Universidad del País Vasco
Euskal Herriko Unibertsitatea

Escuela Universitaria de Ingeniería Vitoria-Gasteiz
Ingeniaritzako Unibertsitate Eskola Vitoria-Gasteiz

Extensión móvil para sistemas de ventas propietarios (Intraza)

Juan Ignacio Cuerno Lagos

12 de junio de 2018

Índice general

Índice de figuras

Parte I.

Introducción

1 Descripción del proyecto

Esta memoria documenta el trabajo realizado como freelance para desarrollar una aplicación que se detalla más adelante. Para este TFG se han realizado modificaciones al código para eliminar elementos creados específicamente para este cliente, con la finalidad de mostrar más claramente las características de la aplicación. El desarrollo del proyecto siguió los pasos que se han descrito en este documento. [J.I.Cuerno(2018b)]

Intraza es una aplicación Android desarrollada para PYMES que quieren dotar de nuevas funcionalidades móviles a su sistema de pedidos sin tener que realizar grandes modificaciones ni una inversión costosa. Se integra con su sistema de pedidos sin necesidad de realizar cambios en la aplicación, tras un análisis del funcionamiento de su sistema de ventas. En el ejemplo que nos ocupa la aplicación fue desarrollada para una empresa de distribución cárnica de Mallorca, que disponía de un ERP desarrollado por TECHNICALNORMS SL.

Intraza está orientada a servir como herramienta portátil para realizar pedidos por parte del área comercial de la PYME. Los comerciales disponen de los datos necesarios para tramitar pedidos en una Tableta con sistema Android y guardarlos para después enviarlos al sistema de pre-pedidos de la aplicación. Uno de los requisitos planteados por la especial orografía de Mallorca fue la posibilidad de generar los pedidos offline y poder enviarlos cuando hubiese posibilidad de conexión, ya fuese por conexión wifi o 3G.

Cabe destacar que a día de hoy sociedad que dirigía la empresa de distribución cárnica, está disuelta y tal ruptura ha generado empresas paralelas. Estas han roto la relación contractual con TECHNICALNORMS SL. Los acuerdos realizados con TECHNICALNORMS SL para el mantenimiento de la misma quedaron disueltos.

El autor de este proyecto es propietario de los derechos y se presenta simpli-

ficado, eliminando funcionalidad específica de TECHNICALNORMS SL. Para poder ejemplificar su funcionamiento se ha creado un ERP simple desarrollado con metodologías ágiles, cuya funcionalidad no forma parte de este proyecto.

Intraza se compone de dos módulos diferenciados.

* Por un lado, un servicio web JAVA - REST que hará de interfaz de comunicación entre la app Android y la base de datos del ERP. Este módulo contiene las consultas necesarias para alimentar la base de datos interna de la app, además de los procesos necesarios para enviar los pre-pedidos a la herramienta.
* Por otro lado, una aplicación Android, con una base de datos interna con datos de clientes, productos, e histórico de pedidos, para poder realizar pedidos in situ.

1.1. Contexto

Dado que el proyecto se basa en la adaptación a un sistema ERP, lo primero que se ha de documentar es en qué se basa la tecnología ERP.

Los sistemas ERP (Planificación de Recursos Empresariales) son sistemas de gestión de información que integran y automatizan muchas de las prácticas de negocio asociadas con los aspectos operativos, productivos o de distribución de una empresa o compañía comprometida en la producción de bienes o servicios. Se caracterizan por estar compuestos por diferentes partes integradas en una única aplicación, estas partes suelen corresponder a cada uno de los diferentes departamentos de una empresa, por ejemplo: producción, ventas, compras, contabilidad, RRHH, logística, etc.

Los objetivos principales de los sistemas ERP son:

- Optimización de los procesos empresariales.
- Acceso a toda la información de forma confiable, precisa y oportuna (integridad de datos).
- La posibilidad de compartir información entre todos los componentes de la organización.
- Eliminación de datos y operaciones innecesarias de re-ingeniería.

El propósito fundamental de un ERP es otorgar apoyo a los clientes del negocio, tiempos rápidos de respuesta a sus problemas, así como un eficiente manejo de información que permita la toma oportuna de decisiones y disminución de los costos totales de operación.

Los ERP son sistemas software que se caracterizan por ser modulares, inte-

grales y adaptables a las necesidades del cliente. Además un ERP no es solo una aplicación sino que requiere de un equipo técnico para darle soporte según las necesidades del cliente.

1.2. Objetivos del proyecto

El objetivo de esta aplicación, es ser una herramienta portátil, sencilla de gestionar y visualmente atractiva, que agilice el proceso de pedido por parte de los comerciales cuando realizan visitas al cliente. El objetivo como diseñador y desarrollador de un producto, es satisfacer las necesidades del cliente, priorizando la sencillez y una estética depurada, con tal de conseguir una herramienta fácil de usar por los usuarios, además de ser una tarjeta de presentación del saber hacer de la PYME.

1.3. Cómo surgió la idea

La idea de crear esta aplicación me fue transmitida por TECHNICALNORMS SL. La empresa solicitante buscaba dar soporte a una pequeña parte de su negocio, clientes menores, a través de desarrollos puntuales, ya sean como outsourcing o con desarrolladores Freelance. La empresa cliente plantea la mejora de una herramienta en una versión obsoleta. Tras el análisis y viendo que tenía varios errores de concepto. Se plantea el desarrollo de una aplicación nueva con el conocimiento adquirido, de la anterior, y mejorando el rendimiento de la misma. Este proyecto se desarrolló durante el 1º semestre del año 2015.

Parte II.

Viabilidad

2 Análisis de riesgos

El análisis de riesgos es el proceso que permite la identificación de las amenazas que pueden perjudicar al desarrollo del proyecto y determinar el impacto o grado de perjuicio que pueden ocasionar. Implica analizar las amenazas y vulnerabilidades que puedan darse y se valoran en términos de probabilidad de ocurrencia y gravedad de impacto sobre el proyecto.

A continuación, se establece una escala (de menor a mayor gravedad) para valorar el impacto que causarían los riesgos que se describen a continuación:

- ***Muy grave:*** De consecuencias fatales para el desarrollo del proyecto, requeriría replantearse, incluso, si merece la pena continuar con el proyecto.
- ***Grave:*** Problemas que podrían imposibilitar terminar el proyecto en el plazo previsto. Requeriría realizar una profunda re planificación ampliando el número de horas a dedicar o reduciendo el trabajo a realizar.
- ***Perjudicial:*** Pequeños retrasos relativamente sencillos de solventar, aunque pueden resultar peligrosos si no se identifican y subsanan a tiempo.

2.1. Fichas de riesgo

Los posibles riesgos comunes que se tienen en cuenta en la realización del proyecto los siguientes:

Elección equivocada de las herramientas de trabajo:

Descripción: Es posible que con alguna de las tecnologías elegidas no sea posible cumplir los objetivos planteados o dar a la aplicación la funcionalidad deseada.
Probabilidad: 5
Alcance: Muy grave, ya que supondría replantear por completo el proyecto.
Medidas preventivas: no es necesario adoptar medidas preventivas ya que por requisitos tecnológicos las herramientas de trabajo estaban muy definidas.

Desconocimiento del software elegido:

Descripción: En este proyecto se van a usar varias herramientas con las que el desarrollador no está nada familiarizado y esto podría ocasionar retrasos a la hora de realizar la aplicación.
Probabilidad: 90
Alcance: Perjudicial, impediría avanzar en el tiempo estimado.
Medidas preventivas: no es necesario adoptar medidas preventivas ya que el desarrollador conoce las herramientas y tiene experiencia con las mismas.

Estancamiento en la codificación:

Descripción: Podría llegar a darse algún problema de difícil solución que paralizaría momentáneamente el avance del proyecto.
Probabilidad: 20
Alcance: Perjudicial, impediría avanzar en el tiempo estimado.
Medidas preventivas: Medidas preventivas: Se Mantiene un seguimiento por medio de metodologías ágiles, así pues, el cliente sabe en todo momento cómo va el desarrollo del trabajo a través de los Sprints

Pérdida de los datos por fallo del hardware o software maligno:

Descripción: Debido a un virus o a un fallo en el disco duro del ordenador en el que se desarrolla el proyecto se podrían perder todos los datos.
Probabilidad: 10
Alcance: Muy grave, pues supondría empezar de cero.
Medidas preventivas: Se ha implantado un sistema de control de versiones (git) y una copia de seguridad del entorno de desarrollo, ademas de herramientas anti-virus y anti-malware.

Problema con los recursos materiales:

Descripción: El coste material y del software no puede ser asumido.
Probabilidad: 10
Alcance: Perjudicial, ya que retrasarían el proyecto de una forma indefinida.
Medidas preventivas: El material adicional necesario es proporcionado por la pyme en modelo de préstamo. Las licencias de desarrollo de los sistemas elegidos son de libre uso y distribución.

Baja del desarrollador de la aplicación:

Descripción: Podría suceder que, por enfermedad o accidente, el desarrollador del proyecto quede incapacitado durante una temporada.
Probabilidad: 5
Alcance: Perjudicial, puesto que impide la realización del proyecto.
Medidas preventivas: No aplicables.

3 Planificación del tiempo del proyecto

Este apartado describe la estructura de descomposición del trabajo utilizada en la realización de la aplicación, las fases, tareas y entregables del proyecto, así como la agenda de trabajo y los recursos materiales empleados. Hay que tener en cuenta que vamos a utilizar una metodología de gestión ágil. [P. Deemer and Vodde.(2013)]

3.1. Descripción General de la Metodología

3.1.1. Fundamentación

Las principales razones del uso de un ciclo de desarrollo iterativo e incremental de tipo SCRUM para la ejecución de este proyecto son:

- Sistema modular. Las características del sistema Intraza permiten desarrollar una base funcional mínima y sobre ella ir incrementando las funcionalidades o modificando el comportamiento o apariencia de las ya implementadas.
- Entregas frecuentes y continuas al cliente de los módulos terminados, de forma que puede disponer de una funcionalidad básica en un tiempo mínimo y a partir de ahí un incremento y mejora continua del sistema.
    - Previsible inestabilidad de requisitos.
    - Es posible que el sistema incorpore más funcionalidades de las inicialmente identificadas.
    - Es posible que durante la ejecución del proyecto se altere el orden en el que se desean recibir los módulos o historias de usuario terminadas.
    - Al cliente le resulta difícil precisar la dimensión completa del sistema, y su crecimiento puede continuarse en el tiempo suspenderse o detenerse.

3.1.2. Artefactos

- Documentos
  - Pila de producto
  - Pila de Sprint
- Gráficas para registro y seguimiento del avance.
  - Gráfica de producto
  - Gráfica de avance
- Comunicación Reunión de inicio de Sprint
  - Reunión técnica diaria
  - Reunión de cierre de Sprint y entrega del incremento

3.1.2.1. Pila de producto

Es el equivalente a los requisitos del sistema o del usuario en esta metodología. El gestor de producto puede recabar las consultas y asesoramiento que pueda necesitar para su redacción y gestión durante el proyecto al SCRUM MASTER de este proyecto.

- Responsabilidades del gestor de producto
  - Registro en la lista de pila del producto de las historias de usuario que definen el sistema.
  - Mantenimiento actualizado de la pila del producto en todo momento durante la ejecución del proyecto.
    - Orden en el que desea recibir terminada cada historia de usuario.
    - Incorporación / eliminación /modificaciones de las historias o de su orden de prioridad.
    - Disponibilidad: mantiene directamente la pizarra o intranet o medios de comunicación.
- Responsabilidades del SCRUM MASTER
  - Supervisión de la pila de producto, y comunicación con el gestor del producto para pedirle aclaración de las dudas que pueda tener, o asesorarle para la subsanación de las deficiencias que observe.
- Responsabilidades del equipo técnico
  - Conocimiento y comprensión actualizado de la pila del producto.
  - Resolución de dudas o comunicación de sugerencias con gestor de pro-

ducto / SCRUM MASTER.

- Responsabilidades del resto de implicados
  - Conocimiento y comprensión actualizado de la pila del producto.
  - Resolución de dudas o comunicación de sugerencias con gestor de producto o SCRUM MASTER.

3.1.2.2. Pila del Sprint

Es el documento de registro de los requisitos detallados o tareas que va a desarrollar el equipo técnico en la iteración (actual o que está preparándose para comenzar)

- Responsabilidades del gestor de producto
  - Presencia en las reuniones en las que el equipo elabora la pila del Sprint. Resolución de dudas sobre las historias de usuario que se descomponen en la pila del Sprint.
- Responsabilidades del SCRUM MASTER
  - Supervisión y asesoría en la elaboración de la pila del Sprint.
- Responsabilidades del equipo técnico
  - Elaboración de la pila del Sprint
  - Resolución de dudas o comunicación de sugerencias sobre las historias de usuario con el gestor del producto.

3.1.2.3. Sprint

Cada una de las iteraciones del ciclo de vida iterativo SCRUM. La duración de cada Sprint es en el caso que nos ocupa 2 semanas de 5 días laborables cada una.

3.1.2.4. Gráfica de producto (Burn Up)

Representación gráfica del plan de producto previsto por el gestor de producto. Es una gráfica que representa los temas o epics del sistema en el orden que se desean, y el tiempo en el que se prevé su ejecución.

- Responsabilidades del gestor de producto
  - Confección.

- Mantenimiento actualizado en todo momento durante la ejecución del proyecto.
- Orden en el que desea disponer de los temas o “epics” del sistema, e hitos del producto (versiones).
- Incorporación / eliminación /modificaciones de los temas, de su orden de prioridad, estimaciones o hitos.
- Disponibilidad: mantiene directamente la pizarra o intranet o medios de comunicación.

- Responsabilidades del SCRUM MASTER
  - Supervisión del gráfico de producto, y comunicación con el gestor del producto para pedirle aclaración de las dudas que pueda tener, o asesorarle para la subsanación de las deficiencias que observe.
- Responsabilidades del equipo técnico
  - Conocimiento y comprensión actualizado del plan del producto.
  - Resolución de dudas o comunicación de sugerencias con gestor de producto o SCRUM MASTER.
- Responsabilidades del resto de implicados
  - Conocimiento y comprensión actualizado del plan de producto.
  - Resolución de dudas o comunicación de sugerencias con gestor de producto o SCRUM MASTER.

3.1.2.5. Gráfica de avance (Burn Down)

Gráfico que muestra el estado de avance del trabajo del Sprint en curso.

- Responsabilidades del gestor de producto
  - Sin responsabilidades específicas, más allá de mantenerse regularmente informado del avance del Sprint y disponible para atender decisiones para la resolución de opciones en Sprints sobre-valorados o infravalorados (la gráfica de avance predice una entrega anterior o posterior a la fecha prevista)
- Responsabilidades del SCRUM MASTER
  - Supervisión de la actualización diaria por parte del equipo.
- responsabilidades del equipo técnico
  - Actualización diaria del gráfico de avance.

3.1.2.6. Reunión de inicio de Sprint

Reunión para determinar las funcionalidades o historias de usuario que se van a incluir en el próximo incremento.

- Responsabilidades del gestor de producto
  - Asistencia a la reunión.
  - Exposición y explicación de las historias que necesita para la próxima iteración y posibles restricciones de fechas que pudiera tener.
- Responsabilidades del SCRUM MASTER
  - Moderación de la reunión
- Responsabilidades del equipo técnico
  - Confección de la pila del Sprint.
  - Auto-asignación del trabajo.

3.1.2.7. Reunión técnica diaria

Puesta en común diaria del equipo con presencia del Coordinador del proyecto o SCRUM MASTER de duración máxima de 10 minutos.

- Responsabilidades del SCRUM MASTER
  - Supervisión de la reunión y anotación de las necesidades o impedimentos que pueda detectar el equipo.
  - Gestión para la solución de las necesidades o impedimentos detectados por el equipo.
- Responsabilidades del equipo técnico
  - Comunicación individual del trabajo realizado el día anterior y el previsto para día actual.
  - Actualización individual del trabajo pendiente.
  - Actualización del gráfico de avance para reflejar el estado de avance.
  - Notificación de necesidades o impedimentos previstos u ocurridos para realizar las tareas asignadas.

3.1.2.8. Reunión de cierre de Sprint y entrega del incremento

Reunión para probar y entregar el incremento al gestor del producto.

Características.

- Prácticas: sobre el producto terminado, no sobre simulaciones o imágenes.
- De tiempo acotado máximo de 2 horas.
- Responsabilidades del gestor de producto
  - Asistencia a la reunión.
  - Recepción del producto o presentación de reparos.
- Responsabilidades del SCRUM MASTER
  - Moderación de la reunión
- Responsabilidades del equipo técnico
  - Presentación del incremento.

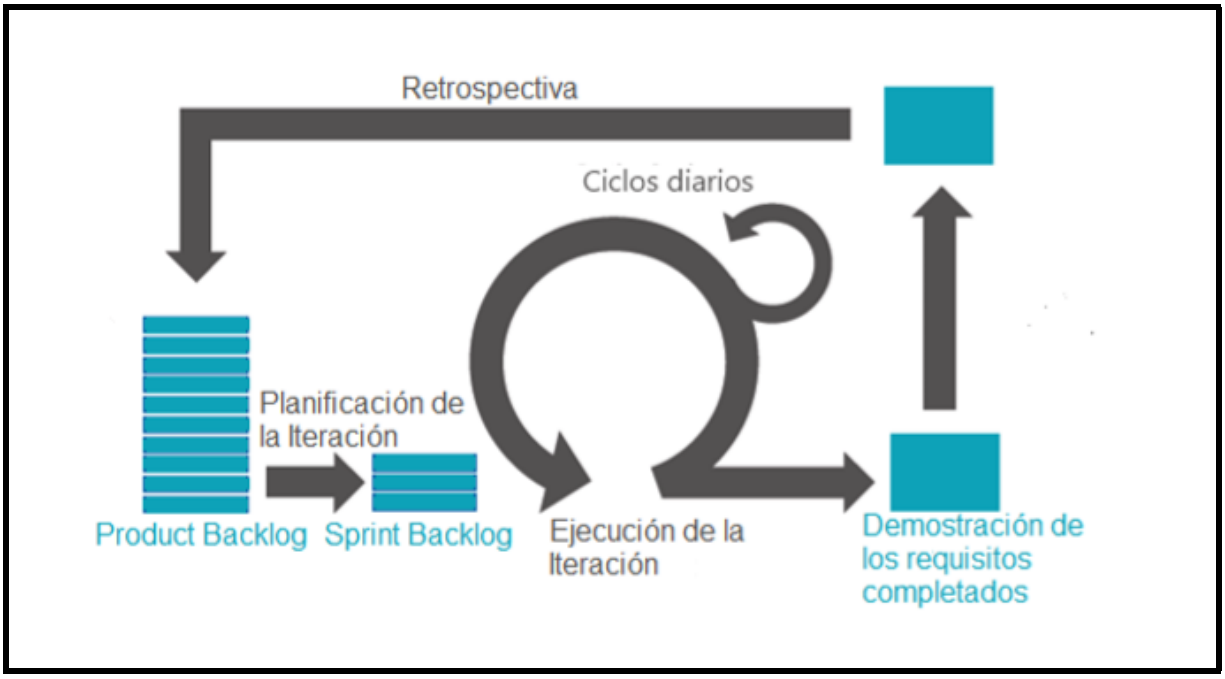

Figura 3.1.: Ciclo del proyecto

3.1.3. Particularidades de este proyecto

En este proyecto la figura del SCRUM MASTER y el equipo técnico recaen sobre una sola persona. Bajo esta particularidad desaparecen las reuniones técnicas diarias. La figura del gestor del producto viene dada por un contacto con la empresa cliente. Las reuniones de comienzo del Sprint y final de Sprint se realizan bajo herramientas de videoconferencia, y teamviewer.

3.2. Historias de usuario

Las historias de usuario, son pequeñas descripciones de los requerimientos de un cliente. Su utilización es común cuando se aplica marcos de entornos ágiles como SCRUM.

Al redactar las historias de usuario se debe tener en cuenta describir el Rol, la funcionalidad y el resultado esperado en una frase corta. Debe venir acompañada de los criterios de aceptación.

Es deseable que las historias de usuario sean escritas por el usuario, en una frase corta. Debe describir el rol desempeñado por el usuario de forma explícita e indicar el beneficio para el área de negocio que representa esta funcionalidad.

3.2.1. Listado de historias de usuario

Historia 1:

Como: Cliente.
Quiero: Analizar el know how de la empresa.
Para: Estudiar la mejor forma de adaptar los requisitos a la forma de trabajar de la empresa.

Historia 2:

Como: Cliente.
Quiero: Disponer de un listado de clientes offline.
Para: Tener a disposición los clientes sin necesidad de estar conectado.

Historia 3:

Como: Cliente.
Quiero: Disponer de un listado de productos offline.
Para: Tener a disposición los productos sin necesidad de estar conectado.

Historia 4:

Como: Cliente.
Quiero: Disponer de un listado de pedidos por cliente offline.
Para: Tener a disposición los pedidos realizados por un cliente sin necesidad de estar conectado.

Historia 5:

Como: Cliente.
Quiero: Poder crear pedidos por cliente.
Para: Gestionar en la aplicación los pedidos y tenerlos almacenados.

Historia 6:

Como: Cliente.
Quiero: Poder editar los pedidos no enviados.
Para: Realizar correcciones antes de enviar los pedidos.

Historia 7:

Como: Cliente.
Quiero: Tener listados de productos por pedido.
Para: Visualizar los productos y gestionar filtros.

Historia 8:

Como: Cliente.
Quiero: Poder enviar los pedidos realizados a la central.
Para: Disponer de una interface que comunique con la la aplicación de gestión.

Historia 9:

Como: Cliente.
Quiero: Completa abstracción de la app vs aplicación de gestión.
Para: No modificar en la medida de lo posible el producto usado hasta ahora, y mantener la forma de trabajar.

Historia 10:

Como: Cliente.
Quiero: probar la funcionalidad.
Para: tener un tiempo de pruebas para verificar la funcionalidad de la app.

3.3. Product Backlog

Se estima que la realización del proyecto tendrá una estimación de 5 Iteraciones (2 semanas por iteración) para la fase de desarrollo y 1 Iteración adicional para testeo y pruebas finales con el cliente. Se ha concretado con el cliente unas funcionalidades a través de las historias de usuario. Dejando para una fase posterior, fuera de esta planificación y por ende fuera de este proyecto, Modificaciones adicionales o sugerencias por parte del cliente.

La jornada diaria tendrá un valor de 1 punto correspondiendo este a una jornada laboral de 8h/dia. Así pues una iteración dispondrá de 10 puntos, que equivalen a los 10 dias laborables incluidos en las dos semanas descritas.

A continuación en la Tabla 4.1 se detalla el Product Backlog inicial, con las historias de usuario a implementar priorizadas según las necesidades detectadas por el equipo de desarrollo y el Director del proyecto.

Cuadro 3.1.: Product backlog

Product Backlog			
ID	**Nombre historia**	**Estimación**	**Prioridad**
1	Analizar el know how de la empresa	10	Media
2	Disponer de un listado de clientes offline	5	Media
3	Disponer de un listado de pedidos por cliente offline	5	Media
4	Disponer de un listado de productos offline	5	Media
5	Poder crear pedidos por cliente	8	Alta
6	Poder editar los pedidos no enviados.	8	Alta
7	Tener listados de productos por pedido	5	Alta
8	Poder enviar los pedidos realizados a la central	5	Alta
9	Completa abstracción de la app vs aplicación de gestión	2	Baja
10	Probar la funcionalidad	2	Alta

Más adelante en la Tabla 4.2 se muestra la planificación de las iteraciones con sus respectivas historias y objetivos a cumplir en cada ciclo.

Cuadro 3.2.: Iteraciones

Iteraciones					
Ite.	**Inicio**	**Fin**	**Historias**	**Puntos**	**Objetivos**
1	01/01/2018	12/01/2018	1	10	Analizar como recuperar los datos a procesar. Analizar como se insertan los pedidos en la aplicación de gestión existente. Ver y plantear la Querys necesarias para operar la DB. Iniciar la generación del servicio web que va a comunicar con la BD.
2	15/01/2018	26/01/2018	2	10	Desarrollar conjunto para realizar sincronización de datos. Aplicar métodos de consulta al servicio web.
3	29/01/2018	09/02/2018	1.8	10	Desarrollar conjunto para realizar pedidos. Aplicar métodos de consulta al servicio web. Aplicar métodos de inserción al servicio web.
4	12/02/2018	23/02/2018	1.7	10	Desarrollar conjunto para editar pedidos no enviados. Desarrollar conjunto para ver pedidos no previos. Aplicar métodos de consulta al servicio web.
5	26/01/2018	09/03/2018	2.5	10	Terminar la generación del servicio web.
6	12/03/2018	23/03/2018	1	10	Pruebas de funcionamiento.

3.4. Estimación de costes del proyecto

Teniendo en cuenta los datos anteriores podemos estimar los siguiente recursos de trabajo:

- **Trabajo total estimado:** 480 horas.
- **Coste pactado:** 6.999 €.
- **Coste hora:** 14,58 € /h.

3.4.1. Cálculo de costes del proyecto

3.4.1.1. Recursos de trabajo

Analista/Programador informático: $28{,}000€/14p/20d/8h = 12{,}5€/h$

3.4.1.2. Recursos materiales

Los recursos materiales se consideran cada uno de ellos con un coste por uso de 1,50€en concepto de luz y otros gastos que se cobran independientemente del tiempo de uso: $60d \times 1{,}50€ = 90€$

Cuadro 3.3.: Software

Software		
Concepto	**Coste**	**Licencias**
MS Windows 10	149,00€	1
Text Studio 2.12	0,00€	1
Gimp 2.8	0,00€	1
Eclipse oxygen	0,00€	1
Android Studio 3.1	0,00€	1

3.4.1.3. Costes totales

En la siguiente lista se detallan los costes del proyecto. Aunque a priori parezcan que son deficitarios con una desviación de 853€en perdidas, hay que tener en cuenta que la figura del analista programador y el gestor del proyecto es la misma y las perdidas son absorbidas por su remuneración.

Concepto	Coste
Trabajo	6.000,00€
Materiales	240,00€
Total	6.240,00€
Gastos(4)	249,60€
Subtotal	6.489,60€
IVA(18)	1.362,82€
Total	7852,42€

3.5. Seguimiento del Proyecto

3.5.1. 1º Iteración

Durante la primera iteración, se proyecta realizar el análisis correspondiente de la aplicación cliente y sus posibles implicaciones posteriores. La aplicación cliente se encuentra en un servidor centralizado con una base de datos relacional PostgreSQL. En principio se tiene bastante claro que datos se tienen que recoger y con que formato se insertan los pedidos en la misma. Se desarrollan consultas *(query)* SQL previas para su uso posterior.

3.5.1.1. Comentarios y conclusiones

No hay desviaciones aparentes y se encuentra en plazo.

3.5.2. 2º Iteración

En la segunda iteración se trabaja en la gestión de los listados y el esqueleto de las dos aplicaciones, se ha definido la comunicación y la manera de operar de las mismas.

3.5.2.1. Comentarios y conclusiones

Aparece una ligera desviación que se presupone será incremental en las subsiguientes iteraciones. Hay una pequeña linea difusa entre los diferentes listados que origina la creación de código que a priori esta planificado para iteraciones posteriores.

3.5.3. 3º Iteración

En la tercera iteración estaba planificado el desarrollo primario de la gestión de pedidos. Durante la creación de las pantallas básicas no se había tenido en cuenta el tiempo de creación de los iconos e imágenes del sistema.

3.5.3.1. Comentarios y conclusiones

El diseño de la apariencia de la aplicación Android no se había tenido en cuenta, así pues se dedica una parte importante de la iteración a realizar todos los diseños que vamos a necesitar, tanto para esta iteración como para las posteriores. Se espera absorber este tiempo en iteraciones posteriores.

3.5.4. 4º Iteración

Se procede al desarrollo de código fuente para gestionar los pedidos al completo

3.5.4.1. Comentarios y conclusiones

La elección de tecnologías de desarrollo ágiles como el proyecto Lombok, o AndroidAnnotations, que simplifican la generación de código, permiten absorber las desviaciones temporales acarreadas por las iteraciones posteriores.

3.5.5. 5º Iteración

Se finalizan algunas funcionalidades pendientes y se termina el servicio web para gestionar los datos a enviar/recibir.

3.5.5.1. Comentarios y conclusiones

Como estamos por debajo de la estimación se aprovecha para realizar una instalación previa y algunas pruebas.

3.5.6. 6º Iteración

Durante esta Iteración, se realizan pruebas unitarias, de integración y funcionales con el cliente, identificando ligeros fallos que se corrigen durante la duración de esta iteración.

3.5.7. Final de iteraciones

Una vez extrapolados los datos obtenidos por el tablón de seguimiento, obtenemos unas gráficas que muestran las desviaciones generadas por los diferentes escollos que se han generado durante el desarrollo de la aplicación

En la siguiente gráfica podemos ver la desviación por iteración:

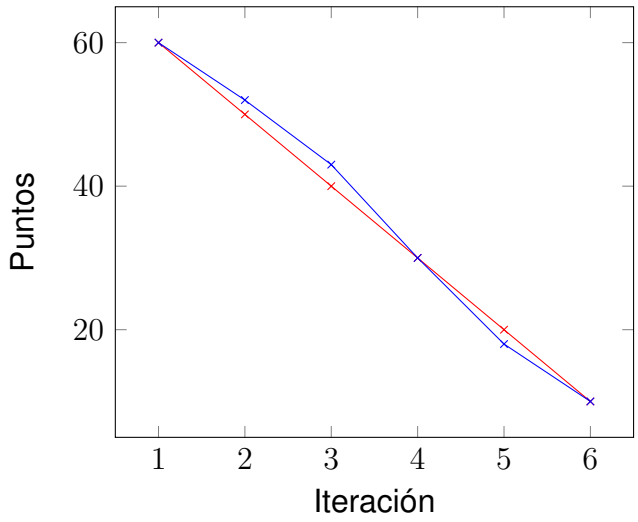

Figura 3.2.: Product Burn-Down

En la siguiente gráfica podemos ver la desviación por Iteraciones/día:

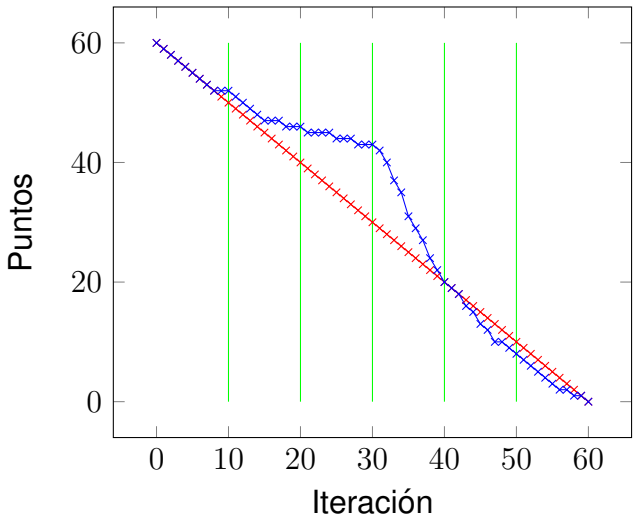

Figura 3.3.: Sprints BurnDown

Parte III.

Tecnologías

4 Tecnologías utilizadas

A continuación se presentarán las tecnologías empleadas que permitieron el desarrollo de la herramienta en cuestión. Primero se listan y explican los lenguajes utilizados para la programación y diseño/desarrollo de interfaces, luego las herramientas que dieron soporte al proceso de desarrollo y por último los entornos utilizados.

4.1. Lenguajes utilizados

Se pueden clasificar según su utilización:

- ***Java:*** La herramienta está desarrollada en mayor parte en este lenguaje, ya previamente conocido por el equipo de desarrollo y además tiene las ventajas de disponer de mucha documentación en la web. Para el desarrollo en este lenguaje se utiliza el jdk 1.8. [wikipedia(2017e)]

- ***Groovy:*** Es un lenguaje de programación orientado a objetos implementado sobre la plataforma Java. Tiene características similares a Python, Ruby, Perl y Smalltalk. Se obtiene mayor nivel de abstracción con este lenguaje. [wikipedia(2017d)]

4.2. Herramientas utilizadas

A continuación se listan las herramientas utilizadas dentro del proceso de desarrollo de la herramienta:

- ***Git+GitHub:*** Para el control de versiones se utilizó Git, un sistema distribuido de control de versiones. Al ser distribuido, cada desarrollador cuenta con un "clon" completo del repositorio. Cada uno puede realizar cambios sobre el proyecto, lo que permite ser autónomo y trabajar en cualquier situación. Como repositorio central, se decidió utilizar los servicios de GitHub, una plataforma de desarrollo colaborativo de software para el alojamiento de repositorios de proyectos utilizando el sistema de control de versiones Git. El código se almacena de forma pública bajo licencias de código Open Source.
- ***Kunagi:*** Entre las necesidades para encarar una metodología ágil inspirada en SCRUM, era necesario una herramienta de soporte para la gestión del desarrollo ágil sin implicar mayores complejidades técnicas. Como se explicó anteriormente, Kunagi ofrece la administración integrada de proyectos complementando la metodología SCRUM a través de otras mejores prácticas para cubrir todas las necesidades para la administración de proyectos. Esta herramienta corre localmente en la máquina de un integrante del equipo, sobre un servidor Apache Tomcat.
- ***Gimp:*** Es un programa de edición de imágenes digitales, tanto dibujos como fotografías. Es un programa libre y gratuito. Forma parte del proyecto GNU y está disponible bajo la Licencia pública general de GNU y GNU Lesser General Public License.

4.3. Entornos de desarrollo

Los entornos de programación utilizados fueron:

- ***Eclipse:*** Es una plataforma de software compuesto por un conjunto de herramientas de programación de código abierto multi-plataforma para desarrollar lo que el proyecto llama .ªplicaciones de Cliente Enriquecido", opuesto a las aplicaciones Çliente-liviano"basadas en navegadores. Esta plataforma, típicamente ha sido usada para desarrollar entornos de desarrollo integrados (del inglés IDE), como el IDE de Java llamado Java Development Toolkit (JDT) y el compilador (ECJ) que se entrega como parte de Eclipse (y que son usados también para desarrollar el mismo Eclipse). [wikipedia(2017b)]
- ***Android Studio:*** Es el entorno de desarrollo integrado oficial para la plataforma Android. Fue anunciado el 16 de mayo de 2013 en la conferencia Google I/O, y reemplazó a Eclipse como el IDE oficial para el desarrollo de aplicaciones para Android. La primera versión estable fue publicada en diciembre de 2014. [wikipedia(2017a)]

4.4. Últimas consideraciones

A parte de los entornos de programación tomamos en consideración dos herramientas que mejoran sustancialmente la estructura de las aplicaciones.

- ***Lombok project:*** paquete de librerías en java que permiten ignorar ciertas estructuras tediosas (getters, setters, constructores...) que son integradas en tiempo de compilación. [lom(2017)]
- ***AndroidAnnotations:*** paquete de librerías en java que facilita el mantenimiento de aplicaciones en Android, dando una estructura más sencilla de gestionar y fácil de mantener. [Martin(2017)]
  - ***Inyección de dependencias:*** inyecta vistas, extras, servicios, recursos, ...
  - ***Modelo de subprocesamiento simplificado:*** anotar métodos para ejecutarlos en subprocesos.
  - ***Enlace de eventos:*** anotar métodos para controlar eventos en vistas, eliminamos las subclases del tipo listener.
  - ***Cliente REST:*** Crea una interfaz de cliente, AndroidAnnotations genera la implementación.
  - ***Sin Magia:*** Como se generan subclases en tiempo de compilación, se puede comprobar el código para ver cómo funciona.

Parte IV.

Fundamentos

5 Fundamentos en la creación de aplicaciones

5.1. Fundamentos de la creación de web-services JAVA

Para la realización de esta sección se ha tomado como referencia técnica el libro: Fundamentos de programación Java. [Villalobos(2016)]

5.1.1. Definición

Un servicio Web*(web-service)* es una colección de protocolos y estándares que sirven para intercambiar datos entre aplicaciones. La interoperatividad se consigue mediante la adopción de estándares abiertos.

Estos servicios proporcionan mecanismos de comunicación estándares entre diferentes aplicaciones, que interactúan entre sí para presentar información dinámica al usuario. Para proporcionar interoperatividad y expansibilidad entre estas aplicaciones, y que al mismo tiempo sea posible su combinación para realizar operaciones complejas, es necesaria una arquitectura de referencia estándar.

5.1.2. Aspectos comunes de los web-services

Los aspectos técnicos comunes son:

- Los Servicios Web exponen funcionalidad útil a los usuarios Web mediante un protocolo Web estándar. En la mayoría de casos, el protocolo utilizado es **Simple Object Access Protocol (SOAP)**.
- Los Servicios Web proporcionan un modo de describir sus interfaces con suficiente detalle para permitir a un usuario construir una aplicación cliente para comunicarse con ellos. Esta descripción se proporciona generalmente en un documento XML que responde al nombre de documento **Servicios web Description Language (WSDL)**.
- Los Servicios Web se registran de modo que los potenciales usuarios puedan encontrarlos. Esto se realiza mediante **Universal Discovery Description and Integration (UDDI)**.

Aunque la idea de la programación modular no es nueva, el éxito de esta tecnología reside en que se basa en estándares conocidos en los que ya se tiene una gran confianza, como el XML.

5.1.3. Características

Las principales características son:

- Aportan interoperatividad entre aplicaciones de software independientemente de sus propiedades o de las plataformas sobre las que se instalen, permitiendo la interoperatividad entre plataformas de distintos fabricantes mediante protocolos estándar.
- Los servicios web fomentan los estándares y protocolos basados en texto, que hacen más fácil acceder a su contenido y entender su funcionamiento.
- Al apoyarse en HTTP, los servicios web pueden aprovechar los sistemas cortafuegos sin necesidad de cambiar sus reglas de filtrado. La principal razón para usar servicios Web es que se basan en HTTP sobre TCP en el puerto 80. Dado que las organizaciones protegen sus redes mediante cortafuegos (firewalls) que filtran y bloquean gran parte del tráfico de Internet, cierran casi todos los puertos TCP salvo el 80, que es, precisamente, el que usan los navegadores. Los servicios Web se canalizan por este puerto, por la simple razón de que no resultan bloqueados.
- Permiten que servicios y software de diferentes compañías ubicadas en diferentes lugares geográficos puedan ser combinados fácilmente para proveer servicios integrados.
- Los servicios web son muy prácticos al aportar gran independencia entre la apli-

cación que usa el servicio web y el propio servicio. De esta forma, los cambios a lo largo del tiempo en uno no deben afectar al otro. Esta flexibilidad será cada vez más importante, dado que la tendencia a construir grandes aplicaciones a partir de componentes distribuidos más pequeños es cada día más acusada.

5.1.4. SOAP

SOAP es el acrónimo de “Simple Object Access Protocol” y es el protocolo que se oculta tras la tecnología que comúnmente denominamos “Web Services” o servicios web. SOAP es un protocolo extraordinariamente complejo pensado para dar soluciones a casi cualquier necesidad en lo que a comunicaciones se refiere, incluyendo aspectos avanzados de seguridad, transaccionalidad, mensajería asegurada y demás.

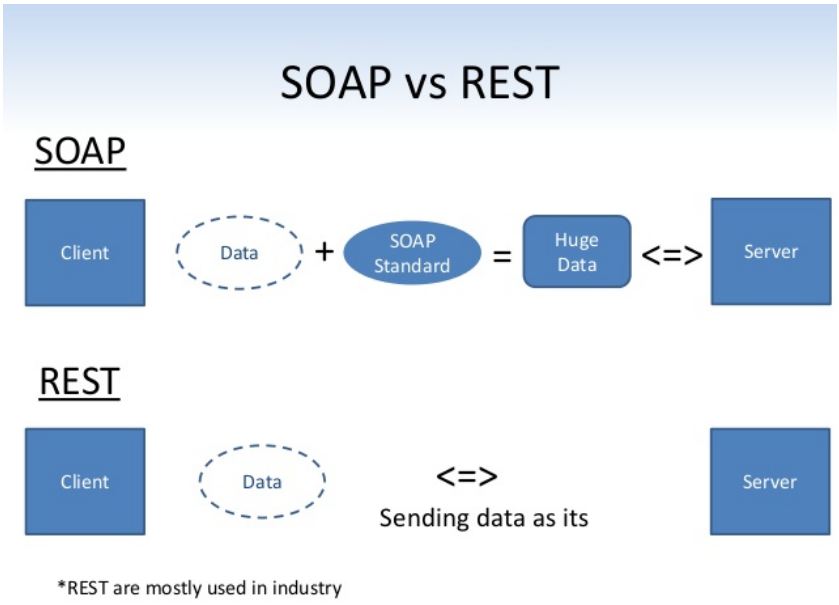

Figura 5.1.: Esquema principal del proyecto Android

5.1.5. REST

REST deriva de *Representational State Transfer”*, que traducido vendría a ser “transferencia de representación de estado”, más o menos servicio REST no tiene estado *(stateless)*. El estado lo mantiene el cliente y por lo tanto es el cliente quien debe pasar el estado en cada llamada. Si quiero que un servicio REST me recuerde, debo añadirle quien soy en cada llamada. Y lo mismo aplica para el resto de información.

El no tener estado es una desventaja clara: tener que pasar el estado en cada llamada es, como mínimo, tedioso, pero la contrapartida es clara: esca labilidad. Para mantener un estado se requiere algún sitio (generalmente memoria) donde guardar todos los estados de todos los clientes. A más clientes, más memoria, hasta que al final podemos llegar a no poder admitir más clientes, no por falta de CPU, sino de memoria.

5.1.6. API REST

En una API REST la idea de "servicio" como tal desaparece. Lo que tenemos son recursos, accesibles por identificadores (URIs). Sobre esos recursos podemos realizar acciones, generalmente diferenciadas a través de verbos HTTP distintos.

Así, en un servicio web clásico **(SOAP)** tendríamos un servicio llamado ***CoffeeService*** que tendría un método ***GetAll()*** que me devolvería todas los tipos de café. La idea, independientemente de la tecnología usada para consumir el servicio web, es que se llama a un método ***GetAll()*** de un servicio remoto ***CoffeeService***. Del mismo modo para obtener un tipo de café en concreto llamaríamos al método ***GetById()*** pasándole el id del tipo de café. De ahí que se diga que están orientados a RPC *(Remote Procedure Call – Llamada a método remoto)*.

Por su parte en un servicio **(REST)** la propia idea de servicio se desvanece. En su lugar nos queda la idea de un "recurso", llamémosle "Colección de cafés" que tiene una URI que lo identifica, p. ej. /Coffee. Así, si invoco dicha URI debo obtener una representación de dicho recurso, es decir, debo obtener el listado de todos los tipos de café.

Para obtener datos de un tipo de café, habrá otro recurso (café) con una URI asociada. Para acceder a uno de sus elementos con una URI tipo /Coffee/123, siendo *123* el ID de la café. A la hora de hacer operaciones típicas la URI para acceder es siempre /Coffee/123 y es el verbo HTTP (GET, POST, PUT, DELETE,...) el que indica cual es la operación que deseamos hacer.

Figura 5.2.: El conjunto de los Servicios web

5.1.7. La elección

Los servicios web SOAP terminan siendo un monstruo con muchas capacidades pero que en la mayoría de los casos no necesitamos. Por su parte REST es simple. REST no quiere dar soluciones para todo y por lo tanto no pagamos con una demasiada complejidad una potencia que quizá no vamos a necesitar. Así que la elección es clara usaremos REST

5.2. Fundamentos de la creación de aplicaciones Android

Para la realización de esta sección se ha tomado como referencia técnica el libro: Fundamentos del desarrollo de aplicaciones para Android. [robles(2017)]

5.2.1. Cómo funciona el sistema operativo Android

Una vez instalada en un dispositivo, cualquier aplicación corre en su propio entorno limitado de seguridad [robles(2017)]:

- El sistema operativo Android es un sistema multi-usuario de Linux en el que cada aplicación es un usuario diferente.
- Por defecto, el sistema asigna a cada aplicación de una única ID de usuario *(el ID es utilizado únicamente por el sistema y desconocido para la aplicación)*. Establece permisos para todos los archivos en una aplicación para que sólo el ID asignado pueda acceder a ellos.
- Cada proceso tiene su propia máquina virtual, por lo que el código de una aplicación se ejecuta de forma aislada de otras aplicaciones.
- Por defecto, cada aplicación se ejecuta en su propio proceso de Linux, lo inicia cuando alguno de los componentes se ejecuta, a continuación, cierra el proceso cuando ya no se necesita o cuando el sistema debe recuperar la memoria para otras aplicaciones.

Con esto, el SO implementa **el principio de privilegios mínimos**, que consiste en que cada aplicación sólo puede acceder a los componentes requeridos para hacer su trabajo. Esto genera un entorno muy seguro en el que la aplicación en cuestión no puede acceder a partes del sistema para las cuales no se le ha otorgado permiso. De cualquier modo, existen manera para que una aplicación pueda compartir datos con otras aplicaciones, o también que, una aplicación pueda acceder a los servicios del sistema:

- En el caso en el que dos aplicaciones compartan el mismo ID, son capaces de acceder cada una a los archivos de la otra. Para ahorrar recursos del sistema, en este caso en el que dos aplicaciones tienen el mismo ID también se pueden organizar para ejecutar en el mismo proceso de Linux y compartir la misma máquina virtual.
- Una aplicación puede solicitar permiso para acceder a los datos del dispositivo, tales como los contactos del usuario, el almacenamiento externo (tarjeta SD), cámara, etc. Todos los permisos de la aplicación debe ser autorizados por el usuario durante la instalación.

5.2.2. Componentes de las Aplicaciones Android

Los principales componentes de toda aplicación Android son:

- ***Actividades(Activities):*** Cada pantalla de una aplicación. Utilizan Vistas (Views) como componentes que muestran información y responden a las acciones del usuario
- ***Servicios(Services):*** Componentes de la aplicación que se ejecutan deforma invisible, actualizando los datos y las Actividades, y disparando Notificaciones. Realizan el procesamiento normal de la aplicación que debe continuar incluso cuando las Actividades de la aplicación no están visibles.
- ***Proveedores de Contenidos(Content Providers):*** Almacenes de datos compartidos. Gestionan las Bases de Datos para las aplicaciones.
- ***Intenciones(Intents):*** Mecanismo que permite el paso de mensajes destinados a ciertas Actividades o Servicios, o a todo el sistema *(Broadcast Intents)*. Exponen la intención de que se haga algo.
- ***Receptores de Broadcast(Broadcast Receivers):*** Los crean las aplicaciones como consumidores de las Intenciones de broadcast que cumplan ciertos criterios.
- ***Notificaciones(Notifications):*** Mecanismo que permite a las aplicaciones señalar algo a los usuarios sin interrumpir la Actividad en primer plano.

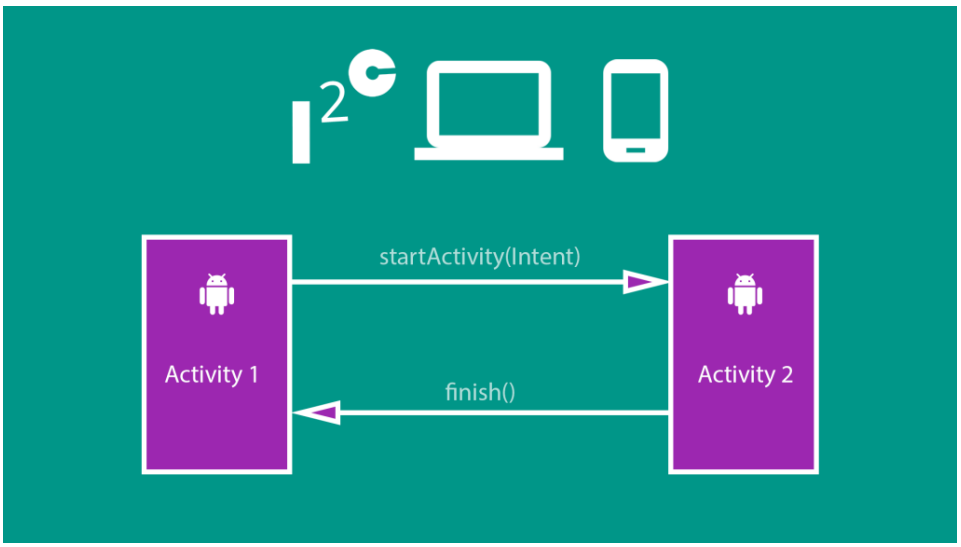

Figura 5.3.: Interacción de actividades

Ademas también existen nuevas implementaciones *(fragmentos, Asíncronos, etc.)* que no vamos a explicar.

5.2.3. Manifiesto de una Aplicación Android *(Android manifest)*

Toda aplicación desarrollada en Android incluye un fichero de Manifiesto, el ***AndroidManifest.xml***. Este define la estructura de la aplicación y sus componentes. Incluye un nodo raíz y un nodo para cada uno de sus tipos de componentes. A través de filtros determina como interactuará la aplicacións. Algunos de los nodos más importantes son:

- ***Nodo raíz(manifest):*** Incluye el nombre del paquete de la aplicación.
- ***Nodo aplicación(application):*** Indica los metadatos *(título, icono, tema, etc.)* y contiene los nodos de actividades, servicios, proveedores de contenido y receptores de broadcast.
- ***Nodo permisos a usar(uses-permission):*** Declara los permisos necesarios para operar. Estos serán presentados al usuario durante la instalación para que los acepte o deniegue.
- ***Nodo permisos a proveer(permission):*** Nodo . Define un permiso que se requiere para que otras aplicaciones puedan acceder a partes restringidas de la aplicación. Las otras aplicaciones necesitarán poner un uses-permission en su Manifiesto para utilizar este permiso.
- ***Nodo instrumentación(instrumentation):*** Nodo . Permite definir test de ejecución para las Actividades y Servicios.

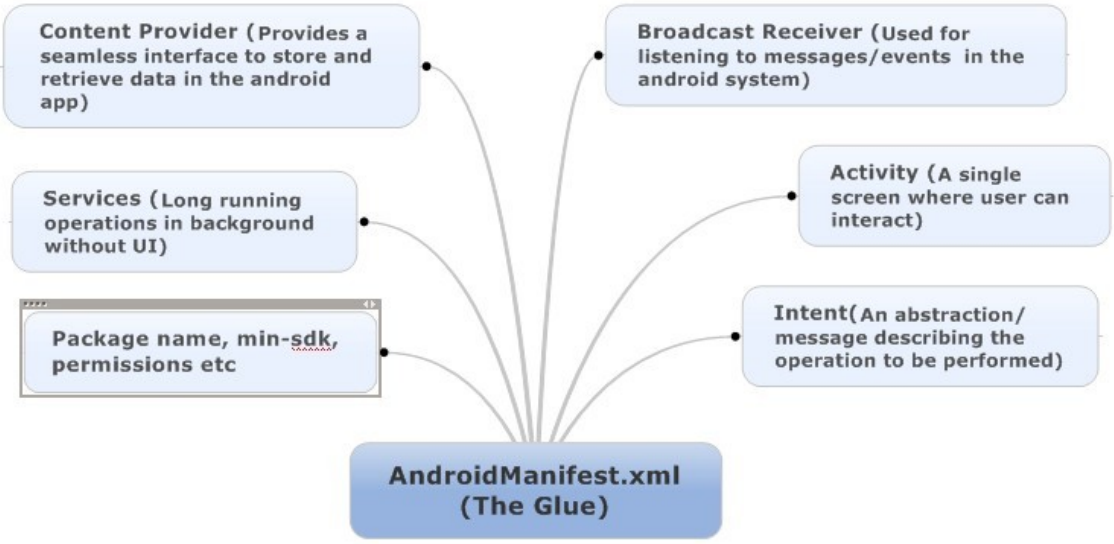

Figura 5.4.: Elementos de un manifiesto Android

5.2.4. Creación y destrucción de Aplicaciones y Actividades *(Ciclo de vida)*

Las aplicaciones Android son diferentes a sus homologas en los sistemas operativos tradicionales, en Android sólo hay una aplicación en primer plano que normalmente estará ocupando toda la pantalla. Las aplicaciones estarán formadas por Actividades.

Al arrancar una nueva aplicación, pasa a primer plano situando una Actividad encima de la que hubiera, formándose así una pila de actividades. En el momento en el que el usuario presiona el botón “back”, se cierra la actividad en primer plano y recupera la de la cima de la pila. Las aplicaciones Android no tienen control ninguno sobre su propio ciclo de vida, esto implica que deben estar pendientes de posibles cambios en su estado y reaccionar como corresponda. En particular deben estar preparadas para su terminación o destrucción en cualquier momento.

Una actividad se puede encontrar en los siguientes estados:

- ***Activa(Running):*** La actividad está encima de la pila, es visible, tiene el foco. Cuando otra actividad pase a estar activa, esta pasará a estar pausada.
- ***Pausada(Paused):*** La actividad es visible pero no tiene el foco. Se alcanza este estado cuando pasa a activa otra actividad transparente o que no ocupa toda la pantalla. Cuando una actividad es tapada por completo pasa a estar parada.
- ***Parada(Stopped):*** Cuando la actividad no es visible. Permanece en memoria reteniendo su estado. Cuando una actividad entra en parada puede ser bueno que salve todos sus datos y el estado de la interfaz de usuario.
- ***Destruida(Destroyed):*** Cuando la actividad termina, o es matada por el runtime de Android. Sale de la pila de actividades. Necesita ser reiniciada para volver a estar activa.

Y del mismo modo existirán una serie de métodos de transición entre unos estados y otros:

- ***onCreate():*** Se invoca cuando la Actividad se arranca por primera vez. Se utiliza para tareas de inicialización, como crear la interfaz de la Actividad.
- ***onStart():*** Se invoca cuando la Actividad va a ser mostrada al usuario.
- ***onResume():*** Se invoca cuando la Actividad va a empezar a interactuar con el usuario.
- ***onPause():*** Se invoca cuando la actividad va a pasar al fondo porque otra actividad ha sido lanzada para ponerse delante. Se utiliza para guardar el estado persistente de la Actividad
- ***onStop():*** Se invoca cuando la actividad va a dejar de ser visible y no se necesitará durante un tiempo. Si hay escasez de recursos en el sistema, este método

podría no llegar a ser invocado y la Actividad ser destruida directamente.

- ***onRestart():*** Se invoca cuando la Actividad va a salir del estado de parada para volver a estar activa.
- ***onDestroy():*** Se invoca cuando la Actividad va a ser destruida. Si hay escasez de recursos en el sistema, este método podría no llegar a ser invocado y la Actividad ser destruida directamente.
- ***onSaveInstanceState():*** Se invoca para permitir a la actividad guardar su estado, por ejemplo la posición del cursor en una caja de texto.
- ***onRestoreInstanceState():*** Se invoca para recuperar el estado guardado por onSaveInstanceState().

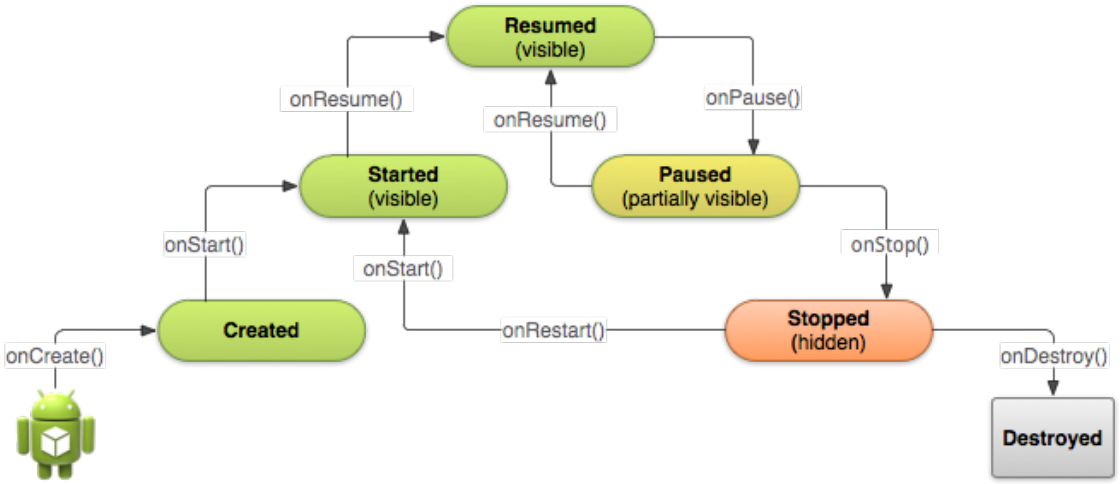

Figura 5.5.: Ciclo de una actividad

5.2.5. Recursos de las aplicaciones Android

Una aplicación para Android se compone de algo más que código, requiere de recursos que están separados, como imágenes, archivos de audio, y todo lo relativo a la presentación visual de la aplicación.

Para todos estos recursos Android proporciona un identificador único entero dentro de la aplicación, que puede utilizarse para hacer referencia al recurso en el código de aplicación o de otros recursos definidos en XML. Hay distintos tipos de recursos, que se definen en ficheros XML alojados en una cierta subcarpeta de res:

- ***Valores simples (carpeta values):*** Strings, colores y dimensiones. Cada fichero XML contiene la definición de uno o más de estos elementos. Todos estos recursos se identifican con el valor de su atributo name.
- ***Recursos dibujables(carpeta drawable):*** Ficheros con imágenes, incluyendo

el icono de la aplicación. Estos recursos se identifican con su nombre de fichero, y los recuadros de color con el valor de su atributo name.

- ***Animaciones(carpeta anim):*** Usados para animaciones sencillas sobre uno o varios gráficos: rotaciones. Fading, movimiento, etc. Cada animación se define en un fichero xML. Se identifican con su nombre de fichero.
- ***menu(carpeta menu):*** Existen tres tipos de menús: de opciones, contextual y submenú. El menú de opciones y el menú contextual se identifican con su nombre de fichero y el submenú con el valor de su atributo id.
- ***Diseños(carpeta layout):*** Cada layout se define en un fichero XML. Dentro del layout se definen los elementos que lo componen, como puedan ser los Views o los ViewGroups. Se identificará por su nombre de fichero y los elementos del layout se podrán identificar con su atributo id.
- ***Estilos(carpeta values):*** Un estilo es uno o más atributos que se aplican a un elemento. El tema se define como uno o más atributos que se aplican a todo lo que hay en pantalla, este se asigna como atributo a una actividad en el Manifiesto. El estilo se referencia con el valor de su atributo name.

También existen otros recursos como drawables, internacionalización, etc.

Parte V.

Estructura

6 Diseño Y Estructura

En este capítulo se describe el diseño final del funcionamiento de la aplicación y se explican los elementos que han influido en las decisiones.

6.1. Descripción general

La solución propuesta, se compone de dos componentes diferenciados , por un lado una aplicación Android y por otro lado un servicio web [J.I.Cuerno(2018d)]

El servicio se comporta como una interface entre la aplicación Android y la plataforma ERP[8.1.4] que dispone el cliente. A través de la base de datos del ERP el servicio recupera los datos necesarios para proveer a la aplicación Android y también inserta los prepedidos realizados por la misma. [J.I.Cuerno(2018c)]

Una de las principales características de la aplicación Android, es que tiene que disponer de los datos necesarios para operar, aunque no dispongamos de conexión en este momento. [J.I.Cuerno(2018a)] Para ello dispone de una opción de sincronización de sus datos almacenados con los datos actualizados por la empresa. Para optimizar su rendimiento intentado realizar el menor numero de conexiones, se puede configurar el nº de registros por conexión queremos recuperar.

A efectos de estructurar los datos durante las comunicaciones, la opción elegida es el estándar JSON, que sera explicado mas adelante.

En la siguiente figura podemos ver un esquema, mostramos de una manera esquemática como es la comunicación entre las partes.

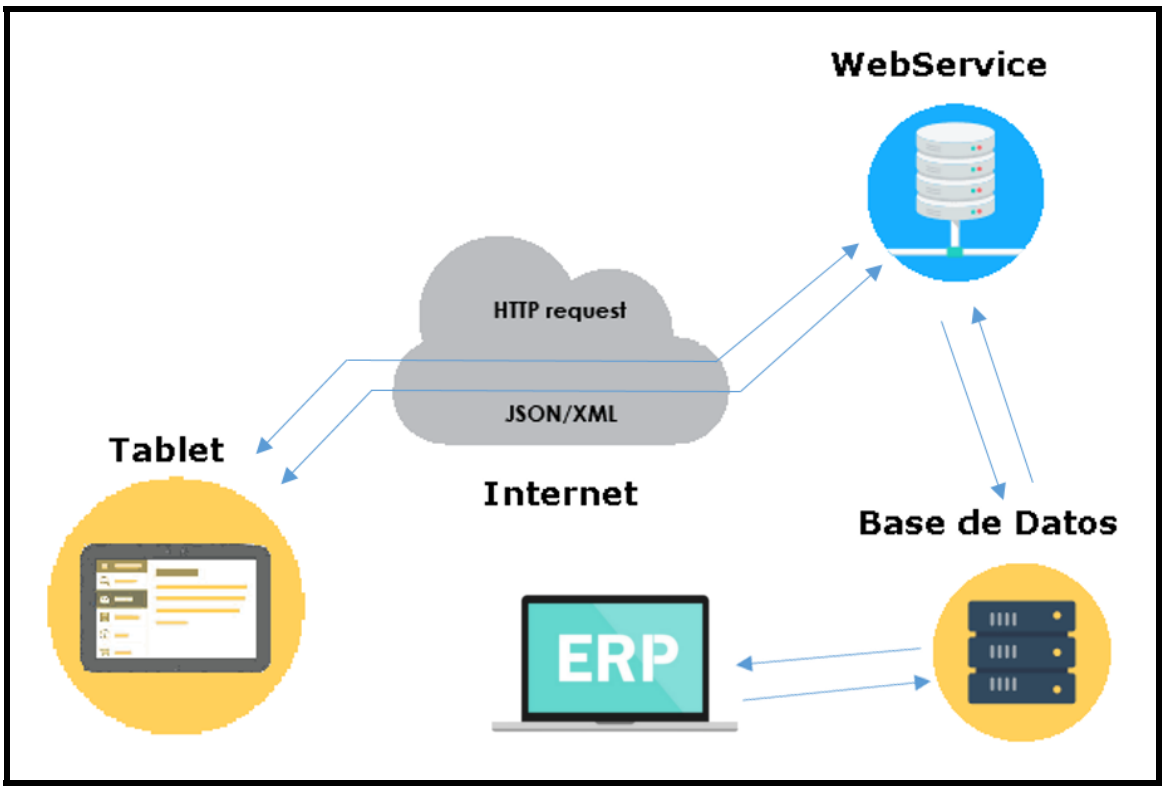

Figura 6.1.: Esquema principal del proyecto

6.2. Servicio web

6.2.1. Descripción general

Antes de abordar la descripción general vamos a definir el termino faceta para su comprensión posterior.

Una faceta, es un aspecto configurable para ejecutar una determinada tarea, cumplir determinados requisitos o tener determinadas características. Por ejemplo, la faceta **EAR** configura un proyecto para que funcione como una aplicación de empresa añadiendo un descriptor de despliegue y configurando la **classpath** del proyecto.

Dicho esto, dentro del entorno elegido *(Eclipse Oxygen)* hemos elegido un tipo de proyecto **Dynamic Web Project** que implementa de serie una colección de facetas que serán de ayuda a la hora de desarrollar el servicio web.

- ***Dynamic Web Module 3.0:*** Añade soporte para la generación de contenido web dinámico en el API de Servilleta Java.
- ***Java 1.8:*** Añade soporte para la escritura de aplicaciones en Java.
- ***JavaScript 1.0:*** Permite el desarrollo de Suscriptor utilizando múltiples archivos de origen en una ruta de inclusión configurable.
- ***JAX-RS (REST Web Services) 2.0:*** Permite que el proyecto se implemente con las capacidades de **JAX-RS**.

6.2.2. Metodología empleada

Como hemos comentado en apartados anteriores la elección del tipo de especificación ha sido **REST**. Al usar esta especificación debemos cumplir con una estructura definida, extbfJAX-RS.

6.2.2.1. JAX-RS

Es una **API** del lenguaje de programación Java que proporciona soporte en la creación de servicios web de acuerdo con el estilo arquitectónico **REST** ***(Representational State Transfer)***. **JAX-RS** usa anotaciones, introducidas en Java SE 5, para simplificar el desarrollo y despliegue de los clientes y puntos finales de los servicios web. [wikipedia(2017f)]

6.2.3. Diseño e implementación

Una vez configurado el tipo de proyecto en Eclipse, elegimos **MAVEN** *[8.1.1]* como su constructor de proyectos.

Durante el desarrollo usaremos anotaciones *[8.1.3]*, que nos proporciona por defecto **JAX-RS**, esto nos permite relacionar las URIS del servicio web con el código fuente de una manera mas intuitiva.

En el ejemplo siguiente mostramos el código fuente correspondiente a la URI : *http://[servidor:puerto]/[servicio]/sincroniza/totales* .

```
@Path(value = "/sincroniza")
public class InTrazaWS {

  @GET
  @Path("totales")
  @Produces(MediaType.APPLICATION_JSON)
  public Totales consultaTotalesBD() {
```

```
        return JDBCQuery.getRegistrosTotales(); }
      .......
   }
```

Vamos a disponer de varias etiquetas de anotación que nos van a proveer de los métodos necesarios para la comunicación con el servicio.

```
@GET - doGet del Servlet
@POST - doPost del Servlet
@Path("prepedido") - path final de la uri
@Consumes(MediaType.APPLICATION\_JSON) consulta por JSON
@Produces(MediaType.APPLICATION\_JSON) escritura por JSON
```

Debemos de definir las rutas para recuperar/insertar los diferentes conjuntos de datos con los que trabajará la aplicación Android. A continuación mostramos un listado de los mismos:

- **totales:** consultaTotalesBD
- **artículos:** consultaArticulosBD
- **clientes:** consultaClientesBD
- **ruteros total fraccionados:** getRuterosTotalFraccionados
- **rutero tarifa cliente:** consultaTarifaClienteRuteroBD
- **rutero peso total anio:** consultaPesoTotalAnioRuteroBD
- **rutero datos:** consultaDatosParaRuteroBD
- **pre-pedido:** enviaPrepedidoBD

Cada elemento del listado corresponde a un objeto o conjunto de objetos que se ha procesado de una cadena de caracteres estructurada bajo la especificación **JSON**.

Tanto para insertar como para obtener los elementos, se realizaran a través de los métodos del listado, que implementan toda la lógica de operación de la base de datos del cliente, a través del conector java correspondiente. En este caso **Postgre SQL**.

6.2.4. Diagramas

En la siguiente figura vemos el diagrama **UML** con las clases principales del servicio web, los accesos a través de intrazaWs y su relación esquemática.

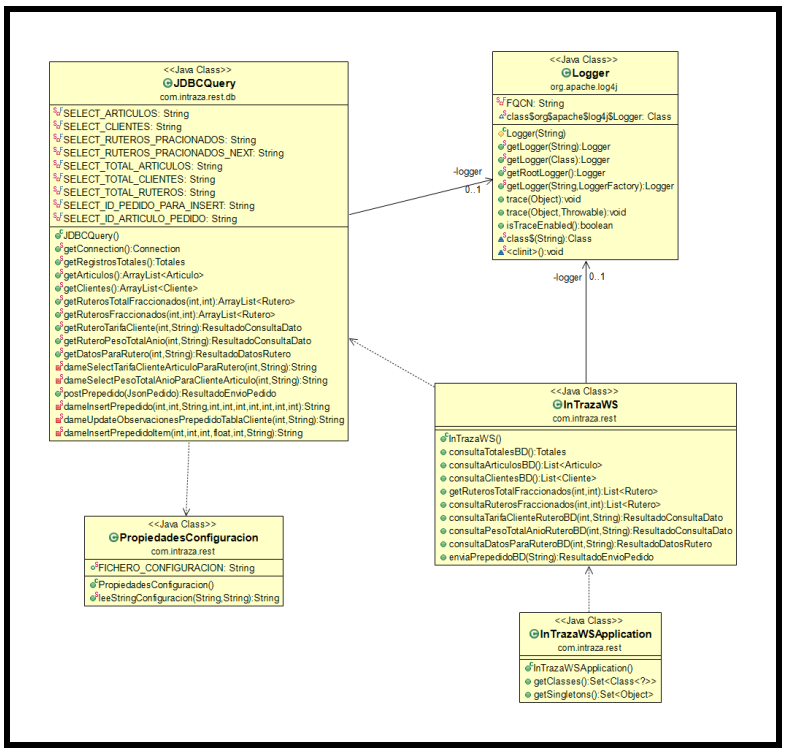

Figura 6.2.: Esquema principal del proyecto

En la siguiente figura vemos el diagrama **UML** con las relaciones de los objetos transformables a partir de la cadena recibida con la especificación **JSON**.

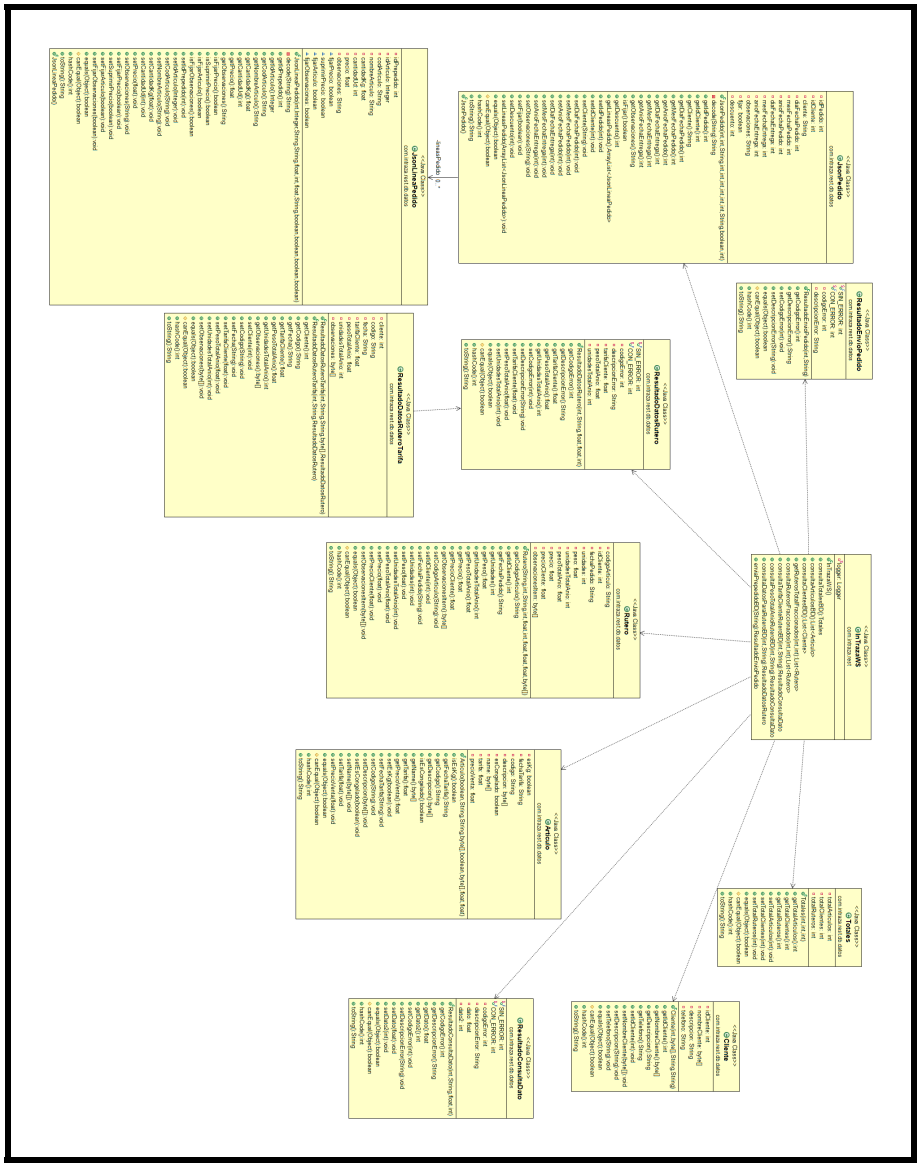

Figura 6.3.: Relaciones de datos

En la siguiente figura vemos el diagrama **UML** con la relación de la clase que gestiona la comunicación con la Base de datos y las clases objeto del servicio.

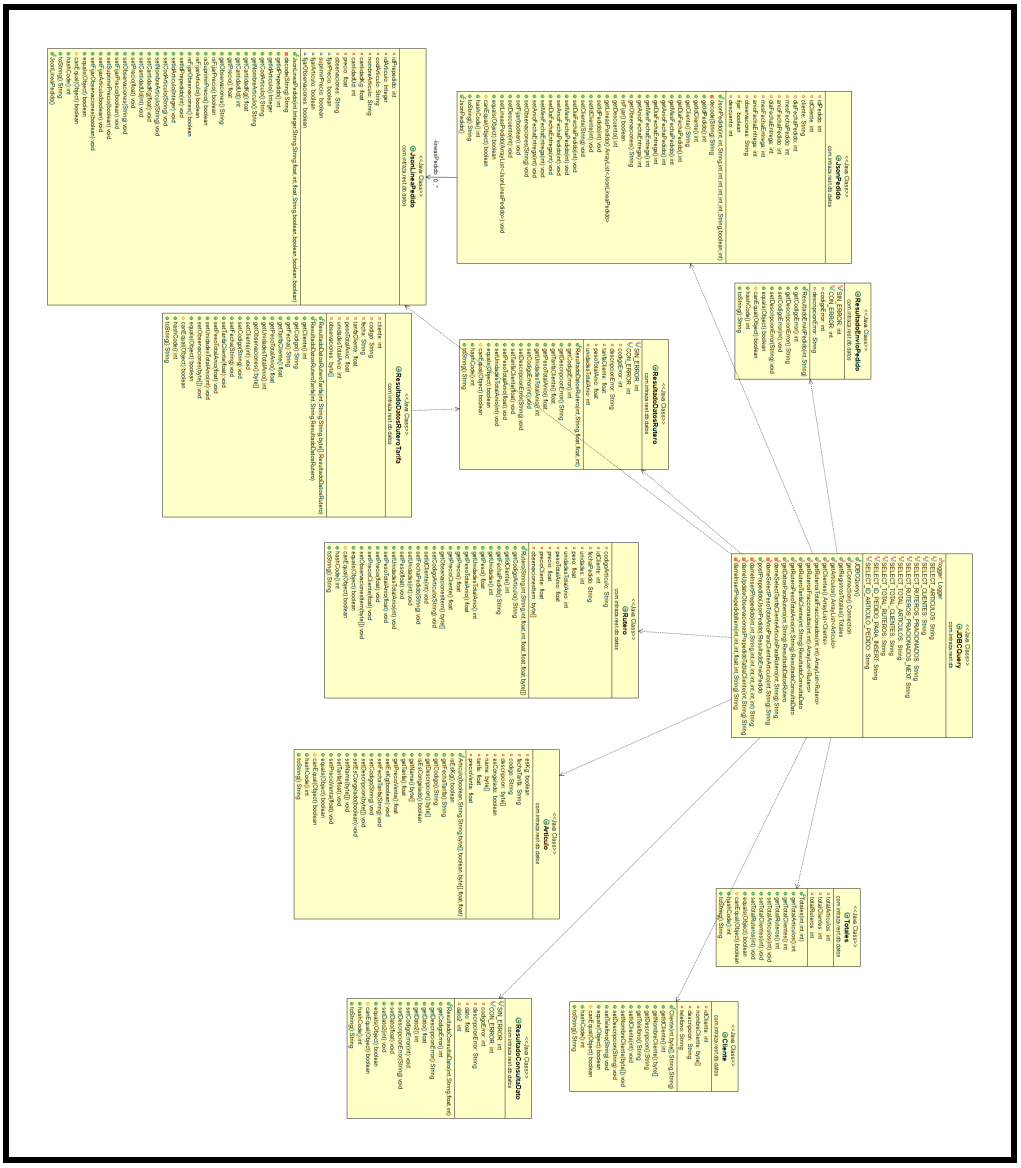

Figura 6.4.: Relaciones JSON

6.3. Aplicación Android

6.3.1. Descripción general

Para el desarrollo de la aplicación Android, se ha optado por un programa nativo, usando la herramienta de desarrollo Android studio. Se ha decidido hacerla retrocompatible hasta versión 5, asegurándonos un 80 % de compatibilidad con los terminales del mercado. Como su estructura tópica ya ha sido comentada en capitulo de fundamentos, no vamos a repetir en esta sección lo ya comentado.

Figura 6.5.: Aplicaciones Android

La aplicación es un interface de usuario para la gestión de pedidos mientras se encuentra desplazado, pudiendo disponer de datos de gestión de los mismos aunque no pueda establecer comunicación con el servidor. Para ello ademas de las pantallas de operación, dispondrá de una pequeña base de datos (**SQLlite**) para almacenar los datos de operación.

6.3.2. Metodología empleada

Así como el servicio disponía de **MAVEN** [8.1.1] como su constructor de proyectos, en Android studio tendremos **GRADLE** [8.1.2].

Ademas del SDK de Android dispondremos de las siguientes librerías para poder gestionar todas las necesidades operativas de la aplicación.

- **Jackson-core:** librería para la gestión de objetos JSON.
- **Jackson-databind:** librería para la gestión de objetos JSON.
- **Lombok:** librería anteriormente comentada.
- **Androidannotations:** librería anteriormente comentada.

6.3.3. Diseño e implementación

Una vez tenemos la estructura de la aplicación definida, dividiremos la estructura en una actividad principal y 4 actividades secundarias para la gestión de las secciones correspondientes y por último un conjunto de clases ***Dialog*** para la gestión de ventanas emergentes durante la ejecución

El almacenamiento de datos internos se realiza a través de una base de datos en SQLlite y se definen las siguientes tablas:

- TablaObservacion.
- TablaRutero.
- TablaPrepedidoItem.
- TablaPrepedido.
- TablaCliente.
- TablaConfiguracion.

Para la gestión de la internacionalización*(I18N)* se definen 4 archivos con los idiomas correspondientes: Castellano, Catalán, Francés, Ingles.

Se realiza el diseño gráfico de botones, backgrounds, etc, por medio del programa Gimp, y se generan drawables para los efectos visuales del pulsado de botones.

En el ejemplo siguiente mostramos parte del código fuente correspondiente a la actividad principal de la aplicación, se puede comprobar que con las anotaciones realizadas manejamos recursos,eventos,etc, transformando la clase a un formato mas amigable.

```
@EActivity(layout.main)
public class InTrazaActivity extends Activity {
   .......

   @AfterViews
   void init() {
      config = Configuracion.getInstance();
      config.preparaPropiedades(this);
   }
   .......

   @OnActivityResult(DIALOGO_PIDE_DATOS_NUEVO_PEDIDO)
   void onResultUno(final int resultCode, final Intent data) {
      if (Activity.RESULT_OK == resultCode) {
         final DatosPedido datosPedido =
            data.getParcelableExtra("DATOS_PEDIDO");
         pantallaRutero(datosPedido);
      }
      habilitaClickEnActivity(true);
   }
   .......
}
```

Aquí vemos si lo hubiésemos escrito en el formato original java.

```
public final class InTrazaActivity extends InTrazaActivity
    implements HasViews, OnViewChangedListener {

   private final OnViewChangedNotifier onViewChangedNotifier_ = new
      OnViewChangedNotifier();

   final static String DATOS_PEDIDO_EXTRA = "DATOS_PEDIDO";

   @Override
   public void onCreate(Bundle savedInstanceState) {
      OnViewChangedNotifier previousNotifier =
         OnViewChangedNotifier.replaceNotifier(onViewChangedNotifier);
      init(savedInstanceState);
      super.onCreate(savedInstanceState);
      OnViewChangedNotifier.replaceNotifier(previousNotifier);
      setContentView(R.layout.main);
   }
```

```
  @Override
  public<T extends View> T internalFindViewById(int id) {
    return ((T) this.findViewById(id));
  }

  private void init_(Bundle savedInstanceState) {
    Resources resources_ = this.getResources();
    ..........
    OnViewChangedNotifier.registerOnViewChangedListener(this);
    ..........
    injectExtras();
  }
    ..........
}
```

6.3.4. Diagramas

A continuación mostramos los diferentes esquemas, tanto de la base de datos como de las clases de la aplicación Android.

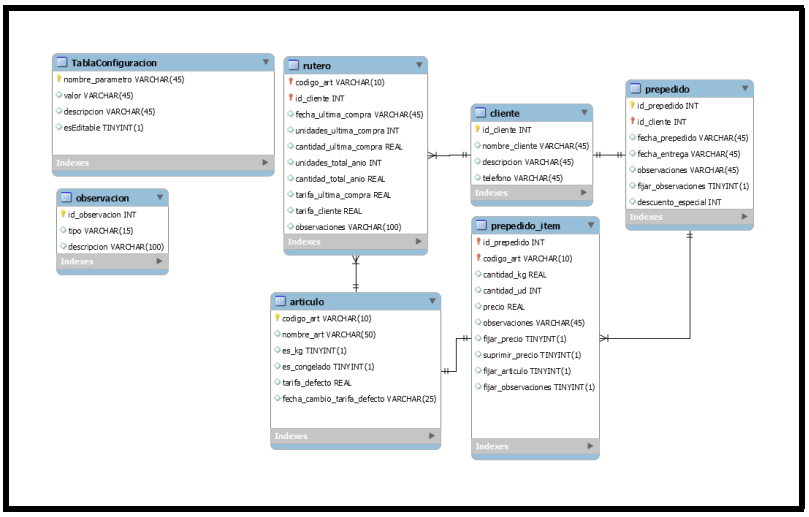

Figura 6.6.: Esquema de la base de datos SQLlite

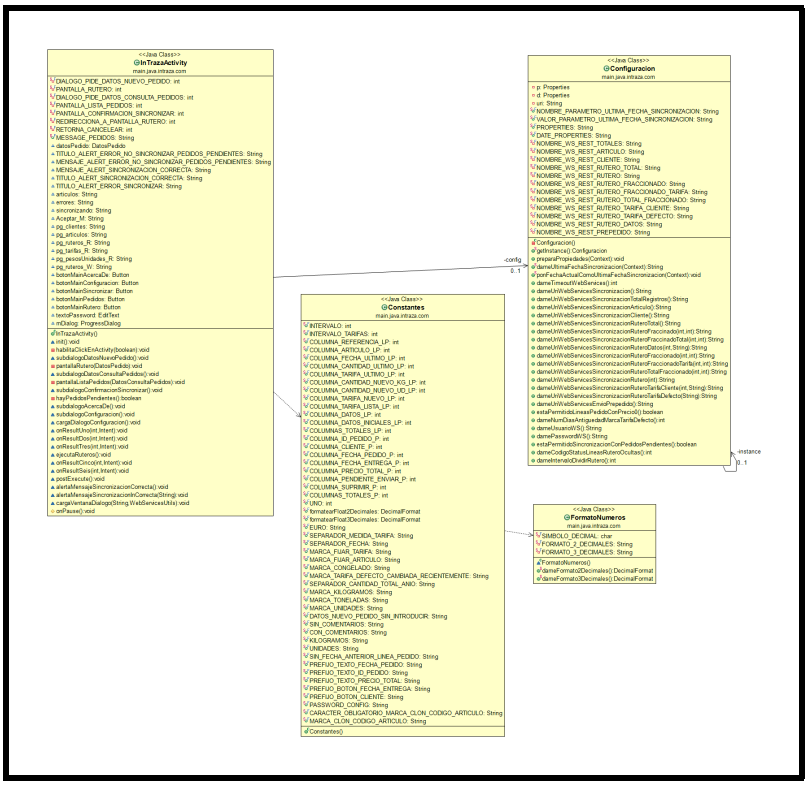

Figura 6.7.: Esquema principal del proyecto Android

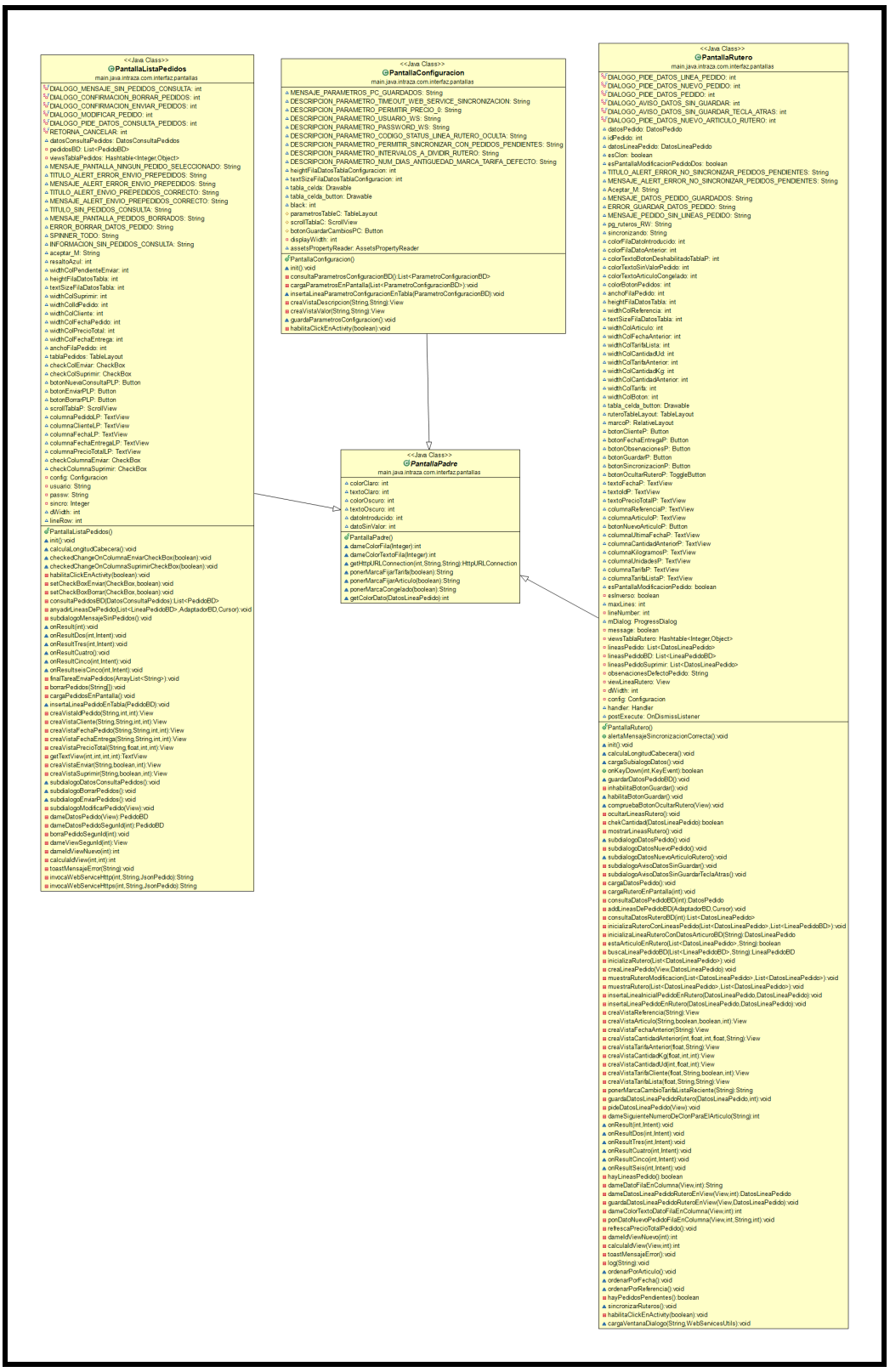

Figura 6.8.: Relaciones de datos

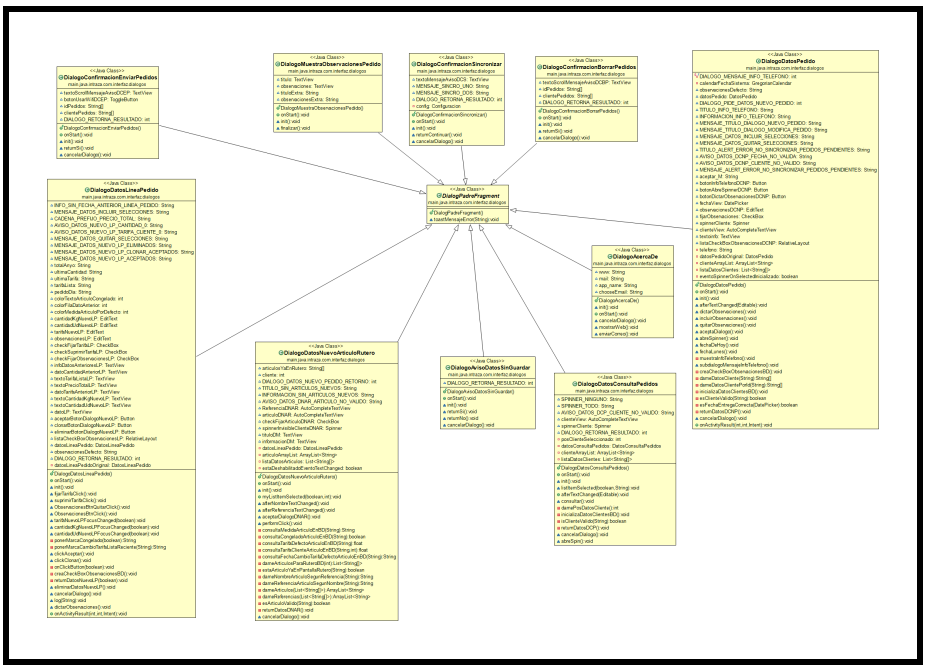

Figura 6.9.: Relaciones de datos

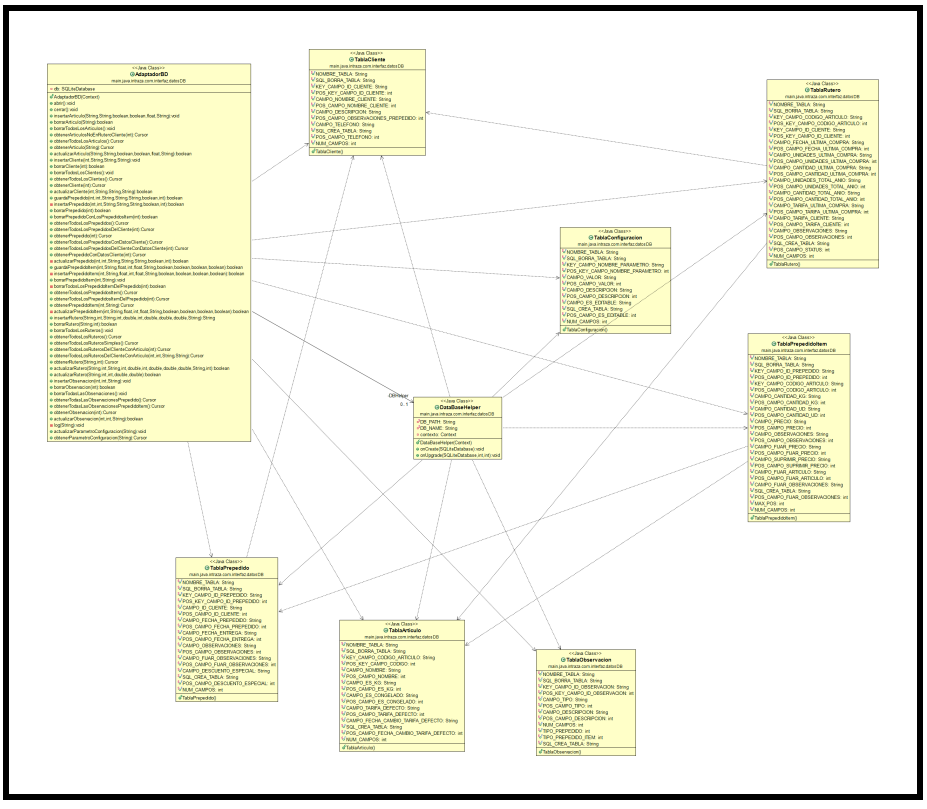

Figura 6.10.: Relaciones JSON

6.3.5. Pruebas

Debido a la imposibilidad de realizar las simulaciones, frente a un entorno de desarrollo de la empresa cliente, por las razones ya expuestas anteriormente, se decide crear una estructura de simulación propia.

6.3.5.1. Desarrollo del ERP de simulación

Una de las cosas a tener en cuenta, a la hora de desarrollar el ERP[8.1.4] de pruebas, es que debe ser rápido en su codificación y con una complejidad suficiente para poder realizar todas las pruebas de comunicación.

Con esas premisas, se opta por realizar el aplicativo usando un Framework ligero. La elección es GRAILS[8.1.6] , un Framework escrito en Groovy y compatible con los entornos JAVA.

El uso de un lenguaje dinámico, de sintaxis parecida a JAVA y su principal peculiaridad de que abarca las tres capas del desarrollo web: acceso a base de datos, capa de negocio y vista, permiten acortar los tiempos de desarrollo notablemente.

Ademas de lo anteriormente dicho, añadiendo herencias y usando reflexión nos encaminan a disponer un producto totalmente operativo en unos pocos días.

6.3.5.2. Estructura Básica

Definiremos una estructura básica compuesta por :

- **Clases dominio:** Serán el equivalente a las tablas en la base de datos.
- **Controladores:** Serán los encargados de trazar los flujos de operación, al navegar por las diferentes vistas.
- **Servicios:** Serán los encargados de proporcionar todas las operaciones de comunicación con agentes externos, filtros y demás código necesario.
- **Vistas:** Serán las que proporciones las pantallas de la vista del modelo MVC.
- **TagLibs:** Serán librerías de utilidades para agilizar o simplificar procesos mas complejos en las vistas.

Aquí tenemos un ejemplo de código perteneciente a la clase controlador principal. Corresponde a una acción de listado genérica.

```
private doList() {
  //offset
  doBefore()
  def entidad = clazz

  if (entidad) {
    instanceTotal = entidad?.count()
    params.max = Math.min(params.max ? params.int('max') : 10, 100)

  if (params.offset)
    offset = params.offset
  else if (instanceTotal > offset)
    params.offset = offset

  if (params.sort || !sort)
    instanceList = entidad.list(params)
  else
    instanceList = entidad.list(max: params?.max, offset:
       params?.offset, sort: sort[0], order: sort[1])
  }
  render(view: 'list', model: mapaVariables(), params: params)
}
```

Aquí un ejemplo de acciones de un controlador para el dominio Pedido. Se puede ver que la mayoría del código es procesado por una clase padre, y solo definimos particularidades si lo necesitamos. La creación de nuevas entidades se simplifica, y disponemos de un alto porcentaje de reutilización de código.

```
package pedido
import controller.MainController

class PedidoController extends MainController {
  def pedidoService
  def index()    { super.doIndex()     }
  def list()     { super.doList()     }
  def create()    { super.doCreate()     }
  def save()     { super.doSave()      }
  def show()     { super.doShow()      }
  def edit()     { super.doEdit()      }
  def update()    { super.doUpdate()    }
  def delete()    { super.doDelete()    }
  def limpiarFiltro() { super.doLimpiarFiltro() }
  def ajx_filtrar() { super.doAjx_filtrar() }
}
```

6.3.5.3. Diseño de la base de datos ERP

Nos planteamos un diseño base de datos sencillo con las tablas básicas que debería tener el ERP. Evidentemente un ERP comercial tendrá mayor complejidad, pero este desarrollo es para realizar un entorno de pruebas. En la figura siguiente planteamos el esquema de tablas y relaciones.

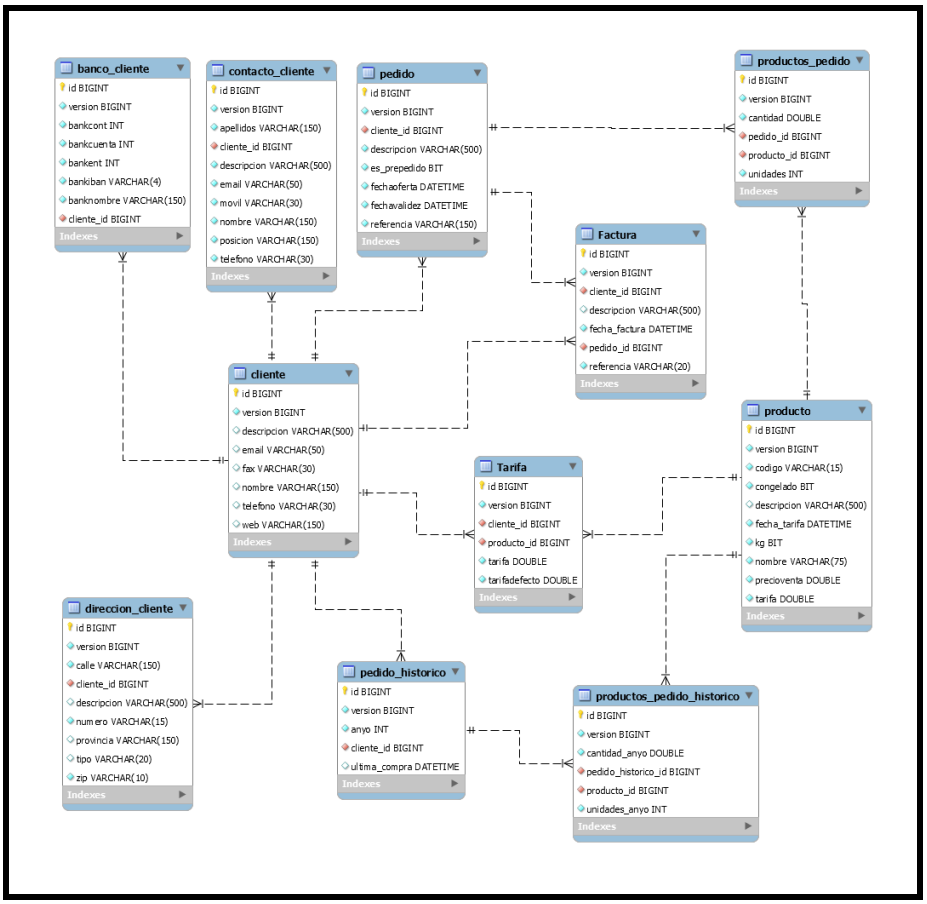

Figura 6.11.: Modelo de datos del ERP

Parte VI.

Conclusiones

7 Conclusiones

7.1. Introducción

Durante la realización del proyecto se procede a la ejecución de las diferentes fases. Analizando la viabilidad, planificando a traves de metodologías ágiles SCRUM, se describen y enumeran las tecnologías y fundamentos teóricos. Se analiza la estructura que evalúa los resultados. En los anexos se definen y aclaran conceptos, además de un manual de usuario y diferentes guías de instalación y uso de los diferentes componentes.

7.2. Conclusiones

A continuación vamos a recordar cuál era el Objetivo principal del proyecto marcado al inicio del mismo:

Construir una aplicación en Android que permita operar con el gestor de pedidos del ERP de la compañía de manera remota. A la finalización del proyecto, se puede afirmar que se ha conseguido satisfacer completamente el Objetivo principal. El grado de consecución se puede considerar bastante alto, resaltando el hecho de que la empresa empezó a operar con la app en la campaña de verano, sin incidentes reseñables.

El sistema/módulo desarrollado cumple perfectamente el objetivo propuesto y además está totalmente preparado para realizar futuras ampliaciones e integraciones.

La consecución del Objetivo principal incluía la satisfacción de unos sub-objetivos que van totalmente ligados. A continuación se describen los que se han satisfecho:

- El principal objetivo a cumplir a nivel personal fue adquirir conocimientos sobre

el entorno y el contexto del proyecto.

Siendo estos variados como:

- ***Conocer las principales características del lenguaje Java:*** Sin tener un previo conocimiento básico del lenguaje Java habría sido imposible programar la aplicación en Android.
- ***Conocer las principales características de Android:*** Al igual que en el punto anterior, si no se hubiera estudiado y entendido las principales características de Android, así como su funcionamiento, llevar a cabo este proyecto con el desarrollo de la aplicación habría sido imposible.
- ***Estudiar el entorno de desarrollo de Android:*** Con la elección del entorno Android studio, fue necesario estudiarlo, entenderlo y manipularlo con la suficiente soltura que nos permitiera el correcto desarrollo de la aplicación.
- ***Conocer funcionamiento y desarrollo de aplicaciones J2EE:*** Sin este conocimiento, habría sido imposible desarrollar el servicio web de conexión.
- ***Conocer funcionamiento y manejo de bases de datos:*** Sin este conocimiento, no podríamos abordar la persistencia de datos solicitada.
- ***Conocer funcionamiento y manejo de servidores de aplicaciones java:*** Era necesario conocer los diferentes entornos de hospedaje de aplicaciones, ya sea tomcat, jboss, websphere, etc.

Otro de los objetivos a cumplir en este proyecto era la realización de la memoria de proyecto debidamente cumplimentada con el reglamento existente.

7.3. Planificación temporal final

Por otro lado estaba el factor temporal. El proyecto se debía ajustar a una planificación realizada al comienzo del mismo. La duración era de 3 meses aproximadamente. Ya que se quería tener operativa la aplicación antes de la campaña de verano, de vital importancia para la empresa.

Pero por lo general, a la finalización del proyecto, se puede afirmar que se ha logrado seguir bastante bien la planificación temporal realizada en su comienzo, siempre con las pequeñas y lógicas desviaciones.

7.4. Conclusión final

Con lo visto anteriormente, podemos afirmar que, a la finalización del proyecto, el grado de consecución de objetivos es bastante alto y que se ha logrado ajustar

bastante bien su desarrollo en el espacio temporal planificado. Por tanto, se puede concluir que el proyecto ha finalizado con éxito.

Parte VII.

Anexos

8

ANEXO I

8.1. Conceptos y aclaraciones

8.1.1. Maven

Maven es una herramienta de software para la gestión y construcción de proyectos Java creada por Jason van Zyl, de Sonatype, en 2002. Es similar en funcionalidad a **Apache Ant** , pero tiene un modelo de configuración de construcción más simple, basado en un formato XML. Estuvo integrado inicialmente dentro del proyecto Jakarta pero ahora ya es un proyecto de nivel superior de la Apache Software Foundation.

Maven utiliza un **POM*(Project Object Model)*** para describir el proyecto de software a construir, sus dependencias de otros módulos y componentes externos, y el orden de construcción de los elementos. Viene con objetivos predefinidos para realizar ciertas tareas claramente definidas, como la compilación del código y su empaquetado. Una característica clave de Maven es que está listo para usar en red. El motor incluido en su núcleo puede dinámicamente descargar plugins de un repositorio, el mismo repositorio que provee acceso a muchas versiones de diferentes proyectos Open Source en Java, de Apache y otras organizaciones y desarrolladores. Este repositorio y su sucesor reorganizado, el repositorio Maven 2, pugnan por ser el mecanismo de facto de distribución de aplicaciones en Java, pero su adopción ha sido muy lenta. Maven provee soporte no solo para obtener archivos de su repositorio, sino también para subir artefactos al repositorio al final de la construcción de la aplicación, dejándola al acceso de todos los usuarios. Una caché local de artefactos actúa como la primera fuente para sincronizar la salida de los proyectos a un sistema local. [wikipedia(2017g)]

8.1.2. Gradle

Gradle es un sistema de automatización de construcción de código abierto que construye sobre los conceptos de Apache Ant y Apache Maven e introduce un lenguaje especifico del dominio (DSL) basado en Groovy en vez de la forma XML utilizada por Apache Maven para declarar la configuración de proyecto. **Gradle** utiliza un **DAG *(Grafo Acíclico Dirigido)*** para determinar el orden en el que las tareas pueden ser ejecutadas. **Gradle** fue diseñado para construcciones multi-proyecto las cuáles pueden crecer para ser bastante grandes, y da apoyo a construcciones incrementales determinando inteligentemente qué partes del árbol de construcción están actualizadas, de modo que cualquier tarea dependiente a aquellas partes no necesitarán ser re-ejecutada. [wikipedia(2017c)]

8.1.3. Anotación Java

En programación, una Anotación Java es una forma de añadir meta-datos al código fuente Java que están disponibles para la aplicación en tiempo de ejecución. Muchas veces se usa como una alternativa a la tecnología XML.

Las Anotaciones Java pueden añadirse a los elementos de programa tales como clases, métodos, campos, parámetros, variables locales, y paquetes. Al contrario que las etiquetas añadidas a la documentación Java y procesadas con las herramientas tales como XDoclet, las Anotaciones Java son completamente accesibles al programador, mientras que el software se ejecuta, usando reflexión.

8.1.4. ERP

El término ERP se refiere a Enterprise Resource Planning, que significa sistema de planificación de recursos empresariales. Estos programas se hacen cargo de distintas operaciones internas de una empresa, desde producción a distribución o incluso recursos humanos.

Además, los ERP ofrecen integración con soluciones de BI o Business Intelligence, permitiendo realizar informes sobre el estado de su empresa directamente con los datos del sistema ERP. Esto ofrece un nivel de conocimiento detallado y actualizado del estado de la empresa que resulta indispensable a la hora de analizar y mejorar procesos internos como el marketing y ventas, la organización u otros aspectos clave de una compañía.

8.1.5. Framework

Un Framework es básicamente un entorno de desarrollo, en el que los podemos desarrollar mucho mas fácil y rápidamente aplicaciones de todo tipo, incluso las aplicaciones web. Un Framework puede estar constituido de librerías de código fuente, utilidades, plugins, modelos de desarrollo, y todo tipo de herramientas cuyo único propósito de acelerar el ritmo de desarrollo de una aplicación.

8.1.6. Grails

Grails (Groovy and rails). Es básicamente un Framework de desarrollo de aplicaciones web, en el que usamos principalmente el lenguaje de Java y Groovy, incluso se puede combinar el código.Es un Framework que funciona bajo un modelo conocido como MVC (modelo vista controlador), en el que principalmente lo que se hace es crear “Controladores” que son como servicios que manipulan nuestra aplicación web, y todo el código de estos controladores es ejecutado en el servidor web.

Ademas Grails utiliza plantillas y vistas. No solo podemos programar controladores, si no que podemos utilizar un lenguaje llamado groovy server pages(GSP) para poder programar directamente sobre lo que seria nuestra pagina web con código HTML, es como si programáramos en jsp o php, esto nos da mucha flexibilidad y orden al programar nuestras aplicaciones web.

Dispone de gran cantidad de plugins para realizar múltiples desarrollos, conversores, etc...

9 ANEXO II

9.1. Manual de usuario

9.1.1. Pantalla principal

Esta pantalla se compone de 5 botones de acción.

- **Crear pedidos:** Lanza el proceso para la creación de pedidos.
- **Gestión de pedidos:** Lanza el proceso para la gestión de pedidos.
- **Sincronizar con InTraza:** Lanza el proceso de sincronización con el ERP.
- **Acerca de:** Ventana informativa.
- **Configuración:** Por medio de una contraseña permite cambiar algunos parámetros de la aplicación.

Figura 9.1.: Intraza - pantalla principal

9.1.2. Creación de pedidos

En esta pantalla elegiremos el cliente por medio del combo, además de la fecha de entrega y observaciones del pedido. Las observaciones pueden dictarse, escribirse o elegirse de una lista predefinida para ese cliente. Cuando seleccionas fijar observaciones añades estas a la lista de observaciones prefijadas.

Figura 9.2.: Intraza - creación de pedidos

Una vez validada la pantalla anterior, nos muestra una pantalla a la que llamaremos ruteros que muestra las lineas de pedido de la empresa seleccionada y nos permite crear adicionales. Podremos mostrar/ocultar las lineas antiguas y volver a la pantalla anterior si queremos cambiar algo, por medio de los botones. Ademas podremos sincronizar los ruteros de la empresa seleccionada.

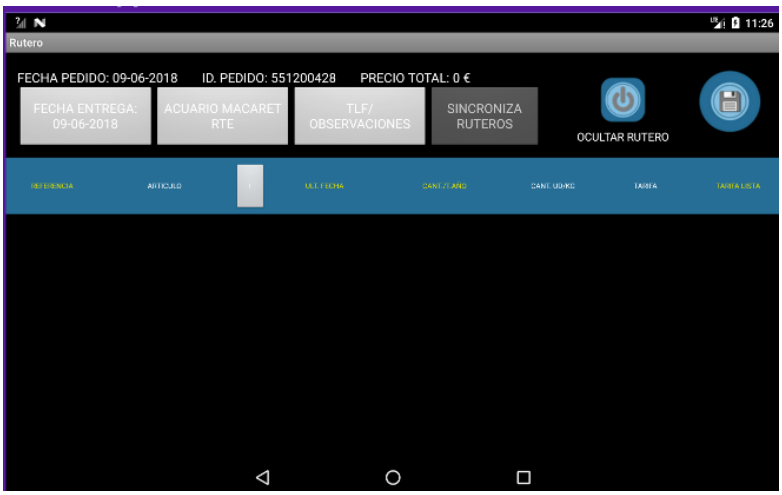

Figura 9.3.: Intraza - ruteros

Una vez pulsado el botón de añadir artículo '(+)' nos mostrará un ventana de dialogo para elegir el artículo que queremos introducir.

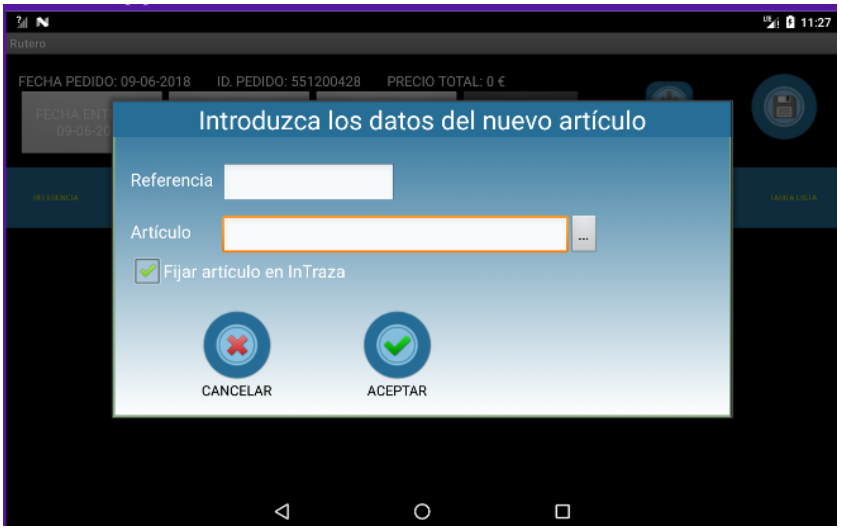

Figura 9.4.: Intraza - introducir articulo

Una vez aceptado el articulo nos mostrará la pantalla para rellenar los campos necesarios para el pedido. Ademas nos permitirá clonar un rutero para pedidos repetitivos.

Figura 9.5.: Intraza - rellenar cantidades/observaciones

Una ver añadido el articulo podemos comprobar que la pantalla ruteros dispone de una linea con los datos del mismo.

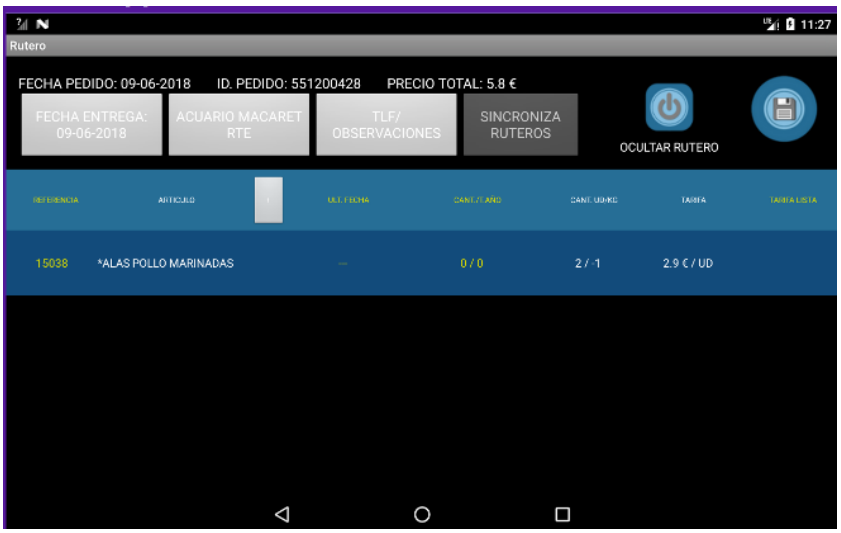

Figura 9.6.: Intraza - articulo insertado

9.1.3. Gestión de pedidos

En esta pantalla podremos elegir todos los pedidos o elegir los pedidos del cliente que queremos visualizar.

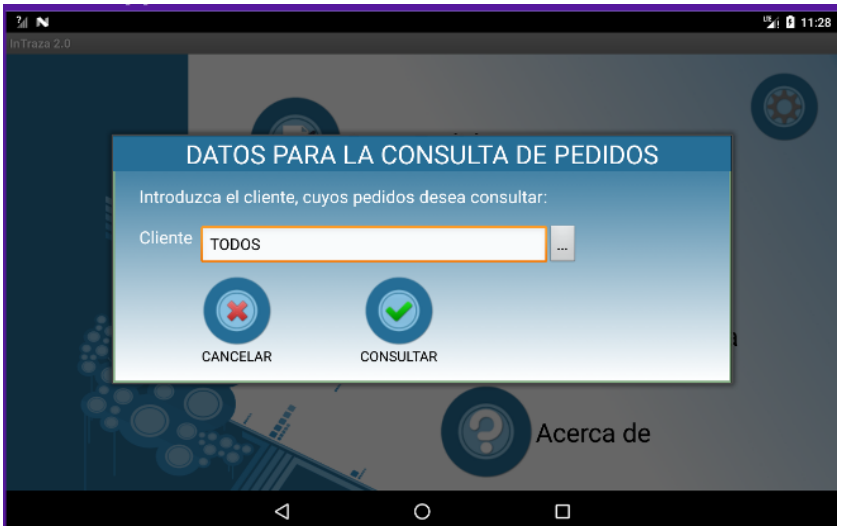

Figura 9.7.: Intraza - gestión de pedidos

En esta pantalla podremos gestionar los pre-pedidos que están almacenados en el dispositivo. Podremos ver su contenido clickando en el id de pedido. Ademas podremos borrarlos o enviarlos a la central, para que pasen a ser parte del ERP.

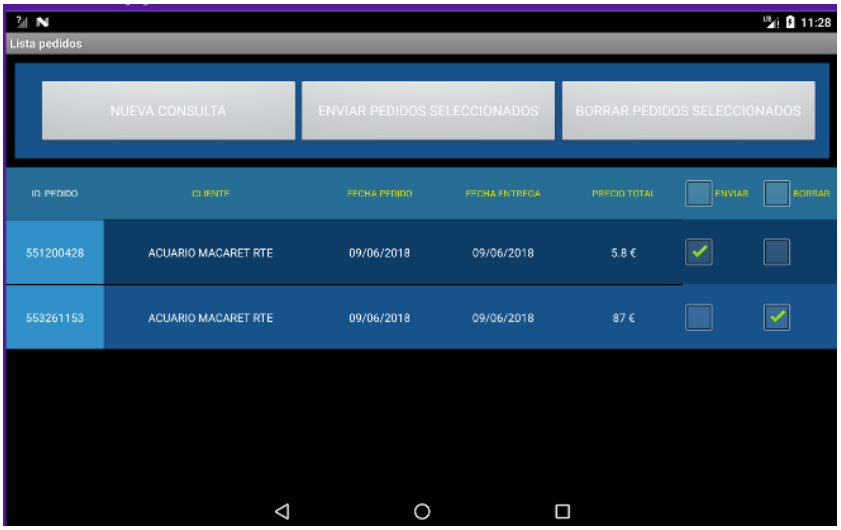

Figura 9.8.: Intraza - lista de pedidos

9.1.4. Sincronizar con Intraza

Por medio de esta pantalla podremos sincronizar los datos guardados en el ERP, con los datos de la aplicación. Sincronizará clientes, artículos y ruteros.

Figura 9.9.: Intraza - sincronización

9.1.5. Configuración

En esta pantalla podremos configurar el valor de algunos parámetros de la aplicación. Como por ejemplo la ruta del WebService de comunicaciones. Esta pantalla está protegida con contraseña.

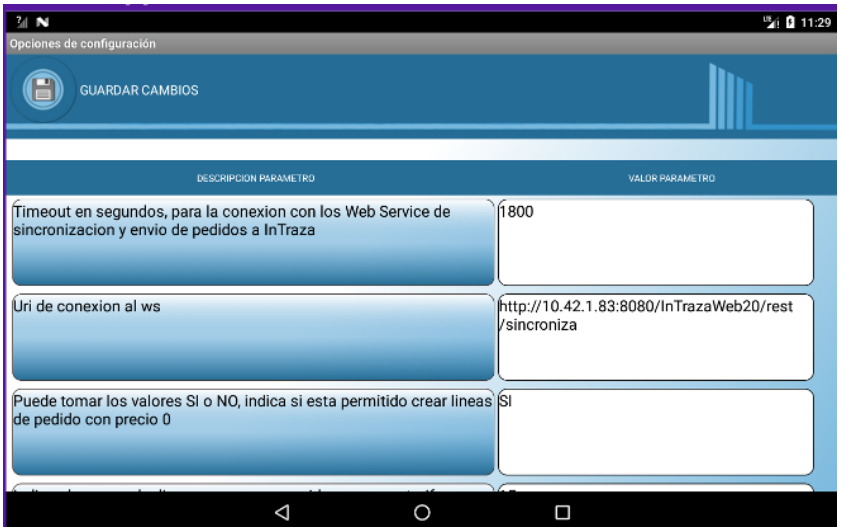

Figura 9.10.: Intraza - configuración

10 ANEXO III

10.1. Guías de Instalación

10.1.1. Instalación del Sistema Máquina Virtual Ubuntu

10.1.1.1. Máquina virtual Ubuntu

Una vez instalado la aplicación virtual box, crearemos una maquina nueva. Pulsamos a nueva, Ubuntu. Como tipo seleccionamos: Linux y la versión la que deseemos: Ubuntu 64 bits.

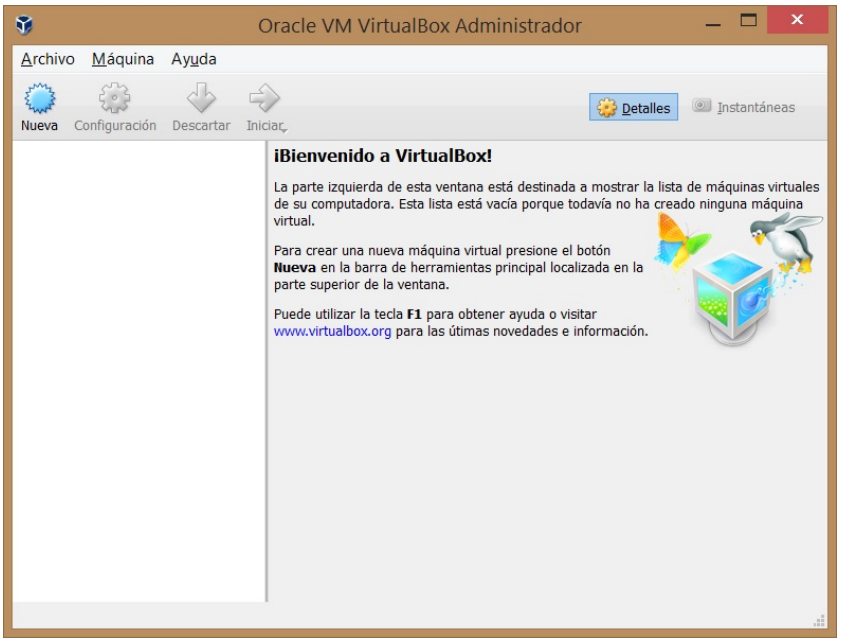

Figura 10.1.: Virtual box - inicio

Le asignamos la memoria RAM a la máquina virtual 2048 MB es el mínimo recomendado, yo he usado 4.096 MB. Seleccionamos la opción *Crear un disco duro virtual ahora*, ya que no tenemos ninguno creado anteriormente y hacemos clic en Crear y seleccionamos en la siguiente pantalla la opción VDI (Virtual BOX Disk Image) y con tamaño “Reservado Dinámico” para que ocupe lo que realmente necesite.

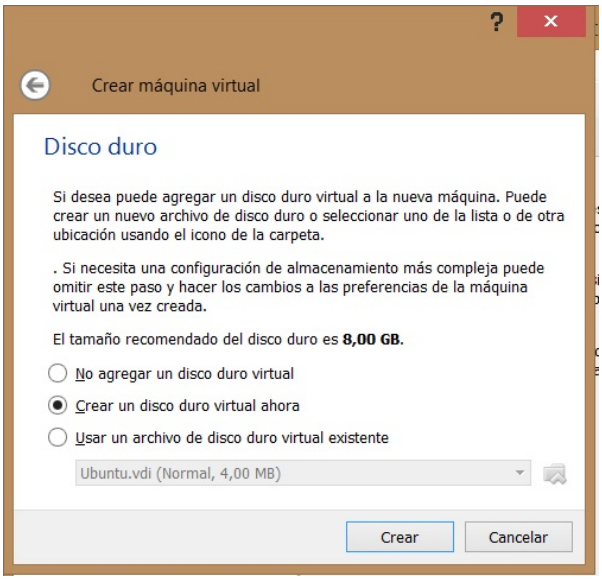

Figura 10.2.: Virtual box - crear disco duro

Ya tenemos creada la máquina virtual. Solo nos queda empezar a instalar Ubuntu en el siguiente apartado.

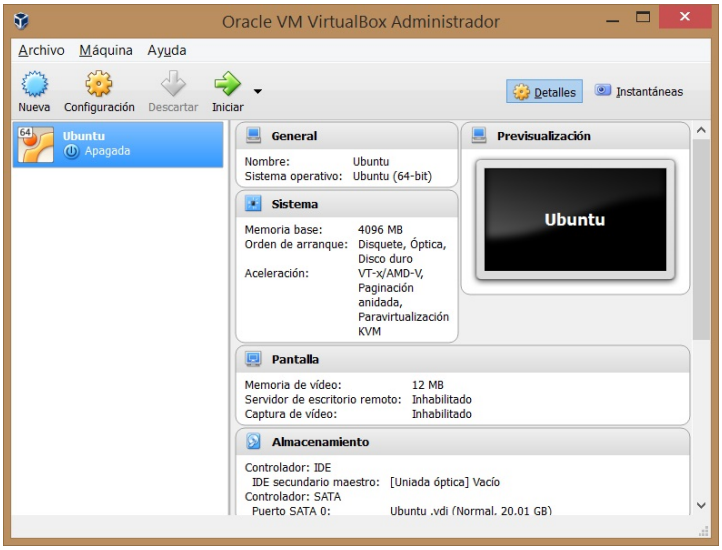

Figura 10.3.: Virtual box-vm creada

Pulsamos sobre *iniciar* para arrancar la máquina virtual y empezar la instalación de Ubuntu, nos pedirá que seleccionemos la imagen ISO que hemos descargado anteriormente. La instalación comienza con un asistente de instalación en el que debemos seleccionar el idioma a usar y pulsar en “instalar Ubuntu”. Su instalación ha sido bastante rápida, nos ha llevado un total de 9 minutos.

Figura 10.4.: Instalación ubuntu

En la siguiente pantalla nos da la opción de descargar las actualizaciones e instalar software de terceros para reproducir archivos multimedia y otros. A continuación, nos saldrá el asistente del particionado del disco duro, en este caso vamos a usar todo el disco virtual que hemos creado por lo que dejamos la opción por defecto y pulsamos en *instalar ahora* Configuramos las particiones por defecto, seleccionamos nuestra zona horaria y pulsamos en *continuar*, y elegimos nuestra distribución del teclado y *continuar*. Luego nos saldrá una última pantalla en la que deberemos poner nuestro nombre de usuario y contraseña. También podemos seleccionar la opción de inicio de sesión automático.

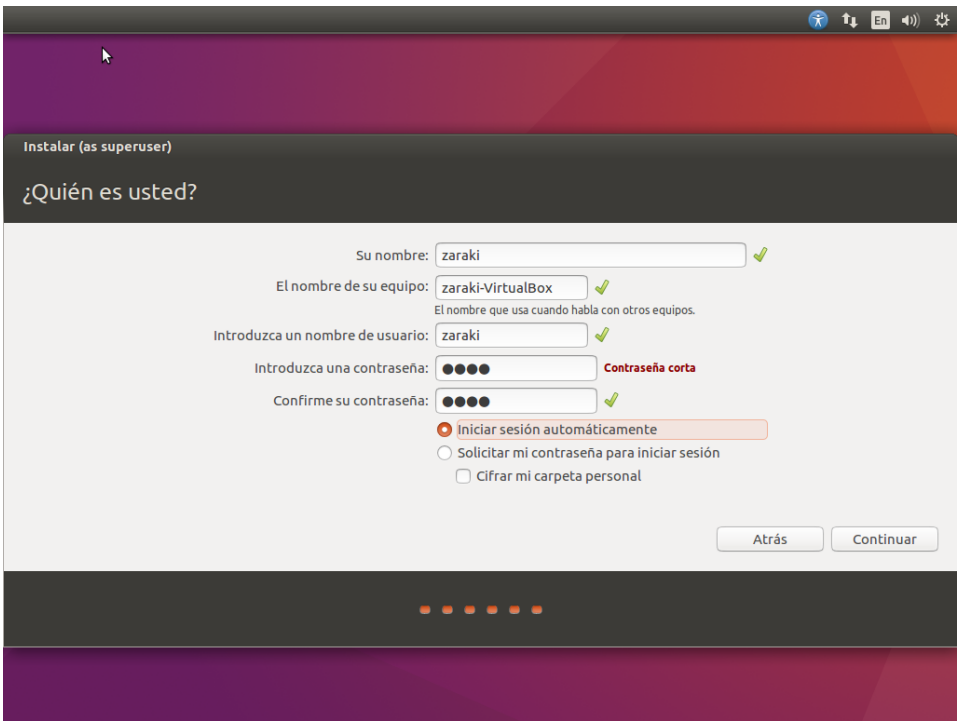

Figura 10.5.: Instalación ubuntu - usuario y password

Finalmente reiniciamos nuestro sistema y ya tendremos un Ubuntu 16.04 LTS funcionando.

10.1.2. Instalar Eclipse

Instalar el JDK de Oracle En un principio debemos verificar si disponemos de una versión de desarrollo de java instalada. Se detallan los comandos para verificar instalar en modo consola.

```
java -version
sudo apt-get update
sudo apt-get install python-software-properties
sudo add-apt-repository ppa:webupd8team/java
sudo apt-get install oracle-java8-installer
```

Una Vez instalada se puede comprobar la instalación.

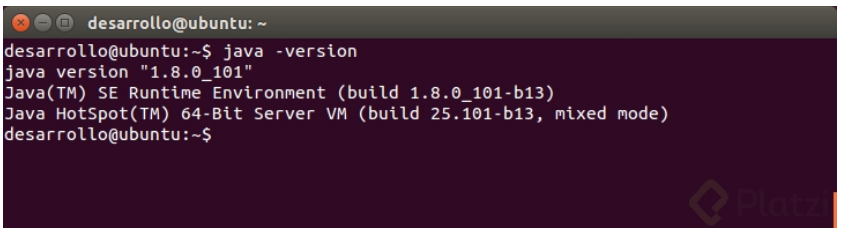

Figura 10.6.: Terminal ubuntu

Descargar e Instalar Eclipse Oxigen Ir a la página oficial y descargar la última versión para Linux 64 bits. Se extrae el archivo de la carpeta y se mueve la carpeta a otra ubicación más cómoda.

```
sudo mkdir -p /opt/ide/64
sudo mv ~/Downloads/eclipse /opt/ide/64
cd /opt/ide/64
sudo chown -R root:root eclipse
sudo ln -sf /opt/ide/64/eclipse/eclipse /usr/bin/
```

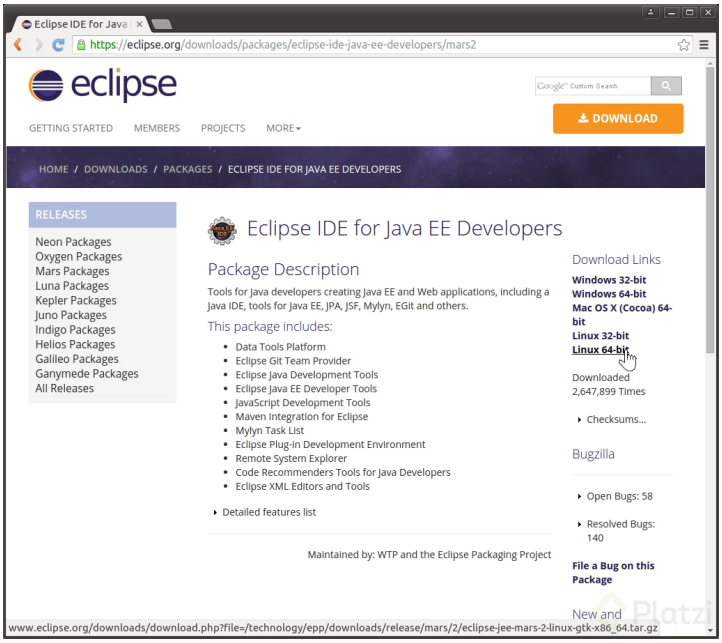

Figura 10.7.: Ventana Eclipse

Definir Eclipse como un programa para su acceso directo Se ejecuta el siguiente comando, lo cual abrirá una carpeta dónde se encuentran las aplicaciones del sistema con todos los permisos para administrarlos.

```
sudo nautilus /usr/share/applications
```

Para generar un acceso directo a eclipse. Eclipse.desktop

```
sudo mv /usr/share/applications/Eclipse.desktop /usr/share/
   applications/eclipse-mars-2.desktop
sudo gedit /usr/share/applications/eclipse-mars-2.desktop
[Desktop Entry]
Name=Eclipse Mars 2
Comment=IDE for Java
Exec=/usr/bin/eclipse
Icon=/opt/ide/64/eclipse/icon.xpm
Terminal=false
StartupNotify=true
Version=2.0
Type=Application
Categories=Development;Utility;
```

10.1.3. Instalar Android studio

Descarga de Android Studio de su página principal, Hacemos click sobre *Download Android Studio* y comenzará la descarga. Nos descargará un fichero comprimido el cual debemos colocar sobre el directorio desde el que queramos ejecutarlo. Instalación de Android Studio Una vez descomprimido el archivo podremos ver una carpeta similar a la siguiente. Debemos acceder a la carpeta android-studio/bin y ejecutar el script llamado studio.sh para arrancar el programa.

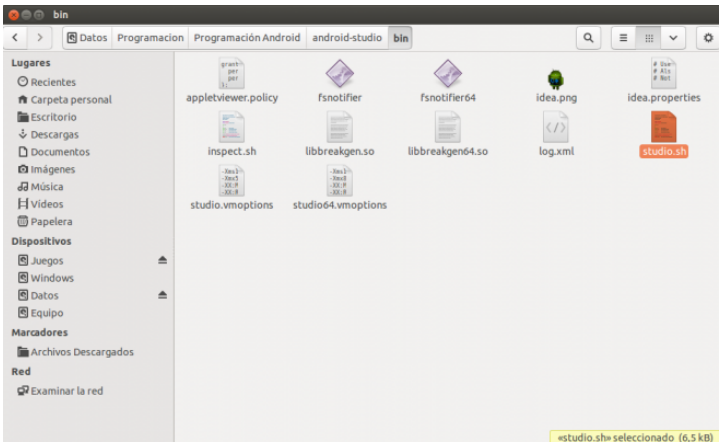

Figura 10.8.: Archivador ubuntu

Seleccionaremos *New project* para crear un nuevo proyecto desde el que podremos comenzar a utilizar la nueva IDE. Una vez finalizamos el asistente de inicio ya podremos ver la interfaz del programa.

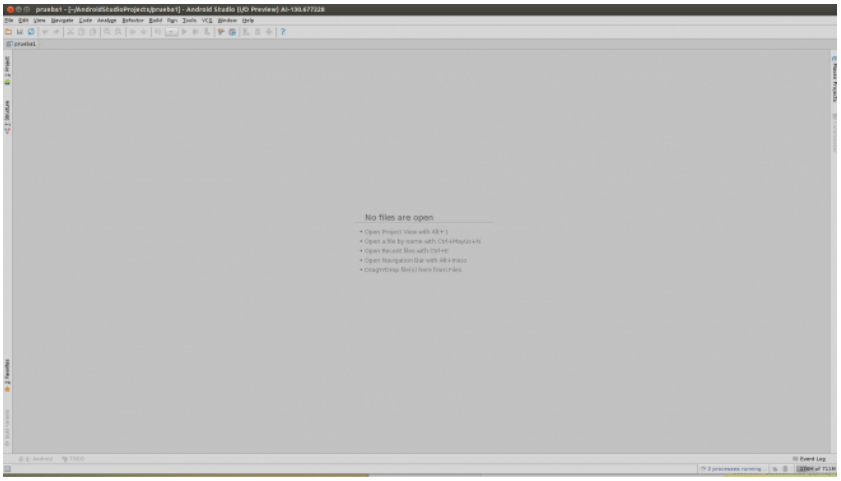

Figura 10.9.: Android studio

En caso de tener proyectos ya creados o empezados con Eclipse podemos importarlos de forma muy sencilla. Para ello debemos situarnos sobre el *menú file – import* y seleccionar el proyecto que queremos importar. Una vez allí seleccionaremos la opción *Create project from existing sources* y ya tendremos nuestro proyecto importado en nuestro nuevo IDE para continuar desarrollando desde él. Podemos ver que la interfaz de la nueva IDE es bastante simple y sencilla de utilizar.

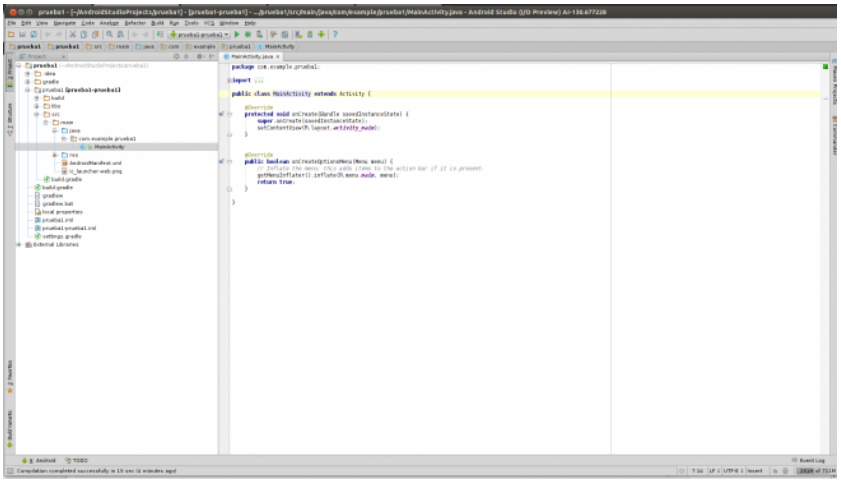

Figura 10.10.: Android studio - ejemplo de proyecto

Para crear un lanzador Primero abrimos la consola y digitamos lo siguiente:

```
sudo gedit /usr/share/applications/eclipse.desktop
```

Esto obviamente abrirá el gedit y en el ingresamos:

```
[Desktop Entry]
Name=Android Studio
Comment=Android Studio IDE
Exec=/home/tu_usuario/android-studio/bin/studio.sh
Icon=/home/tu_usuario/android-studio/bin/idea.png
Terminal=false
Type=Application
```

10.1.4. Instalar Xamp

Descarga el instalador de XAMPP. Puedes conseguirlo en apachefriends.org/download.html. Asegúrate de descargar la versión correcta para tu sistema (32 o 64 bits). En esta guía se utilizará como ejemplo la versión 5.6.3 de 64 bits. Asegúrate de cambiar los comandos según la versión que vayas a instalar. Abre la Terminal. Antes de poder instalar XAMPP, necesitarás cambiar los permisos de modo que puedas ejecutar el archivo que descargaste. Cambia los permisos. Ingresa el siguiente comando y presiona Enter, escribiendo también tu contraseña de ser necesario:

```
sudo chmod +x xampp-linux-x64-5.6.3-0-installer.run
```

Puedes arrastrar el archivo descargado hacia la ventana de la Terminal para ingresar el nombre y la ubicación del archivo en forma automática. Ejecuta el instalador. Después de cambiar los permisos, podrás ejecutar el instalador para comenzar a instalar XAMPP. Escribe el siguiente comando y presiona Enter:

```
sudo ./xampp-linux-x64-5.6.3-0-installer.run
```

Sigue las instrucciones para instalar XAMPP. Se abrirá el instalador gráfico para guiarte a través del resto del proceso de instalación. En la mayoría de los casos, es mejor dejar todas las opciones con su configuración predeterminada.

10.1.5. Instalar Tomcat

Instalamos Tomcat en modo terminal.

```
sudo apt-get install tomcat8
```

Editamos el archivo de configuración del bash:

```
sudo nano ~/.bashrc
export JAVA_HOME=/usr/lib/jvm/default-java
export CATALINA_HOME=/var/lib/tomcat7
```

Por último nos quedaría modificar el archivo de usuarios:

```
sudo nano /var/lib/tomcat7/conf/tomcat-users.xml
Deberemos dejarlo parecido a:
<tomcat-users>
<role rolename="admin-gui"/>
<role rolename="admin-script"/>
<role rolename="manager-gui"/>
<role rolename="manager-status"/>
<role rolename="manager-script"/>
<role rolename="manager-jmx"/>
<user username="admin" password="1234" roles="standard,manager-gui,
   manager-status,manager-script,manager-jmx,admin-gui,admin-script"
    />
</tomcat-users>
```

Por último, sólo tendríamos que reiniciar el servicio de Tomcat (sudo service tomcat8 restart) y ya podríamos acceder a nuestro Tomcat desde cualquier navegador poniendo la siguiente ruta: localhost:8080 En la que nos aparecerá el archivo por defecto con unos enlaces a los ejemplos, documentación, etc. . .

Si hemos instalado también el paquete de administración, podremos de una manera sencilla ver, cambiar y desplegar nuestras aplicaciones Java, desde http://localhost: 8080/manager/html introduciendo el usuario y la contraseña que hayamos puesto en el archivo de configuración.

Figura 10.11.: Tomcat Manager

11

ANEXO IV

11.1. Configuracion de los proyectos cliente y servidor

11.1.1. Creación del proyecto web service

Una vez arrancado el entorno Eclipse, vamos a crear un proyecto nuevo (en el menu Archivo -¿Nuevo -¿Dynamic Web Project.), donde definiremos el nombre de nuestro proyecto, el runtime al que se destina nuestro proyecto, en este caso al servidor tomcat que instalamos en nuestro sistema. Después agregamos al proyecto las librerias de JAX-RS a nuestro proyecto en el boton de “configurar” – “Modificar”.

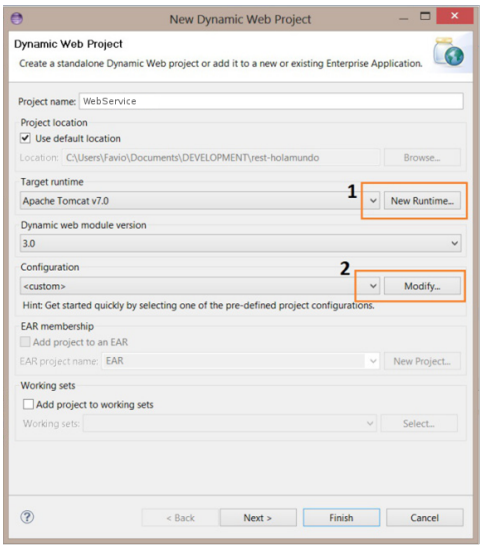

Figura 11.1.: WebService - creación

En el directorio src de nuestro proyecto agregamos la siguiente clase “LoggerWS”, dentro del paquete com.group.six:

```
package com.intraza.rest;

import javax.ws.rs.Consumes;
import javax.ws.rs.GET;
import javax.ws.rs.POST;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
import javax.ws.rs.QueryParam;
import javax.ws.rs.core.MediaType;

import org.apache.log4j.Logger;
import org.codehaus.jackson.map.ObjectMapper;

import com.intraza.rest.db.*;
import com.intraza.rest.db.datos.*;

@Path(value = "/sincroniza")
public class InTrazaWS {

  private static Logger logger = Logger.getLogger(InTrazaWS.class);

  @GET
  @Path("articulos")
  @Produces(MediaType.APPLICATION_JSON)
  public List<Articulo> consultaArticulosBD() {
    return JDBCQuery.getArticulos();
  }

  ..............

  @POST
  @Path("prepedido")
  @Consumes(MediaType.APPLICATION_JSON)
  @Produces(MediaType.APPLICATION_JSON)
  public ResultadoEnvioPedido enviaPrepedidoBD(String
     jsonPrepedido) {
    ResultadoEnvioPedido resultadoEnvio = null;
    try {
      // Convertimos el JSON que nos llega a un objeto java
      ObjectMapper mapper = new ObjectMapper();
      JsonPedido datosPrepedido = mapper.readValue(jsonPrepedido,
         JsonPedido.class);

      logger.debug("##### observaciones (" +
```

```
            datosPrepedido.getObservaciones() + ") ######");

        resultadoEnvio = JDBCQuery.postPrepedido(datosPrepedido);
      } catch (Exception e) {
        resultadoEnvio = new
           ResultadoEnvioPedido(ResultadoEnvioPedido.CON_ERROR,
        "Se ha producido una excepcion al decodificar JSON (" +
           jsonPrepedido + ") (" + e.toString() + ")");
      }

      return resultadoEnvio;
    }
```

Ahora es necesario agregar al archivo web.xml, que eclipse nos genero sobre el directorio WEB-INF de nuestro proyecto, la configuración de Jersey, y el servlet mapping que es la ruta donde se invocara a nuestro servicio, el archivo queda asi:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
   http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
id="WebApp_ID" version="3.0">
<display-name>InTrazaWeb20</display-name>

<servlet>
  <servlet-name>IntrazaWeb</servlet-name>
  <servlet-class>com.sun.jersey.spi.container.servlet.
     ServletContainer</servlet-class>
  <init-param>
    <param-name>com.sun.jersey.config.property.
       packages</param-name>
    <param-value>com.intraza.rest</param-value>
  </init-param>
  <init-param>
    <param-name>com.sun.jersey.api.json.
       POJOMappingFeature</param-name>
    <param-value>true</param-value>
  </init-param>
  <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
  <servlet-name>IntrazaWeb</servlet-name>
  <url-pattern>/rest/ *</url-pattern>
</servlet-mapping>
```

```
<welcome-file-list>
<welcome-file>index.jsp</welcome-file>
</welcome-file-list>
</web-app>
```

Para poder compilar y correr nuestro código es necesario antes agregar a nuestro proyecto la implementación Jersey a las librerías de nuestro proyecto, para ello hay que descargarlas del sitio de Jersey, aqui. Para después copiar los jar en el directorio WEB-INF/libs. Otra opción es convertir el proyecto a un proyecto maven, y usar las configuracion a traves del archivo pom.

Para visualizar nuestra aplicación, podemos publicar el servlet al dar Run as - ¿Run on server. Si aún no tenemos configurado nuestro servidor tomcat, el wizard nos guiara.

La URL donde encontraremos nuestro servicio esta construida de la siguiente manera: http://nuestroservidor:puerto/contextroot/servletmapping/class-path

- Nuestroservidor: es el servidor al que publicamos nuestro servlet, con su puerto en el que funciona
- Contextroot: es generalmente el nombre de nuestro proyecto, en las propiedades de nuestro proyecto podemos cambiar esto
- Servletmapping: es la ruta que definimos en el archivo web.xml donde estaran nuestras clases.
- Class-path: es el nombre de la ruta que definimos en nuestra clase con @Path("/class-path")

11.1.2. Integrar un cliente web service

Por la propia particularidad del web service, un servicio REST, no necesitamos generar un cliente web especifico solo tenemos que tener en cuenta el empaquetado de datos y enviarlos a una url especifica del servicioWeb. Simplemente debemos incluir una clase que gestione la comunicacion. En el codigo siguiente se puede ver un ejemplo que envía al webService un Conjunto de datos a una url especifica.

```
package com.six.group.listener.utils;

import android.os.StrictMode;

import com.fasterxml.jackson.databind.ObjectMapper;
import com.six.group.listener.data.json.Datos;

import java.io.BufferedReader;
```

```
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.OutputStream;
import java.net.HttpURLConnection;
import java.net.URL;

public class WebServicesUtils {
private static Integer sincro = 1800;
private static String urlWebServiceRest;


public WebServicesUtils(String puerto, String ip) {
urlWebServiceRest = "http://" + ip + ":" + puerto +
   "/WebService/enviar/";
}

public String invocaWebServiceHttp(final Datos datos, String
   action) throws Exception {

StrictMode.ThreadPolicy policy = new
   StrictMode.ThreadPolicy.Builder()
.detectAll()
.penaltyLog()
.build();
StrictMode.setThreadPolicy(policy);

String result = "";
String urlToInvocate = urlWebServiceRest + action;

System.out.println("Sincronizacion : TRAZA - URL (" + urlToInvocate
   + ") segundo timeout (" + sincro + ")");

final URL url = new URL(urlToInvocate);
final HttpURLConnection connection = (HttpURLConnection)
   url.openConnection();
connection.setConnectTimeout(sincro * 1000);
connection.setReadTimeout(sincro * 1000);
connection.setRequestMethod("POST");
connection.setDoOutput(true);
connection.setRequestProperty("Content-Type", "application/json");
final ObjectMapper mapper = new ObjectMapper();
final String jsonRequest = mapper.writeValueAsString(datos);
final OutputStream os = connection.getOutputStream();
os.write(jsonRequest.getBytes());
os.flush();
final InputStream content = connection.getInputStream();
```

```
final BufferedReader in = new BufferedReader(new
   InputStreamReader(content));
String line;
while (null != (line = in.readLine())) {
result += line;
}
return result;
}
}
```

12 ANEXO V

12.1. Compilación de los proyectos

12.1.1. Compilación Android

Lo primero que debemos hacer es abrir nuestro proyecto en Android Studio y asegurarnos de que no hay ningún error de código ni de compilación ya que, de lo contrario, no se completará la compilación.

Si todo está correcto (no tenemos nada marcado en rojo en el código) abriremos el menú *Build* de la parte superior de la pantalla y veremos dos opciones:

- Build APK
- Generate Signed APK

La primera opción nos va a permitir generar un archivo apk para instalarlo en un dispositivo, pero el archivo no estará firmado. La segunda opción nos permitirá generar un archivo de forma (o utilizar uno existente) para firmar digitalmente nuestra aplicación. Esta opción es para subir la app al google play. En nuestro caso vamos a seleccionar directamente la segunda opción, la de generar el archivo APK firmado digitalmente. Pulsamos sobre ella y veremos una nueva ventana similar a la siguiente.

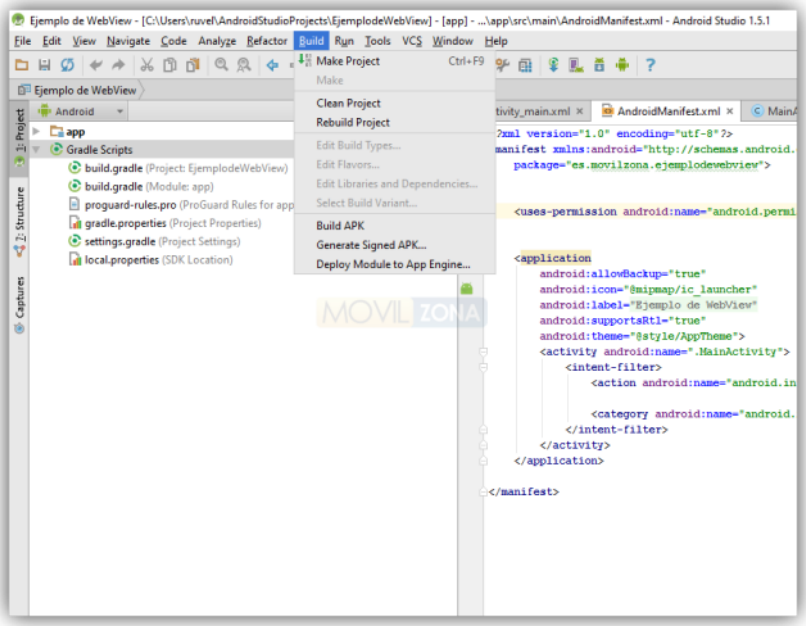

Figura 12.1.: Android studio-buid

Aquí podemos elegir dos opciones. Si ya tenemos una clave creada anteriormente la cargaremos desde el botón *Choose Existing* e introduciremos el correspondiente nombre, usuario y contraseña para poder utilizarla.

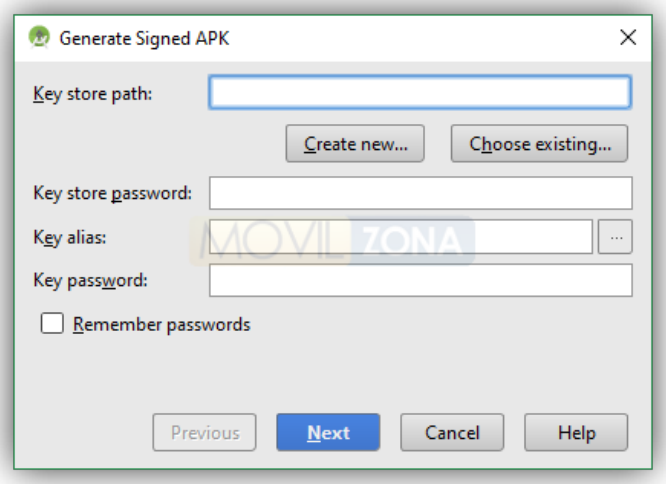

Figura 12.2.: Generar aplicación firmada - key

Si nunca hemos generado una clave o queremos crear una nueva por diversos motivos, pulsaremos sobre *Create New*. Se nos abrirá una nueva ventana como la siguiente:

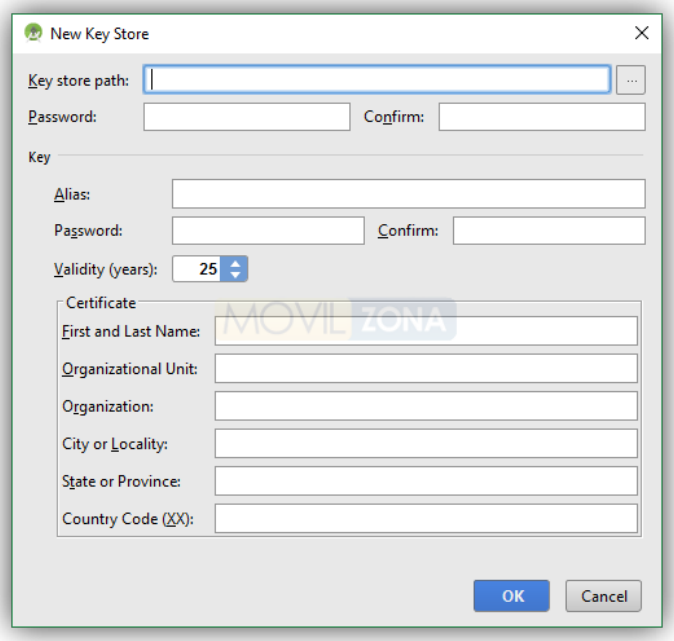

Figura 12.3.: Generar aplicación firmada - nuevo key store

En esta ventana debemos rellenar los siguientes apartados:

- Key Store Path: Ruta donde guardaremos la clave.
- Password: Contraseña 1 para nuestra clave.
- Alias: Nombre que daremos a nuestra clave.
- Password: Contraseña 2 para nuestra clave.
- Validity: Tiendo de validez de la clave (en años).
- First and last name: Nombre y apellidos.
- Organizational Unit: Nombre de nuestra empresa.
- Organization: Nuestra empresa (otra vez)
- City: Ciudad.
- State: Estado, país.
- Country Code: Código de nuestro país.

Aceptamos y Android Studio guardará el fichero de la clave en la ruta especificada. Debemos guardar a buen recaudo este archivo ya que sin él perderemos el control sobre nuestra aplicación y no podremos actualizarla más adelante. Una copia de seguridad en un USB y en la nube (cifrada, para evitar robos) es la mejor opción. Una vez hecho esto, Android Studio cargará automáticamente nuestra clave generada y nos permitirá seguir con la generación del apk.

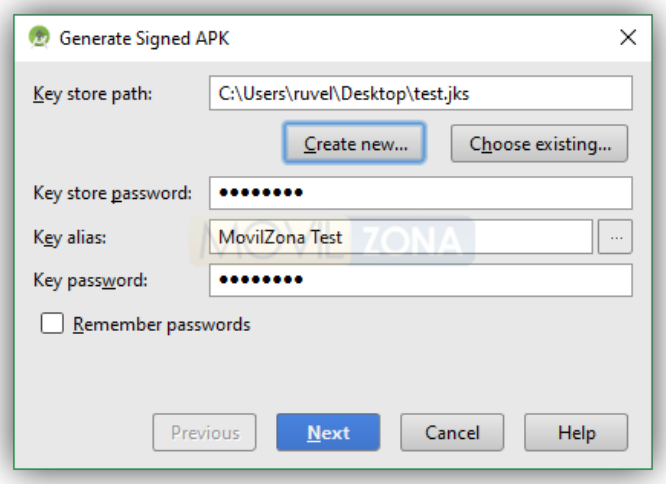

Figura 12.4.: Generar aplicación firmada- key creada

Pulsamos *Next* y en el siguiente paso nos preguntará la ruta donde guardará el APK y el tipo de compilación que va a ser (release para publicar o debug para probar y depurar).

Pulsamos sobre Finish y listo. Android Studio compilará nuestra aplicación y la guardará en la ruta especificada. Una vez finalice el proceso veremos un aviso en el IDE que no indica que todas las tareas se han realizado correctamente.

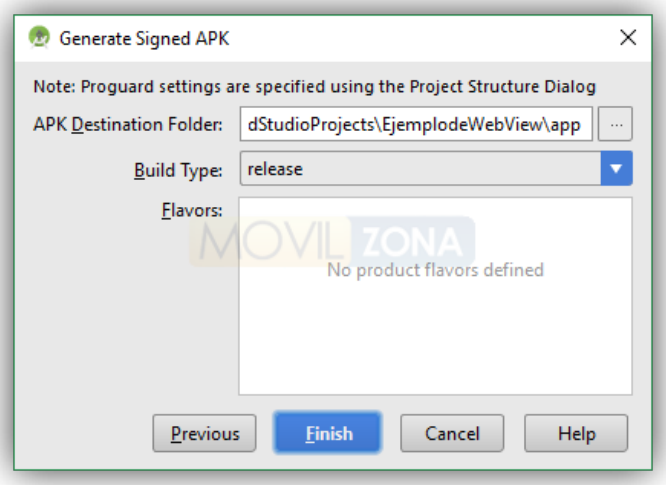

Figura 12.5.: Generar aplicacion firmada - finalizar

12.1.2. Compilación Eclipse

Para la compilación en eclipse utilizamos Maven. Este ya nos proporciona un WAR generado cuando ejecutamos sus comandos. Pulsamos con el botón derecho sobre el proyecto para elegir “Run As”. En el desplegable podremos comprobar como hay ya varios comandos pre-configurados por el plugin de Maven en Eclipse; es decir, que pulsando un botón ejecutaremos estos comandos sin tener que escribir una palabra en la línea de comandos.

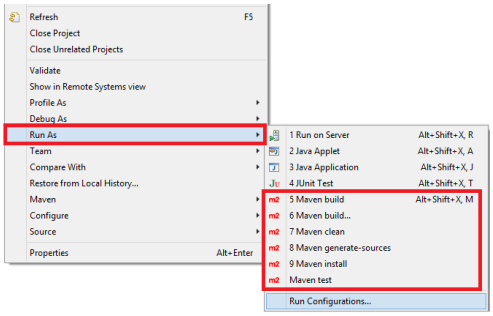

Figura 12.6.: Eclipse - opciones maven

12.1.2.1. POM

El archivo descriptivo que contendrá las opciones de compilación, dependencias, etc, es el “pom.xml”, el de resto de pestañas son asistentes para configurar el POM de una manera más sencilla, que aquí no entraremos pero échalas un vistazo que son útiles.

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.jarroba.ejemplo</groupId>
    <artifactId>nombreDeMiProyecto</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <packaging>jar</packaging>

    <name>nombreDeMiProyecto</name>
    <url>http://jarroba.com/</url>

    <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    </properties>

    <dependencies>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.12</version>
            <scope>test</scope>
        </dependency>

        <dependency>
            <groupId>com.google.code.gson</groupId>
            <artifactId>gson</artifactId>
            <version>2.3.1</version>
        </dependency>
    </dependencies>

    <build>
        <pluginManagement>
            <plugins>
                <plugin>
                    <groupId>org.apache.maven.plugins</groupId>
                    <artifactId>maven-compiler-plugin</artifactId>
                    <configuration>
                        <source>1.8</source>
                        <target>1.8</target>
                    </configuration>
                </plugin>
            </plugins>
        </pluginManagement>
    </build>

</project>
```

Figura 12.7.: Eclipse - pom.xml

Podemos describir su estructura básica con el siguiente listado:

- **build:** Se ocupa de declarar la estructura del proyecto, gestionar plugins, y configura los informes.
- **properties:** Propiedades y/o variables para maven.
- **pluginManagement:** Solo configura los plugins que son referenciados dentro de elementos de los plugins de los hijos (plugins).
- **plugins:** Los plugins nos aportan funcionalidades extra.
  - **groupId,artifactId, version**
  - **extensions,inherited, configuration**
  - **dependencies**
- **dependencies:** Son las dependencias (bibliotecas) que sean necesarias para la ejecución de la aplicación.

12.1.2.2. Run options

Por mediación del comando “maven install” construiremos un war completamente funcional y desplegable en tomcat.

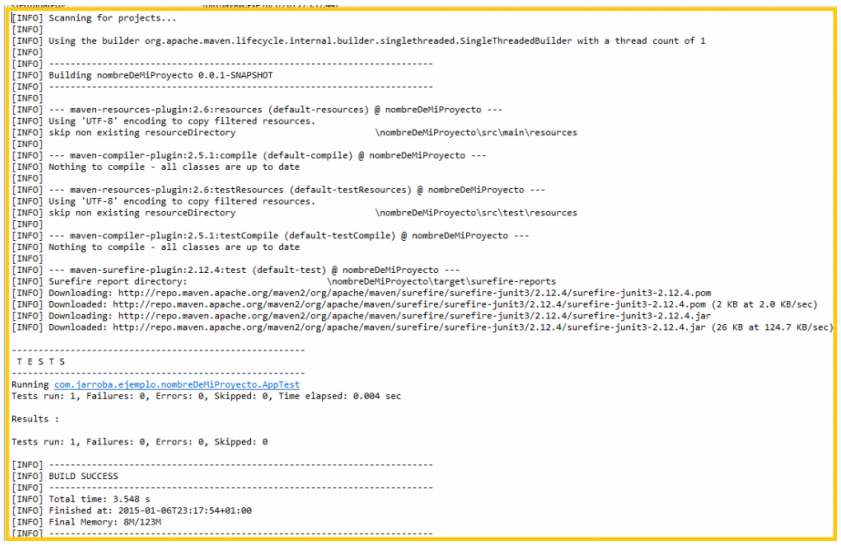

Figura 12.8.: Eclipse - salida de consola maven

Bibliografía

[lom(2017)] Lombok proyect, 2017. URL https://projectlombok.org.

[J.I.Cuerno(2018a)] J.I.Cuerno. Intraza20 android), 2018a. URL https://github.com/trinithy2000/intraza20.

[J.I.Cuerno(2018b)] J.I.Cuerno. Documentación), 2018b. URL https://github.com/trinithy2000/IntrazaDoc.

[J.I.Cuerno(2018c)] J.I.Cuerno. Erp sample groovy), 2018c. URL https://github.com/trinithy2000/erpSample.

[J.I.Cuerno(2018d)] J.I.Cuerno. Intrazaweb), 2018d. URL https://github.com/trinithy2000/IntrazaWeb.

[Martin(2017)] Robert C. Martin. Androidannotations, 2017. URL http://androidannotations.org/.

[P. Deemer and Vodde.(2013)] C. Larman P. Deemer, G. Benefield and B. Vodde. *Información Básica de Scrum*. Scrum Training Institute, 2013. URL http://www.goodagile.com/scrumprimer/scrumprimer_es.pdf.

[robles(2017)] Carlos robles. Fundamentos del desarrollo de aplicaciones para android, 2017. URL https://www.slideshare.net/SantiagoSolis1/fundamentos_del_desarrollo_...

[Villalobos(2016)] Ricardo Walter Marcelo Villalobos. *Fundamentos de programación Java*. Marcombo, 2016.

[wikipedia(2017a)] wikipedia. Android studio, 2017a. URL https://es.wikipedia.org/wiki/Android_Studio.

[wikipedia(2017b)] wikipedia. Eclipse (software), 2017b. URL https://es.wikipedia.org/wiki/Eclipse_(software).

[wikipedia(2017c)] wikipedia. Gradle, 2017c. URL https://es.wikipedia.org/wiki/Gradle.

[wikipedia(2017d)] wikipedia. Groovy (lenguaje de programación), 2017d. URL https://es.wikipedia.org/wiki/Groovy_(lenguaje_de_programacion).

[wikipedia(2017e)] wikipedia. Java (lenguaje de programación), 2017e. URL https://es.wikipedia.org/wiki/Java_(lenguaje_de_programacion).

[wikipedia(2017f)] wikipedia. Jax-rs, 2017f. URL https://es.wikipedia.org/wiki/JAX-RS.

[wikipedia(2017g)] wikipedia. Maven, 2017g. URL https://es.wikipedia.org/wiki/Maven.