solo nos queda detallar en qué momento preciso entra este en funcionamiento, y por supuesto cuando se para la ejecución del mismo.

En cuanto a la iniciación, se hace justo después de autentificarnos en el servidor (es decir, tras hacer el login). Concretamente, en la interfaz correspondiente al Personal Planner, nada más iniciar la misma tal y como se puede ver en su código:

```
@Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        SpinnerProduct=(Spinner)findViewById(R.id.SpinnerProduct);
        SpinnerVersion=(Spinner)findViewById(R.id.SpinnerVersion);

        setTitle("Personal Planner");

        if (comun.HayConexionInternet(this)){

            Entorno.Mensajes=comun.DescargarMensajes();
         rellenarSpinnerProductos();
         //rellenarSpinnerTodasVersiones();
         //rellenarLista();
        }
        else{
         comun.MostrarErrorConexion(this);
        }

        Intent svc = new Intent(this, MessageService.class);
        startService(svc);
```

```
    }
```

En cuanto al momento en que se para, como queremos que el servicio esté siempre en funcionamiento mientras esté funcionando la aplicación, y la única manera de finalizar completamente esta es pulsando el botón de “BACK” en la misma pantalla del Personal Planner, se hace dentro del método onDestroy() de la misma (que recordemos que es el método que ejecuta Android cuando destruye cualquier activity), tal y como se ve a continuación:

```
@Override
    public void onDestroy() {
      super.onDestroy();
      {
        Intent svc = new Intent(this, MessageService.class);
        stopService(svc);
      }
```

7.- Pruebas de desarrollo

Durante el desarrollo de la aplicación, se utilizó la versión TUNE-UP de PC's de escritorio para diseñar las pruebas. Para ello, se creó una nueva UT llamada "Actualización de PAs" (Pruebas de Aceptación), y en ella fuimos añadiendo Tests de Aceptación para cada una de las interfaces, tal y como se puede ver en la figura 6.1

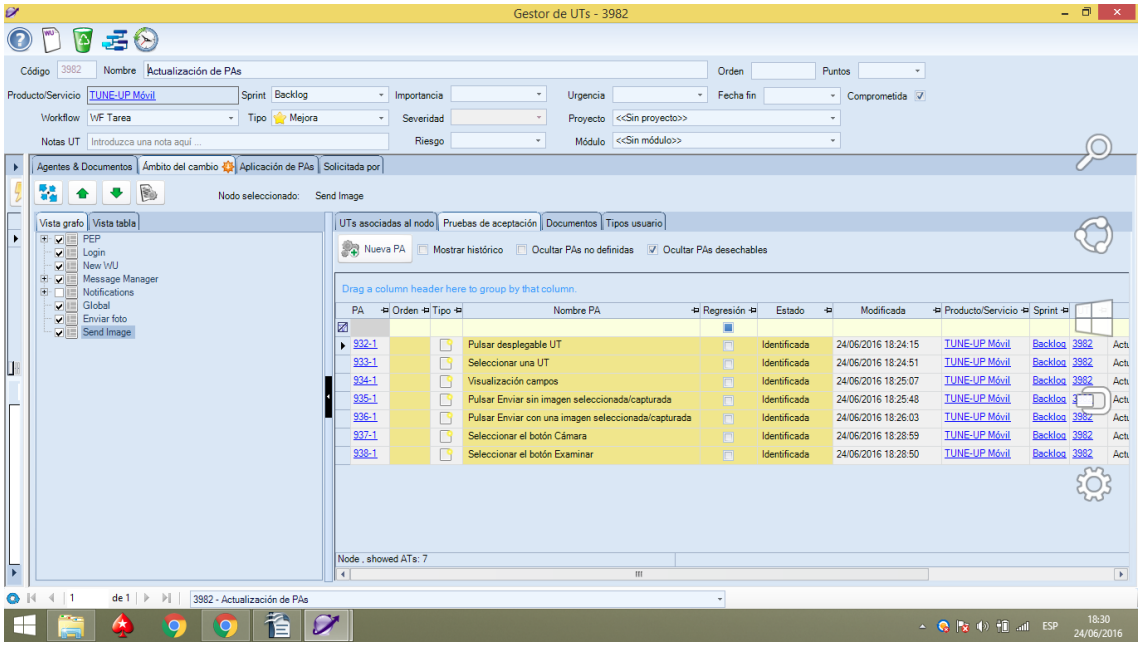
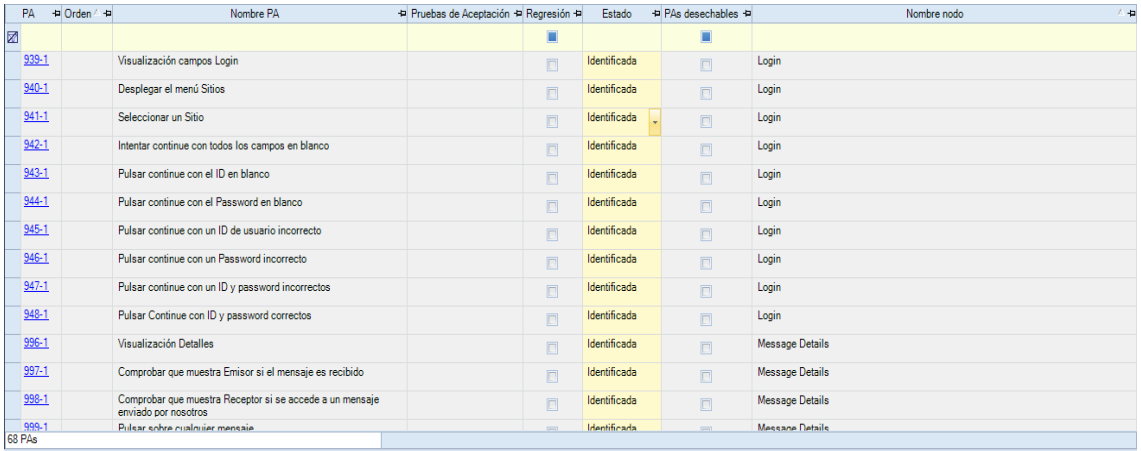


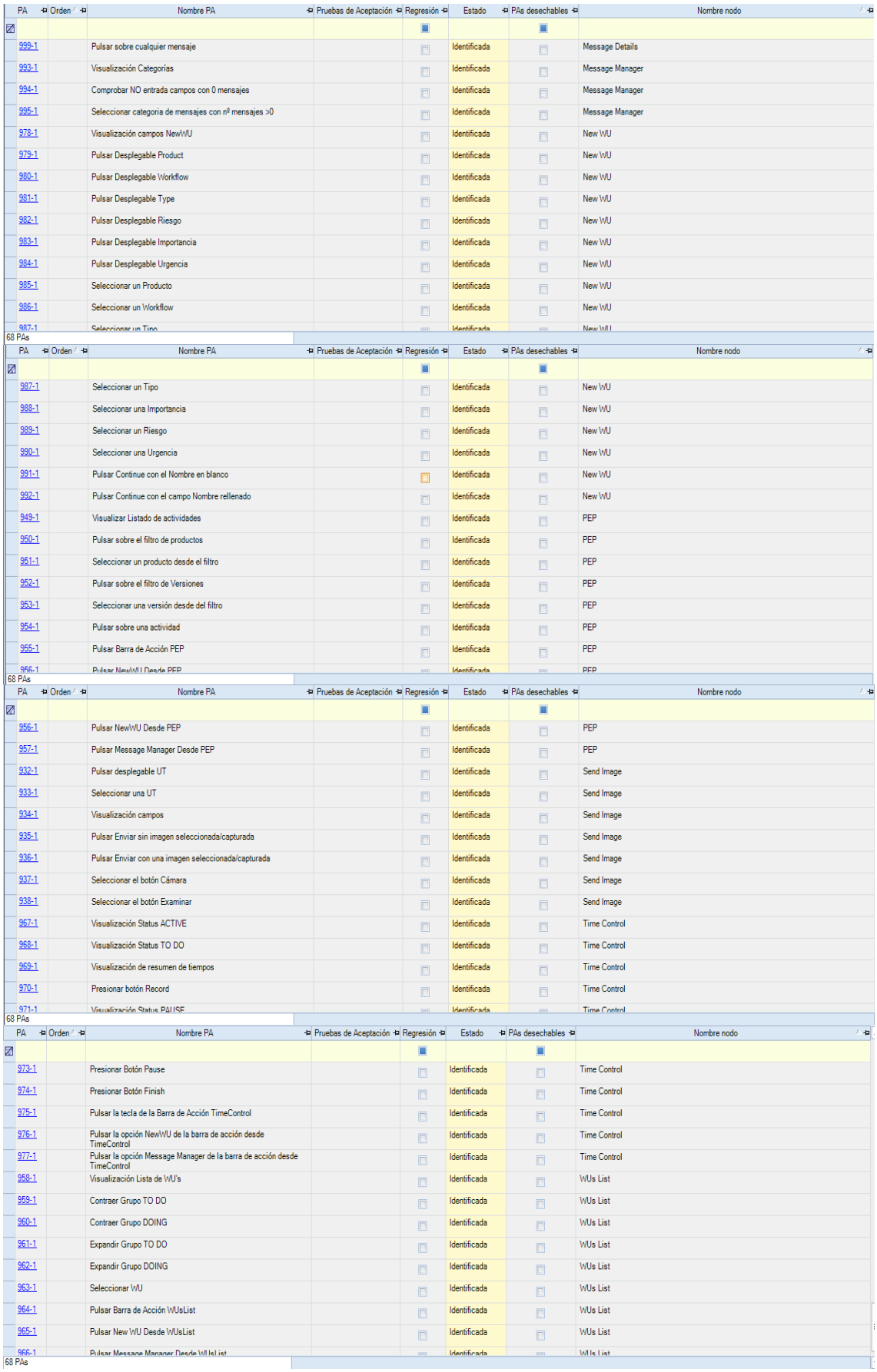
Figura 6.1 Test de Aceptación para la interfaz de envio de imágenes.

Finalmente, en el Requeriments Manager (REM), mostrado en la figura 6.2 , podemos ver el total de todas las pruebas diseñadas (concretamente 67), las cuales serán detalladas en el anexo A.2.



PA	Orden	Nombre PA	Pruebas de Aceptación	Regresión	Estado	PAs desechables	Nombre nodo
939-1		Visualización campos Login			Identificada		Login
940-1		Desplegar el menú Sitios			Identificada		Login
941-1		Seleccionar un Sitio			Identificada		Login
942-1		Intentar continue con todos los campos en blanco			Identificada		Login
943-1		Pulsar continue con el ID en blanco			Identificada		Login
944-1		Pulsar continue con el Password en blanco			Identificada		Login
945-1		Pulsar continue con un ID de usuario incorrecto			Identificada		Login
946-1		Pulsar continue con un Password incorrecto			Identificada		Login
947-1		Pulsar continue con un ID y password incorrectos			Identificada		Login
948-1		Pulsar Continue con ID y password correctos			Identificada		Login
996-1		Visualización Detalles			Identificada		Message Details
997-1		Comprobar que muestra Emisor si el mensaje es recibido			Identificada		Message Details
998-1		Comprobar que muestra Receptor si se accede a un mensaje enviado por nosotros			Identificada		Message Details
999-1		Pulsar sobre cualquier mensaje			Identificada		Message Details

68 PAs



PA	Orden	Nombre PA	Pruebas de Aceptación	Regresión	Estado	PAs desechables	Nombre nodo
999-1		Pulsar sobre cualquier mensaje			Identificada		Message Details
993-1		Visualización Categorías			Identificada		Message Manager
994-1		Comprobar NO entrada campos con 0 mensajes			Identificada		Message Manager
995-1		Seleccionar categoria de mensajes con nº mensajes >0			Identificada		Message Manager
978-1		Visualización campos NewWU			Identificada		New WU
979-1		Pulsar Desplegable Product			Identificada		New WU
980-1		Pulsar Desplegable Workflow			Identificada		New WU
981-1		Pulsar Desplegable Type			Identificada		New WU
982-1		Pulsar Desplegable Riesgo			Identificada		New WU
983-1		Pulsar Desplegable Importancia			Identificada		New WU
984-1		Pulsar Desplegable Urgencia			Identificada		New WU
985-1		Seleccionar un Producto			Identificada		New WU
986-1		Seleccionar un Workflow			Identificada		New WU
987-1		Seleccionar un Tipo			Identificada		New WU

68 PAs

PA	Orden	Nombre PA	Pruebas de Aceptación	Regresión	Estado	PAs desechables	Nombre nodo
987-1		Seleccionar un Tipo			Identificada		New WU
988-1		Seleccionar una Importancia			Identificada		New WU
989-1		Seleccionar un Riesgo			Identificada		New WU
990-1		Seleccionar una Urgencia			Identificada		New WU
991-1		Pulsar Continue con el Nombre en blanco			Identificada		New WU
992-1		Pulsar Continue con el campo Nombre rellenado			Identificada		New WU
949-1		Visualizar Listado de actividades			Identificada		PEP
950-1		Pulsar sobre el filtro de productos			Identificada		PEP
951-1		Seleccionar un producto desde el filtro			Identificada		PEP
952-1		Pulsar sobre el filtro de Versiones			Identificada		PEP
953-1		Seleccionar una versión desde del filtro			Identificada		PEP
954-1		Pulsar sobre una actividad			Identificada		PEP
955-1		Pulsar Barra de Acción PEP			Identificada		PEP
956-1		Pulsar NewWU Desde PEP			Identificada		PEP

68 PAs

PA	Orden	Nombre PA	Pruebas de Aceptación	Regresión	Estado	PAs desechables	Nombre nodo
956-1		Pulsar NewWU Desde PEP			Identificada		PEP
957-1		Pulsar Message Manager Desde PEP			Identificada		PEP
932-1		Pulsar desplegable UT			Identificada		Send Image
933-1		Seleccionar una UT			Identificada		Send Image
934-1		Visualización campos			Identificada		Send Image
935-1		Pulsar Enviar sin imagen seleccionada/capturada			Identificada		Send Image
936-1		Pulsar Enviar con una imagen seleccionada/capturada			Identificada		Send Image
937-1		Seleccionar el botón Cámara			Identificada		Send Image
938-1		Seleccionar el botón Examinar			Identificada		Send Image
967-1		Visualización Status ACTIVE			Identificada		Time Control
968-1		Visualización Status TO DO			Identificada		Time Control
969-1		Visualización de resumen de tiempos			Identificada		Time Control
970-1		Presionar botón Record			Identificada		Time Control
971-1		Visualización Status PAUSE			Identificada		Time Control

68 PAs

PA	Orden	Nombre PA	Pruebas de Aceptación	Regresión	Estado	PAs desechables	Nombre nodo
973-1		Presionar Botón Pause			Identificada		Time Control
974-1		Presionar Botón Finish			Identificada		Time Control
975-1		Pulsar la tecla de la Barra de Acción TimeControl			Identificada		Time Control
976-1		Pulsar la opción NewWU de la barra de acción desde TimeControl			Identificada		Time Control
977-1		Pulsar la opción Message Manager de la barra de acción desde TimeControl			Identificada		Time Control
958-1		Visualización Lista de WU's			Identificada		WUs List
959-1		Contraer Grupo TO DO			Identificada		WUs List
960-1		Contraer Grupo DOING			Identificada		WUs List
961-1		Expandir Grupo TO DO			Identificada		WUs List
962-1		Expandir Grupo DOING			Identificada		WUs List
963-1		Seleccionar WU			Identificada		WUs List
964-1		Pulsar Barra de Acción WUsList			Identificada		WUs List
965-1		Pulsar New WU Desde WUsList			Identificada		WUs List
966-1		Pulsar Message Manager Desde WUsList			Identificada		WUs List

68 PAs

Figura 6.2 Administrador de Requisitos con el listado de pruebas diseñadas

El número total de pruebas realizadas ha sido 67, y todas ellas han sido superadas con éxito, por tanto la aplicación tendrá el comportamiento funcional deseado.

Las pruebas aplicadas se detallan más adelante en el capítulo 10 de la memoria (Anexo).

8. Conclusiones

Se ha alcanzado el objetivo de este TFG, hemos desarrollado una segunda versión de TUNE-UP móvil actualizada a las posibilidades de hoy en día para smartphones que cumple con las funciones básicas de la versión de TUNE-UP para escritorio, como son la visión del Planificador Personal con la lista de actividades que tenemos pendientes y en progreso, un contabilizador de tiempos para incluir nuevos registros, un pequeño creador de nuevas UTs, y un visor de mensajes, además de la funcionalidad añadida de poder cargar imágenes a una UT desde el propio móvil.

Obviamente la versión de TUNE-UP para escritorio ofrece mucho más, pero debido restricciones de los smartphones actuales, tales como su potencia y su reducido tamaño de pantalla y peculiar interface de usuario, sólo hemos implementado ciertas funcionalidades que podían aportar mayor valor salvando dichas restricciones.

La realización del TFG como proyecto, ofrece algunos datos ilustrativos:

- Tiempo aproximado dedicado a aprendizaje inicial de la plataforma Android: 45 horas.
- Tiempo aproximado de desarrollo de programación del módulo TUNE.UP para Smartphone: 125 horas.
- Tiempo aproximado de conversión de métodos SOAP a REST: 60 horas.
- Tiempo aproximado de aplicación de pruebas de aceptación: 20 horas.
- Número aproximado de líneas de código (no generadas automáticamente): 3773 Líneas.
- Número de pantallas: 10 pantallas
- Número de Pruebas de Aceptación: 67 pruebas
- Número de clases: 27 clases.

Hay funcionalidad que por razones de alcance del TFG no se ha incluido. Como siguiente paso se sugiere la siguiente lista de extensiones o mejoras futuras:

Gestión y control de posibles excepciones que pueden surgir al perder la conexión a Internet durante la comunicación con el servidor:

Actualmente la aplicación controla que exista conexión a Internet antes de lanzar cualquier petición de información al servidor, mostrando un mensaje de error si no es así, e incluso mandando al usuario a la pantalla de configuración del dispositivo para que active la conexión a Internet en caso de que ésta estuviera deshabilitada. No obstante, puede darse el caso, sobre todo tratándose de un dispositivo móvil, que la cobertura se pierda durante la negociación de datos con el servidor, y por tanto, la aplicación falle.

Posibilidad de responder a los mensajes recibidos directamente:

La aplicación actualmente sólo muestra los mensajes que hemos recibido o enviado, pero no da la posibilidad de responder a los mismos o reenviar éstos directamente, sin pasar por la versión de Escritorio.

Añadir una pestaña a la pantalla de control de tiempos para ver un histórico de activaciones y pausas:

Dicha funcionalidad, aunque inicialmente estuvo incluida en la aplicación, se desactivó debido a que ocupaba demasiado espacio en pantalla, y no mostraba otros aspectos mucho más funcionales y útiles como son el control de tiempo acumulado, tiempo estimado, y tiempo restante (que es lo que se muestra actualmente). Quizá sería interesante añadir una pestaña a dicha pantalla, que mostrara todo ese histórico.

Crear un pequeño diseñador de Workflows para hacer nuevos Workflows muy simplificados:

Esta funcionalidad representaría un desafío tanto a nivel de diseño de la interfaz, como a nivel de implementación, ya que de no encontrarse otra alternativa es posible que se tenga que implementar la funcionalidad usando el modo gráfico, algo que aumentaría significativamente la complejidad del desarrollo.

8.1 Valoración personal.

A nivel personal, el desarrollo y realización de este proyecto ha sido una experiencia muy positiva por varios aspectos. El primero de ellos, por la función que ha tenido en cuanto al refresco de los conocimientos de programación tanto en JAVA como a nivel general, adquiridos en otras asignaturas. En particular el desarrollo en la plataforma para aplicaciones con S.O. Android y su SDK ha sido un reto que se ha superado. Además, el aprendizaje y contraste de los protocolos SOAP y REST ha sido muy útil.

Me siento muy satisfecho al ver dichos conocimientos reforzados y aumentados, debido fundamentalmente al esfuerzo de luchar contra los pequeños problemas que han ido surgiendo día a día durante el desarrollo, los cuales me obligaban a investigar y a adquirir información por mi cuenta, ya fuera por Internet, consultando bibliografía o foros, o incluso pidiendo ayuda a compañeros de la universidad.

9. Referencias

[1] Página Web de TUNE-UP del DSIC → http://tuneup.dsic.upv.es

[2] Blog de desarrollo para Android → http://www.sgoliver.net/blog/?p=1313

[3] Estadísticas de mercado para las tecnologías de internet→ https://www.netmarketshare.com/operating-system-market-share.aspx?qprid=8&qpcustomd=1

[4] Memoria del alumno Luis Alejandro Ruiz Ramirez

[5] https://thanksnetwork.wordpress.com/2012/10/16/rest-vs-soap/

[6] Wikipedia → https://es.wikipedia.org/wiki/BlackBerry_10

[7] Página oficial de Apple → http://www.apple.com/es/ios/

[8] Foro Nokia → http://www.n8fanclub.com/2012/10/official-nokia-belle-fp2-firmware.html

[9] vBote Tecnología móvil → http:/www.vbote.com/vbote-solutions-academy-blog/124-frameworks-multiplataforma-write-once,-run-everywhere.html

[10] Servicios web RESTful con HTTP → http:/www.adwe.es/general/colaboraciones/servicios-web-restful-con-http-parte-i-introduccion-y-bases-teoricas

[11] TFG del alumno Alejandro Ruiz Ramírez.

10. Anexo

A.1 Manual de Usuario

En este anexo vamos a incluir un manual de usuario de la aplicación a modo de demo, tratando cada pantalla o interfaz de usuario por separado, y comentando las diferentes particularidades y opciones de cada una.

No obstante, antes llegar a ese punto, hay que mencionar un aspecto que, salvo en la pantalla inicial de Login, va a estar presente en todo momento en la aplicación. Y es que como se ha comentado en apartados anteriores, esta aplicación tiene un servicio activo que se encarga de recibir nuevos mensajes, que estará funcionando de fondo siempre y cuando la aplicación se encuentre abierta. Es muy importante tener esto en cuenta ya que si cerramos la aplicación (pulsando la tecla "BACK" del terminal Android en la pantalla del Personal Planner), el servicio dejará de funcionar y ya no recibiremos nuevos mensajes. Si queremos tener esta función activa, y por lo que sea, necesitamos acceder a cualquier otra función o programa de nuestro terminal, hay que "minimizar" la aplicación TUNE-UP pulsando la tecla "Home" del teléfono (en algunos terminales esta tecla tiene el dibujo de una casa). De este modo, la aplicación sigue abierta en segundo plano y el servicio continuará activo. Una vez dicho esto, podemos comenzar a detallar el funcionamiento y los diferentes procesos que nos iremos encontrando a lo largo del uso de la aplicación.

Lo primero que nos encontramos al lanzar la aplicación, es algo que está presente prácticamente en el 100% de las aplicaciones basadas en cliente-servidor. Esto es, el proceso de autentificación (Figura A.1). Una vez completado este proceso, ya entraremos en lo que es la aplicación en sí, se activará el servicio, y tendremos acceso a todas las funcionalidades.

A.1.1 Autentificación con el servidor

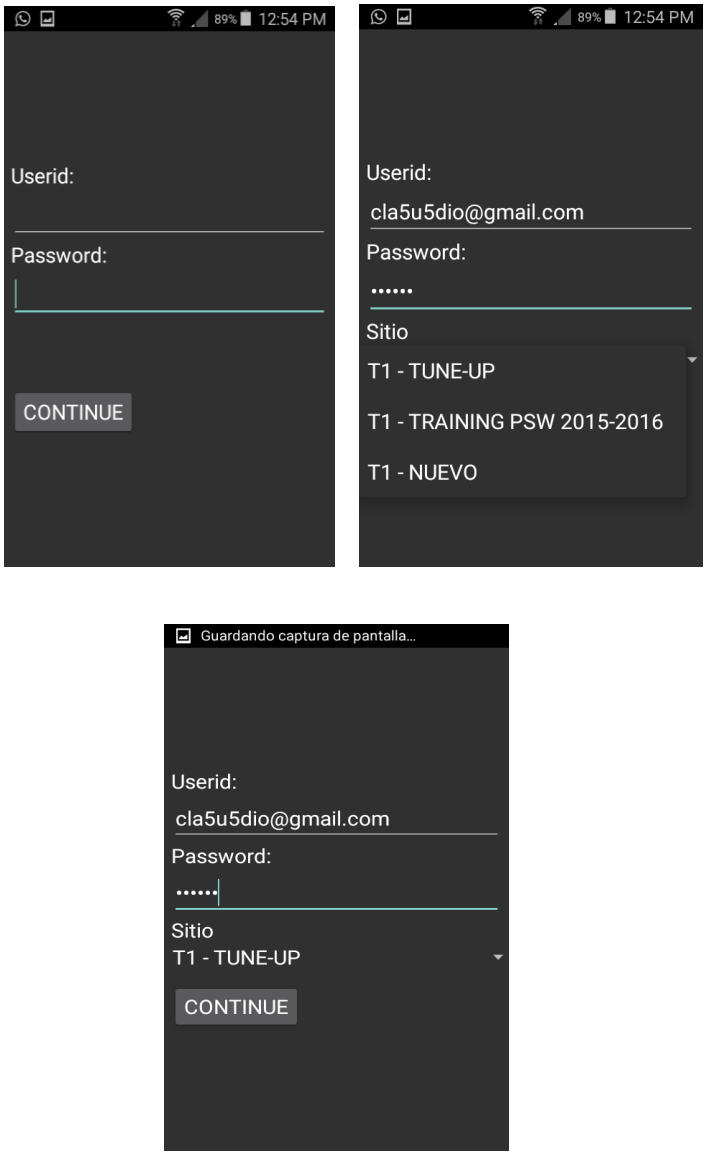


Figura A.1 Proceso de autentificación con el servidor

El proceso de autentificación en el servidor de TUNE-UP mediante Android, intenta ser fiel al mismo proceso de la aplicación Windows para PCs de escritorio. Por lo tanto, el funcionamiento será el mismo. Inicialmente nos aparecen dos campos en blanco (uno para el Id de usuario y otro para el password). Al introducirlos y pulsar el botón "Continue", nos aparecerá un desplegable con los sitios activos. Si pinchamos en el, como se puede ver en la figura A.1 nos dará a elegir entre todos ellos. Una vez seleccionado el sitio, pulsamos de nuevo el botón "Continue" y entraremos en la aplicación en el entorno del Planificador Personal (Figura A.1.2).

A.1.2 Planificador Personal

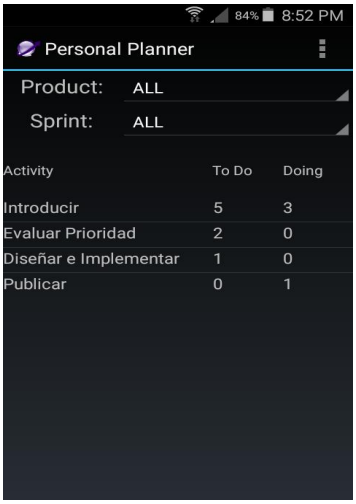


Figura A.1.2 Planificador Personal

El planificador Personal nos muestra una lista con 3 columnas de todas las actividades activas para el usuario autentificado, junto con el número de UT's que tiene pendientes (columna TO DO), y las que tiene en marcha (columna Doing). Como se puede observar en la parte superior de la interfaz, tenemos dos desplegables (Producto y Versión). Al seleccionar cualquiera de estos dos desplegables, lo que hacemos es aplicar un filtro sobre las actividades anteriormente mencionadas, con lo que al hacerlo, notaremos como el programa "refresca" el listado, y lo actualiza con los nuevos datos. Nótese también, que el hecho de seleccionar un producto acota también el listado del filtro de versiones (al igual que pasa en la versión de PC de escritorio).

Si pulsamos el botón de la esquina superior derecha correspondiente a la barra de acción (ActionBar, figura A.1.3), observaremos que se abre un menú con 3 opciones (New WU y Message Manager y Send Image), cuyos efectos detallamos a continuación:

- **New WU:** Accede a la pantalla de creación de nuevas UT's.
- **Message Manager:** Accede al administrador de mensajes.
- **Send Image:** Accede a la pantalla de envío de imágenes.

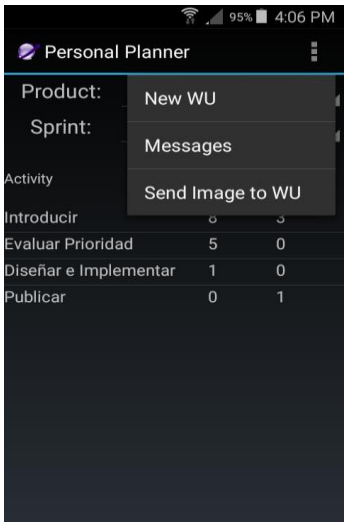


Figura A.1.3 Detalle de la barra de acción

Finalmente, al pulsar sobre cualquier fila del listado de actividades, accederemos a la siguiente interfaz (Figura A.1.4), donde obtendremos el listado detallado de UTs para la actividad seleccionada.

A.1.3 Listado de UTs

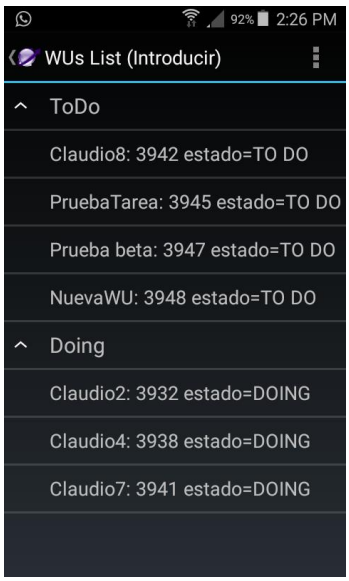


Figura A.1.4 Interfaz del Listado de UTs

Llegados a este punto, hemos obtenido el listado detallado de las UT's correspondientes a la actividad escogida en la pantalla anterior, tal y como podemos ver en la figura, clasificadas en sus dos grupos: pendientes (TO DO), y en progreso (DOING). Esto se muestra a través de una lista expandible, cuyos grupos raíz son los mencionados anteriormente, y sus hijos las UTs que les corresponden. Esta lista, que se muestra totalmente desplegada por defecto, puede contraerse y volverse a expandir de nuevo pinchando sobre cualquiera de los mencionados grupos raíz.

Al igual que en la interfaz anterior, tenemos la posibilidad de pulsar el botón de la barra de acción, para mostrar las mismas opciones con las mismas funcionalidades comentadas anteriormente.

Finalmente, una vez localizada la unidad de trabajo que nos interesa, accederemos a la siguiente interfaz (Figura A.1.5), donde podremos manipular su seguimiento.

A.1.4 Pantalla de control de tiempos

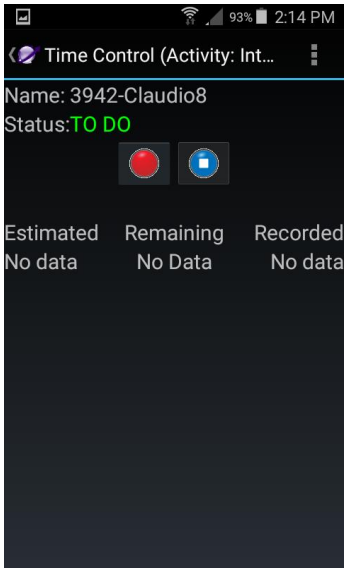


Figura A.1.5 IU de la pantalla de control de tiempos

Lo primero que nos encontramos al llegar a esta interfaz, es una pantalla en la que se nos muestra en la parte superior la actividad seleccionada inicialmente, el nombre de la UT cuyos tiempos se pretenden gestionar, y su estado actual. Adicionalmente, tenemos dos botones, uno de ellos con funcionalidad doble (ya que alterna entre los estados Record y Pause, que cambia su imagen en función de dichos estados), y un botón de STOP para dar por finalizada la UT. Finalmente, en el apartado inferior, tenemos detalles acerca del tiempo estimado (Estimated), tiempo restante (Remaining), y el tiempo total acumulado (Recorded).

Nótese que este listado de tiempos, se actualizará (como es obvio), cada vez que cambiemos del estado “Active”, a “Paused”.

Las posibilidades que tiene el usuario dentro de esta interfaz son las siguientes:

- **Pulsar el botón Record/Pause:** Este botón alterna entre esas dos opciones en función del estado actual de la UT (Record si su estado es Paused, y Pause si su estado es Active).

- **Pulsar el botón Stop:** Este botón dará por finalizado el seguimiento de registro de tiempos de esta UT, y cambiará el estado de la misma a "Done". Cabe destacar que este proceso es IRREVERSIBLE, y a partir de ese momento, ya no se podrán registrar nuevos seguimientos para esta UT, por lo tanto, al hacer la pulsación, se nos mostrará una advertencia donde será necesaria la confirmación de la acción (Figura A.1.6).

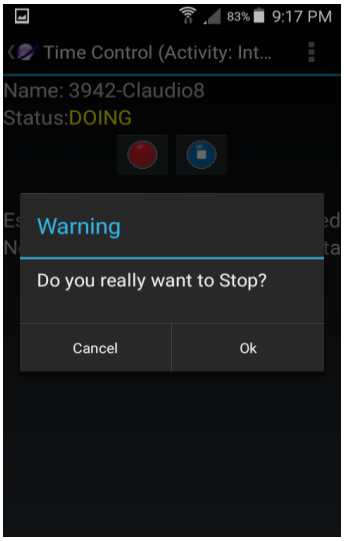


Figura A.1.6 Confirmación de finalización de seguimiento de la UT

Finalmente, y al igual que pasa con las interfaces anteriores, el usuario tiene la posibilidad de pulsar el botón de la barra de acción, donde tendrá las mismas opciones descritas en los apartados anteriores.

A.1.5 Creación de Nueva UT

La siguiente Figura (A.1.7), muestra la IU de la pantalla de creación de nuevas unidades de trabajo. El sistema de creación es muy sencillo: Introducimos un nombre en el campo destinado para ese fin, seleccionamos un Producto, un Workflow, y el Tipo pinchando sobre los menús desplegables, y una vez tengamos todas las opciones seleccionadas pulsamos sobre el botón Continue.

Figura A.1.7 Interfaz de usuario de la pantalla de creación de UT's

Se nos mostrará un cuadro de dialogo de confirmación con todas las opciones seleccionadas (Figura A.1.8), y, si todo es correcto, pulsaremos Ok y pasaremos a la siguiente pantalla donde configuraremos ciertos aspectos de la nueva UT (Figura A.1.9). Nótese no obstante, que tal y como se puede apreciar en la figura, al crear la nueva UT se lanza también una notificación a la barra de estado que nos advierte que la aplicación está contabilizando tiempos para esa nueva UT (tal y como sucede en la versión de PC's de escritorio).

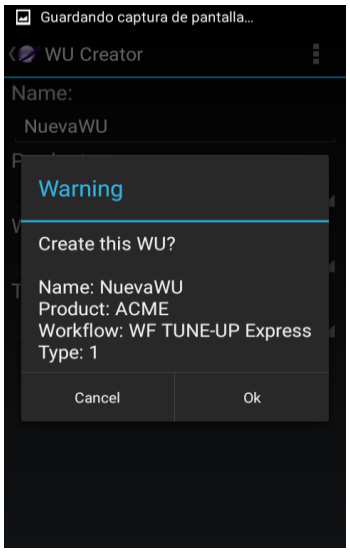


Figura A.1.8 Cuadro de confirmación de creación de nueva UT

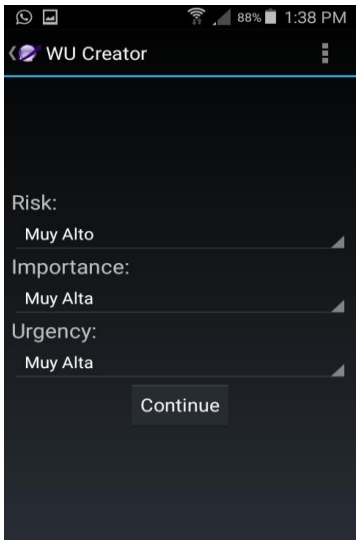


Figura A.1.9 Configuración de diferentes aspectos de la nueva UT

En la figura superior, tal y como podemos observar, se le permite al usuario, mediante tres menús desplegables, la opción de configurar los apartados Riesgo (Risk), Importancia (Importance), y Urgencia (Urgency) de la UT recién creada. Para ello, simplemente tiene que seleccionar dichos aspectos pinchando en los menús desplegables, y una vez los tenga seleccionados, pulsar sobre el botón "Continue".De este modo, se accederá directamente a la pantalla de control de tiempos para poder pausar, activar, o finalizar, el seguimiento de la nueva UT.

A.1.6 Mensajería

La función de mensajería está compuesta por dos interfaces (Figura A.1.10 y Figura A.1.11). Inicialmente lo que se muestra es el Administrador de Mensajes, donde lo que nos encontramos es una lista de 4 filas y dos columnas, donde se nos muestran los 4 tipos de mensajes que tiene la aplicación (Waiting Answer, I Must Read, I Must Answer y I'm Answering), y el número de mensajes pendientes que tenemos en cada una de estas cuatro categorías.

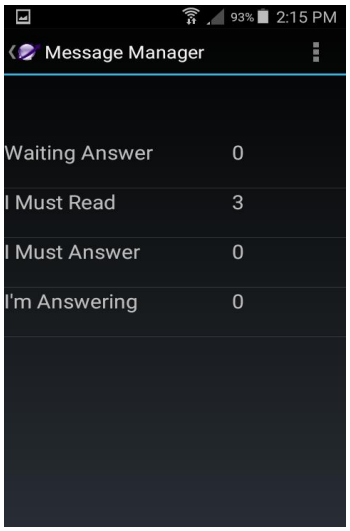


Figura A.1.10 Interfaz del administrador de mensajes

Al pinchar sobre cualquier de ellas (siempre y cuando tengamos como es obvio, mensajes pendientes en ella), accedemos a la siguiente pantalla, donde se nos muestra el listado de mensajes mediante una tabla con 3 columnas: Emisario (o receptor según el caso), del mensaje, Fecha, y mensaje en cuestión. Al pulsar sobre cualquiera de ellos (Figura A.1.12), se nos mostrará un cuadro de diálogo con detalles adicionales como el mensaje, la fecha de respuesta, la unidad de trabajo a la que pertenece, el nombre de dicha unidad, etc.

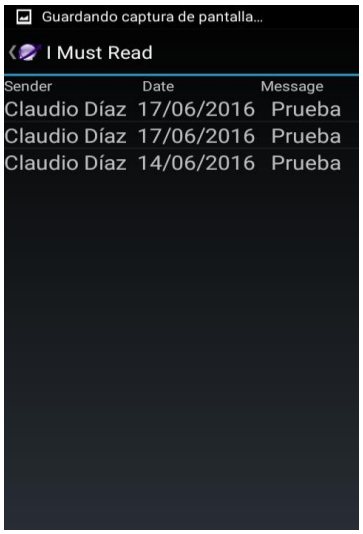


Figura A.1.11 Listado detallado de los mensaje

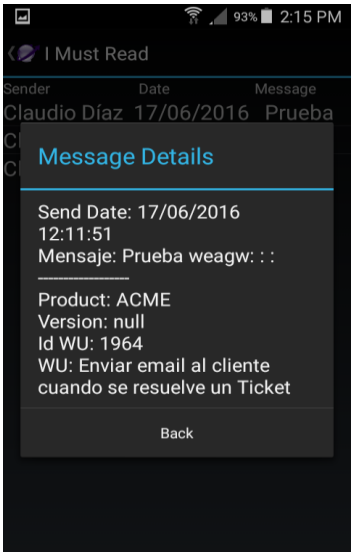


Figura A.1.12 Detalles del mensaje seleccionado

A.1.7 Envío de imágenes.

Por último nos queda por comentar la función de envío de imágenes al servidor de TUNE-UP.

Al pinchar sobre examinar, se abre el explorador de archivos con el cuál se podrá seleccionar la imagen que se desea enviar al servidor(Figura A.1.13).

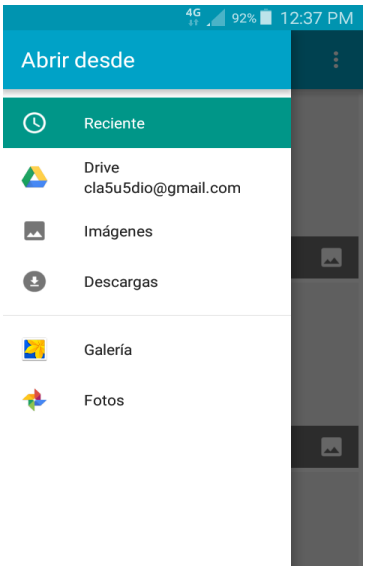


Figura A.1.13 Galería para seleccionar imagen.

Al pulsar el icono de la cámara se abrirá la cámara del teléfono (Figura A.1.14) permitiendo realizar una captura y luego enviarla.

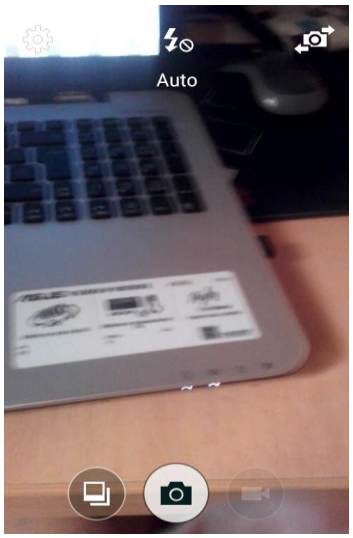


Figura A.1.14 Cámara invocada desde aplicación.

Al pulsar en enviar se enviará la imagen al servidor apareciendo un mensaje de confirmación del envío (Figura A.1.15), salvo que no se haya seleccionado archivo, o el archivo seleccionado no sea válido, en cuyo caso aparecerá mensaje de error.

Figura A.1.15 Interfaz de usuario de la pantalla de envio de imágenes.

A.2 Pruebas detalladas

A continuación se muestra el detalle de las 67 pruebas mencionadas en el capítulo 7, ordenadas por interfaz de usuario.

A.2.1 Pruebas para el Login

Visualizar campos Login

```
CONDICIONES EJECUCIÓN

PASOS
Acceder a la aplicación

RESULTADO ESPERADO
Se muestran los campos Login, Password, y botón continue
```

Pulsar continue con el Password en blanco

```
CONDICIONES EJECUCIÓN
Campo password en Blanco y Id con texto

PASOS

RESULTADO ESPERADO
Se muestra un error
```

Pulsar continue con el ID en blanco

```
CONDICIONES EJECUCIÓN
Campo Id en Blanco y Password con texto

PASOS

RESULTADO ESPERADO
Se muestra un error
```

Intentar continue con todos los campos en blanco

```
CONDICIONES EJECUCIÓN
Id y Password en blanco

PASOS

RESULTADO ESPERADO
Se muestra un error
```

Pulsar continue con un ID de usuario incorrecto

```
CONDICIONES EJECUCIÓN
Id de usuario incorrecto y password correcto

PASOS

RESULTADO ESPERADO
Se muestra un error
```

Pulsar continue con un Password incorrecto

CONDICIONES EJECUCIÓN
Id de usuario correcto y Password incorrecto

PASOS

RESULTADO ESPERADO
Se muestra un error

Pulsar continue con un ID y password incorrectos

CONDICIONES EJECUCIÓN
Id y password incorrectos

PASOS

RESULTADO ESPERADO
Se muestra un error

Desplegar el menú de sitios

CONDICIONES EJECUCIÓN
El Id y el password son correctos y el agente posee acceso a más de un sitio.

PASOS
Pulsar desplegable Sitios

RESULTADO ESPERADO
Se muestran los sitios activos

Seleccionar un Sitio

CONDICIONES EJECUCIÓN
Tener desplegado la lista de sitios

PASOS

RESULTADO ESPERADO
El sitio seleccionado se muestra en el desplegable

Pulsar Continue con ID y password correctos

CONDICIONES EJECUCIÓN
Id de Usuario Ok, Password Ok y sitio seleccionado

PASOS

RESULTADO ESPERADO
Se accede a la aplicación

A.2.2 Pruebas para el Planificador Personal

Visualizar Listado de actividades

```
CONDICIONES EJECUCIÓN

PASOS
Acceder a la IU

RESULTADO ESPERADO
Visualizar el listado de Actividades
```

Pulsar sobre el filtro de productos

```
CONDICIONES EJECUCIÓN

PASOS
Pulsar sobre el desplegable de productos

RESULTADO ESPERADO
Se muestra el listado de productos disponibles
```

Seleccionar un producto desde el filtro

```
CONDICIONES EJECUCIÓN
Tener abierto el desplegable de productos

PASOS

RESULTADO ESPERADO
Se actualiza el desplegable de versiones y el listado de actividades
```

Pulsar sobre el filtro de Versiones

```
CONDICIONES EJECUCIÓN

PASOS
Pulsar sobre el desplegable de Versiones

RESULTADO ESPERADO
Se muestra el listado de versiones disponibles
```

Seleccionar una versión desde del filtro

```
CONDICIONES EJECUCIÓN
Tener el desplegable de versiones abierto

PASOS

RESULTADO ESPERADO
Se actualiza el desplegable de versiones y el listado de actividades
```

Pulsar sobre una actividad

```
CONDICIONES EJECUCIÓN

PASOS
Seleccionar una actividad del listado

RESULTADO ESPERADO
Acceder a la IU del listado de UTs de esa actividad
```

Pulsar Barra de Acción

```
CONDICIONES EJECUCIÓN

PASOS
Pulsar barra de acción

RESULTADO ESPERADO
Se abren las opciones de la barra de acción
```

Pulsar NewWU

```
CONDICIONES EJECUCIÓN

PASOS
Pulsar NewWU

RESULTADO ESPERADO
Se accede a la IU de creacion de UTs
```

Pulsar Message Manager

```
CONDICIONES EJECUCIÓN

PASOS
Pulsar Message Manager

RESULTADO ESPERADO
Se accede a la lista del Administrador de Mensajes
```

A.2.3 Pruebas para el Listado de UTs

Visualización Lista de UT's

```
CONDICIONES EJECUCIÓN

PASOS
Acceder a la IU

RESULTADO ESPERADO
Se muestra la lista con todas las UT's ordenadas por grupos TO DO y DOING
```

Contraer Grupo TO DO

CONDICIONES EJECUCIÓN
La lista del grupo debe estar expandida

PASOS
Pulsar en la raiz del grupo TO DO

RESULTADO ESPERADO
Se contrae la lista de UTs para ese grupo

Expandir Grupo TO DO

CONDICIONES EJECUCIÓN
La lista del grupo debe estar contraida

PASOS
Pulsar en la raiz del grupo TO DO

RESULTADO ESPERADO
La lista relacionada con ese grupo se expande

Contraer Grupo Doing

CONDICIONES EJECUCIÓN
La lista del grupo debe estar expandida

PASOS
Pulsar en la raiz del grupo DOING

RESULTADO ESPERADO
Se contrae la lista de UTs para ese grupo

Expandir Grupo Doing

CONDICIONES EJECUCIÓN
La lista del grupo debe estar contraida

PASOS
Pulsar en la raiz del grupo Doing

RESULTADO ESPERADO
La lista relacionada con ese grupo se expande

Seleccionar UT

CONDICIONES EJECUCIÓN

PASOS
Hacer click en una UT

RESULTADO ESPERADO
Acceso a la pantalla de Time Control

Pulsar Barra de Acción

CONDICIONES EJECUCIÓN

PASOS
Pulsar Barra de Acción

RESULTADO ESPERADO
Se abren las opciones de la barra de acción

Pulsar New UT

CONDICIONES EJECUCIÓN
Tener la barra de acción abierta

PASOS
Pulsar New UT

RESULTADO ESPERADO
Se accede a la IU de creación de UTs

Pulsar Message Manager

CONDICIONES EJECUCIÓN
Tener la barra de acción abierta

PASOS
Pulsar Message Manager

RESULTADO ESPERADO
Se accede a la IU del administrador de Mensajes

A.2.4 Pruebas para Time Control

Visualización Status ACTIVE

CONDICIONES EJECUCIÓN
Estado del seguimiento ACTIVE

PASOS
Acceder a la IU

RESULTADO ESPERADO
Se visualiza Status ACTIVE y el botón de pausa en lugar del botón Record, y el botón de Finish

Visualización Status TO DO

CONDICIONES EJECUCIÓN
Estado del seguimiento TO DO

PASOS
Acceder a la IU

RESULTADO ESPERADO
Se visualiza Status TO DO y el botón de Record y el de Finish

Visualización de resumen de tiempos

```
CONDICIONES EJECUCIÓN

PASOS
Acceder a la IU

RESULTADO ESPERADO
Se visualizan los tiempos Estimated, Remaining y Recorded
En caso que algún valor no tenga registro, debe aparecer NO DATA
```

Presionar botón Record

```
CONDICIONES EJECUCIÓN
El Status es TO DO o PAUSED

PASOS
Presionar el botón Record

RESULTADO ESPERADO
- El botón Record cambia a botón Pause
```

Visualización Status PAUSE

```
CONDICIONES EJECUCIÓN
Estado del seguimiento PAUSED

PASOS
Acceder a la IU

RESULTADO ESPERADO
Se visualiza Status PAUSED y el botón de Record en lugar del botón Pause, y el botón
de Finish
```

Visualización Status FINISHED

```
CONDICIONES EJECUCIÓN
Estado del seguimiento FINISHED

PASOS
Acceder a la IU

RESULTADO ESPERADO
Se visualiza Status FINISHED y el botón de Record/Pausa y el botón de Finish
deshabilitados
```

Presionar Botón Pause

```
CONDICIONES EJECUCIÓN
El Status es ACTIVE

PASOS
Presionar el botón Pause

RESULTADO ESPERADO
 - El botón Pause cambia a botón Record
 - Se actualizan los tiempos
```

Presionar Botón Finish

```
CONDICIONES EJECUCIÓN
El Status es ACTIVE o PAUSED

PASOS
Presionar el botón FINISH

RESULTADO ESPERADO
 - Los botones Record/Pause y Finish se deshabilitan
 - Se cambia el estado a Done
```

Pulsar la tecla de la Barra de Acción

```
CONDICIONES EJECUCIÓN
Estar dentro de la pantalla de Time Control

PASOS
Pulsar Barra de Acción

RESULTADO ESPERADO
Se abre la barra de acción mostrando la posibilidad de Crear nueva UT o acceder al
Administrador de Mensajes
```

Pulsar la opción NewUT de la barra de acción

```
CONDICIONES EJECUCIÓN

PASOS
Pulsar botón barra de acción
Pulsar NewUT

RESULTADO ESPERADO
Acceder a la IU de creación de UTs
```

Pulsar la opción Message Manager de la barra de acción

```
CONDICIONES EJECUCIÓN

PASOS
Pulsar botón de barra de acción
Pulsar MessageManager

RESULTADO ESPERADO
Acceder a la IU del Administrador de Mensajes
```

A.2.5 Pruebas para la Interfaz de Creación de UTs

Visualización campos NewUT

```
CONDICIONES EJECUCIÓN

PASOS
Acceder a la IU

RESULTADO ESPERADO
Se muestran los 3 controles desplegables rellenados con sus datos y el botón
continuar
```

Pulsar Desplegable Product

```
CONDICIONES EJECUCIÓN

PASOS
Hacer click sobre el desplegable Product

RESULTADO ESPERADO
Se muestra la lista de Productos disponibles
```

Seleccionar un Producto

```
CONDICIONES EJECUCIÓN
Tener la lista de Productos disponibles desplegada

PASOS

RESULTADO ESPERADO
Se cierra la lista y aparece en el desplegable el Producto seleccionado
```

Pulsar Desplegable WorkFlow

```
CONDICIONES EJECUCIÓN

PASOS
Hacer click sobre el desplegable WorkFlow

RESULTADO ESPERADO
Se muestra la lista de Workflows disponibles
```

Pulsar Desplegable Riesgo

```
CONDICIONES EJECUCIÓN

PASOS
Hacer click sobre el desplegable Riesgo

RESULTADO ESPERADO
Se muestra la lista de Riesgos disponibles
```

Pulsar Desplegable Importancia

```
CONDICIONES EJECUCIÓN

PASOS
Hacer click sobre el desplegable Importancia

RESULTADO ESPERADO
Se muestra la lista de Importancias disponibles
```

Pulsar Desplegable Urgencia

```
CONDICIONES EJECUCIÓN

PASOS
Hacer click sobre el desplegable Urgencia

RESULTADO ESPERADO
Se muestra la lista de Urgencias disponibles
```

Seleccionar un Workflow

```
CONDICIONES EJECUCIÓN
Tener la lista de Workflows disponibles desplegada

PASOS

RESULTADO ESPERADO
Se cierra la lista y aparece en el desplegable el Workflow seleccionado
```

Seleccionar un Tipo

```
CONDICIONES EJECUCIÓN
Tener la lista de Tipos disponibles desplegada

PASOS

RESULTADO ESPERADO
Se cierra la lista y aparece en el desplegable el Tipo seleccionado
```

Pulsar Continue con el Nombre en blanco

```
CONDICIONES EJECUCIÓN
El campo Nombre debe estar en blanco

PASOS
Hacer click sobre el boton Continue

RESULTADO ESPERADO
Se muestra un error
```

Pulsar Continue con el campo Nombre rellenado

```
CONDICIONES EJECUCIÓN
El campo Nombre debe contener un texto

PASOS
Hacer click en el botón continue

RESULTADO ESPERADO
Se accede a la configuración de la nueva UT
```

Seleccionar un Riesgo

```
CONDICIONES EJECUCIÓN
Tener el desplegable de riesgos abierto

PASOS

RESULTADO ESPERADO
Se cierra la lista y aparece seleccionado el riesgo en el desplegable
```

Seleccionar una importancia

```
CONDICIONES EJECUCIÓN
Tener el desplegable de Importancias abierto

PASOS

RESULTADO ESPERADO
Se cierra la lista y aparece seleccionado la importancia en el desplegable
```

Seleccionar una Urgencia

```
CONDICIONES EJECUCIÓN
Tener el desplegable de urgencias abierto

PASOS

RESULTADO ESPERADO
Se cierra la lista y aparece seleccionado la urgencia en el desplegable
```

A.2.6 Pruebas para la Interfaz del Administrador de Mensajes

Visualización Categorías

```
CONDICIONES EJECUCIÓN


PASOS
Entrar en la UI

RESULTADO ESPERADO
Comprobar que se muestran las 4 categorias de mensajes con su numeracion
correspondiente
```

Comprobar NO entrada campos con 0 mensajes

```
CONDICIONES EJECUCIÓN
Nº mensajes de la categoria = 0

PASOS
Hacer click en cualquier categoria

RESULTADO ESPERADO
No hace nada (no accede al interfaz de detalles de los mensajes)
```

Seleccionar categoria de mensajes con nº mensajes >0

```
CONDICIONES EJECUCIÓN
Nº mensajes de la categoria >0

PASOS
Hacer click en cualquier categoria

RESULTADO ESPERADO
Accede al interfaz de detalle de los mensajes
```

A.2.7 Pruebas para los detalles de los mensajes

Visualización Detalles

```
CONDICIONES EJECUCIÓN

PASOS
Acceder a la interfaz

RESULTADO ESPERADO
Se muestra el listado de categorias con los campos Emisor/Receptor, Fecha y Mensaje
```

Comprobar que muestra Emisor si el mensaje es recibido

```
CONDICIONES EJECUCIÓN
Acceder a la interfaz en la categoria Waiting Answer o I Must Read

PASOS

RESULTADO ESPERADO
La primera columna muestra SENDER
```

Comprobar que muestra Receptor si se accede a un mensaje enviado por nosotros

```
CONDICIONES EJECUCIÓN
Acceder a la interfaz a través de la categoria I Must Answer o I'm Answering

PASOS

RESULTADO ESPERADO
La primera columna muestra "Receiver"
```

Pulsar sobre cualquier mensaje

```
CONDICIONES EJECUCIÓN

PASOS
Pulsar sobre cualquier mensaje

RESULTADO ESPERADO
Se muestra un cuadro de dialogo con los detalles de los mensajes
```

A.2.8 Pruebas para el envío de imágenes

Visualización

CONDICIONES EJECUCIÓN

PASOS
Entrar en la UI

RESULTADO ESPERADO
Comprobar que se muestran el spinner de las unidades de trabajo y los botones de examinar, cámara y envío de la imagen.

Pulsar Desplegable UT

CONDICIONES EJECUCIÓN

PASOS
Hacer click sobre el desplegable UT

RESULTADO ESPERADO
Se muestra la lista de UT asociadas al agente.

Seleccionar una UT

CONDICIONES EJECUCIÓN
Haber desplegado el desplegable de UTs.

PASOS
Hacer click sobre una UT del desplegable de UTs.

RESULTADO ESPERADO
Se selecciona la UT clickada.

Pulsar botón galería UT

CONDICIONES EJECUCIÓN

PASOS
Hacer click sobre el botón Examinar

RESULTADO ESPERADO
Se muestra un explorador de imágenes del terminal.

Seleccionar botón cámara

CONDICIONES EJECUCIÓN

PASOS
Hacer click sobre el botón Cámara

RESULTADO ESPERADO
Se abre la cámara del terminal que permite capturar una foto.

Pulsar enviar sin seleccionar/capturar imagen

CONDICIONES EJECUCIÓN
Ninguna imagen seleccionada/capturada.

PASOS
Pulsar el botón enviar.

RESULTADO ESPERADO
Aparece mensaje de error.

Pulsar enviar con imagen seleccionada/capturada

CONDICIONES EJECUCIÓN
Tener una imagen seleccionada/capturada.

PASOS
Pulsar el botón enviar.

RESULTADO ESPERADO
Aparece mensaje de éxito en el envío y se carga la imagen al servidor.