Memoria del proyecto

Creación de la oficina virtual en Android para la empresa Electra Energía, S. A.

Rubén Pérez Ibáñez

M0.659 - Trabajo final de màster DADM aula 3

Máster Desarrollo de Aplicaciones para Dispositivos Móviles

Universitat Oberta de Catalunya

3 de enero de 2018

Proyecto dirigido por: Francesc D'Assís Giralt Queralt

CC BY NC SA

Resumen

Este documento incluye la memoria técnica del proyecto de la asignatura M0.659 - Trabajo final de máster DADM aula 3 del Máster Desarrollo de Aplicaciones para Dispositivos Móviles. El objetivo de este proyecto es desarrollar el prototipo de una aplicación para dispositivos que dispongan del sistema operativo Android de Google. Se pretende que esta aplicación permita a los usuarios de la aplicación acceder a la información asociada:

- Pólizas.
- Facturas.
- Datos personales.
- Formularios para la realización de modificaciones contractuales

Para ello se ha realizado un análisis de los requisitos y las funcionalidades demandadas, se han estudiado las diferentes tecnologías y herramientas adoptadas por la empresa para el desarrollo de aplicaciones y, posteriormente, se ha procedido con la implementación propiamente dicha de la API REST de comunicación y la aplicación. Para finalizar el desarrollo se han llevado a cabo una serie de pruebas para comprobar que el funcionamiento del producto final es satisfactorio.

Palabras clave

OpenERP, Android, Flask, Python , Java.

Índice general

Índice de figuras

Capítulo 1

Introducción

Índice

En el primer capítulo se documentan cual es el contexto, la motivación del alumno, los objetivos propuestos al inicio del proyecto, así como el entorno de trabajo y la estructura de toda la memoria.

1.1. Contexto y justificación del Trabajo

ELECTRA ENERGÍA, S.A.U.[3] es una empresa comercializadora de energía eléctrica nacida en el año 2000 para atender a los clientes de la zona de distribución de ELECTRA DEL MAESTRAZGO, que tras la liberalización del sector eléctrico carecían de una comercializadora de confianza. Electra Energía comercializa principalmente en el norte de la Provincia de Castellón (comarcas de Els Ports y Baix Maestrat), y noreste de la Provincia de Teruel (comarcas del Bajo Aragón, Maestrazgo y Matarraña), aunque ya se está expandiendo a zonas limítrofes a través de clientes interesados en mejorar sus condiciones contractuales y que, además, buscan la proximidad a la hora de

tratar con su empresa comercializadora.

Para adecuarse a la liberación del sector eléctrico, toda empresa del sector tiene como obligación adaptarse a los cambios que necesitan sus clientes, es decir, adaptarse a las nuevas tecnologías y ofertar a los usuarios todos aquellos mecanismos de comunicación que necesiten. Resultado de la modernización de la empresa es la creación de la Oficina Electrónica que consiste en crear una nueva manera de relacionarse los clientes con el departamento de Atención al Cliente. Con este avance se pretendió mejorar la calidad de los servicios ofertados y la accesibilidad a los mismos por parte de los clientes.

Analizando los resultados del GoogleAnalytics [1], hemos observado que el 30 por ciento de los usuarios acceden a la web a través de dispositivos móviles y que, además un 23 por ciento lo realizan con dispositivos móviles (figura 1.1). Por todo ello, el objetivo de este trabajo es realizar un análisis e implementación de una nueva Oficina Virtual adaptada para usuarios Android[4]. Para realizar este trabajo es necesario analizar los servicios más utilizados de la Oficina Virtual de la web, implementarlos en Android, añadir nuevos servicios y realizar una tarea de difusión entre los clientes de la empresa. Una vez implementada la aplicación se espera mejorar la comunicación entre la empresa y sus clientes, reducir costes asociados a envió de cartas o atención telefónica, mejorar los servicios al usuario y captar nuevos clientes.

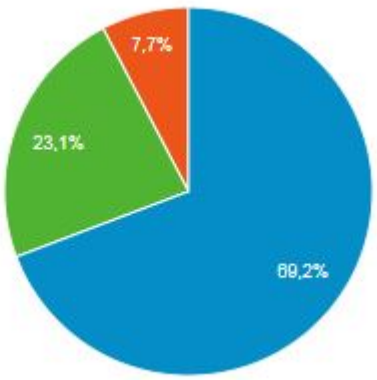


Figura 1.1: Porcentaje de dispositivos que visitan nuestro portal web. El color azul representa el acceso por ordenador, el color verde el acceso por móvil y el naranja con tablets

1.2. Objetivos del Trabajo

Para alcanzar el objetivo general planteado, se pretende lograr una serie de objetivos más específicos:

1. Contextualizar la empresa ELECTRA ENERGÍA, S.A.U.
2. Realizar un análisis de la Oficina Virtual actual de Electra Energía.
3. Realizar un breve estudio entre los usuarios para buscar nuevas funcionalidades.
4. Crear una API donde comunicarse las Apps.
5. Crear una App en Android con las siguientes funciones:
   - Descargar y visualizar los contratos asociados al usuario.
   - Ver la evolución del consumo.
   - Tener acceso al histórico de facturación.
   - Ponerse en contacto con la empresa.
   - Conocer donde está situado la sede más cercana.
   - Interfaz de gestión de usuario.
   - Sistema de configuración de alertas para avisar de nuevas facturas.
6. Realizar una campaña informativa entre los clientes.

1.3. Enfoque y método seguido

Los dispositivos móviles representan una nueva oportunidad de acceso a la información por parte de los usuarios y, en el caso de Electra Energía, supone un nuevo mecanismo de comunicación entre la empresa y los usuarios. Sin embargo, la diversidad de características de hardware y software que presentan estos dispositivos complica el enfoque que debes aplicar cuando deseas llevar a cabo un desarrollo de software para ellos. (Tecnología y desarrollo en dispositivos móviles)

Esta diversidad genera la necesidad de realizar un estudio previo para establecer de forma resumida el ámbito en el que nos vamos a mover. Para empezar, en este momento existes dos tipos de tecnologías para dispositivos móviles muy diferencias, las aplicaciones web y las aplicaciones nativas. El primer punto de nuestro enfoque será definir cada tecnología, analizar sus ventajas y escoger nuestra mejor opción.

Una aplicación web es, básicamente, un sitio web específicamente optimizado

para un dispositivo móvil. Las características que definen una aplicación web son las siguientes: la interfaz de usuario se construye con tecnologías web estándar, está disponible en una URL2 (pública, privada o protegida por una contraseña) y está optimizada para los dispositivos móviles. Una aplicación web no está instalada en el dispositivo móvil. Por el contrario, las aplicaciones nativas están instaladas en el dispositivo móvil, tienen acceso al hardware (altavoces, acelerómetro, cámara, etc.) y están escritas en algún lenguaje de programación compilado (como, por ejemplo, el Objective-C).

Una vez definidas ambas posibilidades, hemos analizado sus ventajas e inconvenientes. Es cierto que las aplicaciones web tienen un ciclo de desarrollo más rápido, la aplicación puede funcionar en cualquier dispositivo que tenga un navegador web y podríamos aprovechar parte del código e implementación actual de la oficina virtual, ahora bien, no podríamos acceder a todas las características del código y sería difícil conseguir algunos efectos. [11]

En otro sentido, tenemos las aplicaciones nativas que, sin olvidarnos que estos desarrollos nos permiten explotar al máximo todas las características de los dispositivos nos encontramos con una serie de desventajas como son:

- La aplicación solo funcionará en la plataforma escogida.
- Hay que desarrollarla usando el lenguaje de programación establecido para la plataforma.
- Es más complicado distribuir parches o actualizaciones que solucionen errores.
- El ciclo de desarrollo es más lento.

Después de una valoración interna sobre las necesidades de nuestra oficina virtual, hemos decidido optar por implementar aplicaciones nativas para Apple[5] y Android y mantener una oficina virtual responsive accesible por navegador para el resto de dispositivos. Los motivos de esta elección han sido ideas que han aportado al proyecto desde nuestro departamento comercial donde sería necesario utilizar todas las características de los dispositivos (permitir pagos con la tecnología NFC, enviar fotos de los contadores con las lecturas del mes o alertar de alguna incidencia en la red, hacer eventos de geoposicionamento, etc.) que no son fáciles de implementar en el desarrollo web. Posteriormente, hemos analizado la Api de comunicación con las aplicaciones, para ello utilizaremos la misma API que hemos implementado en la oficina virtual y añadiremos nuevas funcionalidades. La API se implementara en Flask[9], un framework de desarrollo web de código abierto, escrito en Python. En el punto 7.1 explicaremos como funciona y que funciones hemos implementado.

La elección de este framework es sencilla, el ERP de gestión de la empresa es OpenERP, un sistema de software libre liberado bajo la GPL e implementado en Python. Con los conocimientos actuales que tenemos sobre el OpenErp y, además, con el ORM integrado en Flask nos permite aprovechar gran parte del trabajo ya implementado en OpenErp para acceder a la base de datos de PostgreSQL y MongoDB donde se almacenan los datos.

1.4. Planificación del Trabajo

En este apartado vamos a comentar las distintas tareas que se han llevado a cabo para la realización del proyecto, así como un diagrama de Gantt donde se puede apreciar tanto la duración temporal de cada una de ellas, como las distintas precedencias que existen entre las tareas.

1.4.1. Tareas definidas en el Diagrama

- Definición del Proyecto
  - Análisis de Requisitos Electra Energía, S. A. U.
  - Análisis del Sistema GisceErp.
  - Reunión con el departamento Comercial.
  - Análisis de la futura Api de comunicación.
  - Elaboración del documento de especificaciones.
- Diseño
  - Análisis de la Oficina Virtual en Web
  - Elaboración de un mockup .
  - Elaboración de un prototipo.
  - Diseño de la API de comunicación con el servidor.
- Implementación
  - Implementación de la API de comunicación con el Erp
  - Testeo de la API de comunicación desde Android
  - Creación del login de acceso
  - Testeo del login
  - Acceso del histórico de consumo
  - Testeo histórico de consumo
  - Acceso a un contrato
  - Testeo acceso al contrato

- Acceso a una factura
- Testeo del acceso a la factura.
- Generación de gráfica con el consumo.
- Implementación del buscador de la delegación más cercana
- Implementación de los mecanismos de contacto
- Testeo de la aplicación con usuarios.
- Resolución de problemas y mejoras.

- Entrega Final
  - Elaboración del documento final.
  - Resolución de observaciones indicadas en la implementación

- Tribunal

1.4.2. Diagrama de Gantt

El siguiente diagrama de Gantt (figura 1.2) permite visualizar las duraciones planificadas, fecha de inicio y finalización, y estructura de precedencias al inicio del desarrollo del proyecto. El horario de trabajo contemplado en el diagrama es: lunes a viernes de 20:00 a 21:00 y fines de semana y festivos de 10:00 a 13:00. Como el proyecto es desarrollado por una única persona no se puede ganar tiempo calculando las dependencias entre tareas y ejecutándolas en paralelo, por lo que se realizarán secuencialmente de izquierda a derecha como se observa en el diagrama, excepto las tareas de investigación de los recursos iniciales y las tareas de la documentación básica que se realizarán paralelamente.

Para terminar, esta planificación estaba bien definida pero el desarrollo real del proyecto se ha desviado por no contar con algunos factores que lo han retrasado. Por una parte factores como enfermedad, viajes, etc son factores típicos que atrasaron ligeramente el proyecto.

1.5. Presupuesto del proyecto

El presupuesto profesional de este proyecto depende principalmente del sueldo de los trabajadores, en este caso, el mio. Se considera que el sueldo de un trabajador es de unos 20 euros la hora.

Los gastos de personal se han calculado teniendo en cuenta los siguientes aspectos:

- El trabajador ha dedicado 3 horas por día al proyecto

- No se contabilizan los días festivos
- La duración del proyecto ha sido de unos 106 días

Para realizar el proyecto adecuadamente se necesita el sistema operativo Linux para poder utilizar todas las herramientas necesarias descritas en el proyecto. También ha sido necesario adquirir un ordenador portátil con un coste de 1000 euros y un teléfono Samsung con un coste de 150 euros. El resto de herramientas que se han utilizado son libres o bien gratuitas para el aprendizaje.

En la siguiente tabla se puede observar el coste total del proyecto:

Cuadro 1.1: Requisito de datos - Factura

Desarrollo	318 horas x 1 persona 20 euros la hora = 6360 euros
Formación	20 horas x 1 persona 20 euros la hora = 400 euros
Equipo:	1 ordenador (1200 euros) + 1 dispositivo móvil (150 euros) =1350 euros
Total:	8110 euros

Por lo tanto, el presupuesto total del proyecto será de unos 6750 euros

Como conclusión de la propuesta técnica, se ha realizado un estudio de la viabilidad para determinar si la construcción del proyecto va a ser factible o no.

Analizando las posibles restricciones que pueden condicionar la viabilidad del sistema, se ha visto que:

- La justificación económica analizada en el apartado 1.6.
- El riesgo técnico es bajo
- No existe ningún problema de tipo legal

Todo esto lleva a la conclusión de que la creación de este sistema es totalmente viable.

1.6. Justificación Económica

. La viabilidad económica de este proyecto se basa en los beneficios que generan a la empresa. Aparte del atractivo que puede suponer para los nuevos clientes, hemos realizado una estimación económica que supone tanto la atención al cliente como el coste del envió de una factura.

- El coste del envió de una carta con la factura se estima por 0.72 euros al mes.

- El coste de un cambio de cuenta por teléfono supone 3 euros de tiempo de atención al cliente de un trabajador.

Por lo tanto, si de los 20.000 clientes de la comercializadora accedieran a la aplicación 1500 usuarios nos podríamos ahorrar aproximadamente:

- El envió de 2000 facturas mensuales en un año, que supondría un ahorro de 12960 euros
- La atención al cliente de unos 20 cambios de cuenta mensuales que supondrían un ahorro de 720 euros anuales

Por lo tanto, en un año podríamos recuperar la inversión económica del proyecto.

1.7. Breve sumario de productos obtenidos

Entre los productos obtenidos después de realizar el proyecto serán

- API de comunicación con el servidor desarrollada en Flask
- Aplicación java para distribuir en el principal mercado de Android, Google Play.

1.8. Breve descripción de los otros capítulos de la memoria

Comenzamos el proyecto exponiendo las conclusiones que hemos realizado en las investigaciones bibliográficas iniciales sobre la situación actual de la empresa. En el primer apartado, presentado la empresa Electra Energía, S.A.U. y el software de gestión empresarial que utilizan en este momento.

El segundo apartado presentamos una pequeña recopilación de las diferentes opciones de configuración para realizar la API de comunicación entre nuestra aplicación con el servidor ERP. En este mismo apartado analizaremos la opción escogida, Flask sus beneficios y inconvenientes junto con la configuración escogida.

En el tercer apartado, analizamos las diferentes tecnologías existentes en el mercado para la creación de aplicaciones en dispositivos móviles. Presentamos nuestra opción escogidas y analizamos un poco el funcionamiento de Android

El cuarto apartado se expondrá el diseño escogido para la aplicación. Mostraremos las diferentes vistas y analizaremos sus funcionalidades

En el quinto apartado vamos a presentar las herramientas utilizadas para realizar este proyecto y el porqué de esta elección.

El sexto apartado contiene todo lo relacionado con la implementación de la aplicación. Empezaremos explicando como hemos implementado la API de comunicación entre el servidor y el cliente para continuar explicando la estructura de la aplicación

Finalmente se expondrán la conclusión del proyecto en el último capitulo

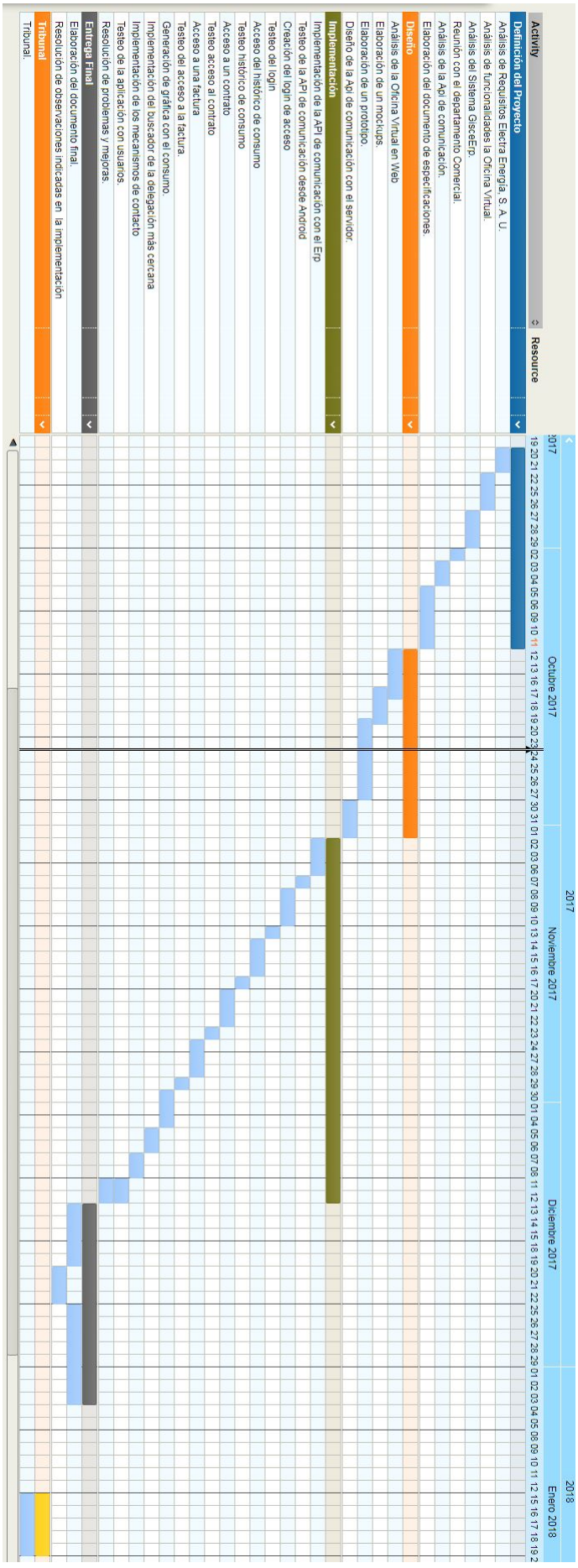


Figura 1.2: Diagrama Gant

CAPÍTULO 2

Análisis de Electra Energía, S.A.U. y OpenERP

Índice

El capítulo está organizado de la siguiente manera. En la sección 2.1 se introduce un poco a la historia y evolución de la empresa. Explicamos su finalidad y sus objetivos con la idea de poder entender mejor la finalidad de generar una aplicación móvil. En el siguiente punto, la sección 2.2 analizamos que es y como esta construido el ERP[1] de gestión de la empresa.

[1]Los sistemas de planificación de recursos empresariales ('ERP', por sus siglas en inglés, enterprise resource planning) son los sistemas de información gerenciales que integran y manejan muchos de los negocios asociados con las operaciones de producción y de los aspectos de distribución de una compañía en la producción de bienes o servicios.[13]

2.1. Electra Energia

Electra Energía, S.A.U. es una empresa del sector eléctrico que, junto a las sociedades Hidroeléctrica del Guadalope, S.A. y Maestrazgo Distribución, S.L. forman parte y crean el grupo empresarial Electra del Maestrazgo, S.A.

Figura 2.1: Logo Electra Energía

La sociedad conocida como Electra Energía, nació el 8 de noviembre del año 2000 en Castellón de la Plana. Su nacimiento fue provocado por la imposición legislativa que obligaba a separar la comercialización y la distribución de energía. Electra del Maestrazgo era la empresa que asumía esa doble función y, a partir de esa fecha, Electra del Maestrazgo, pasó a encargarse únicamente de la distribución energética y Electra Energía la de su comercialización dentro del grupo.

A día de hoy, Electra Energía, comercializa en la zona norte de la provincia de Castellón abarcando las comarcas de Els Ports y Baix Maestrat y también lo hace en las comarcas del Bajo Aragón, Maestrazgo y Matarraña en el noroeste de la provincia de Teruel. Esta es la zona en la que distribuye energía Maestrazgo Distribución; pero, Electra Energía, gracias al mercado libre energético, tiene ya clientes fuera de esa zona y pretende, ampliando los medios que tenia hasta ahora, aumentarlos.

Para lograrlo, su Dirección, sabiendo que la suya es una pequeña empresa del sector energético compuesta por un grupo no muy amplio de trabajadores, debe dar un trato personalizado y cercano a sus clientes actuales y futuros, algo difícil de encontrar en el sector eléctrico. Siendo una comercializadora de confianza que es lo que necesitan sus clientes. Una empresa que les cobre un precio justo por la energía que les suministra. Unos clientes a los que se les explique al detalle todos los conceptos que pagan en su factura de la luz si no los entiende y, sobre todo, que sientan que reciben una atención personalizada tal y como se merecen. Todo el personal de Electra Energía, desde el primero hasta el último del escalafón, tiene claro que uno de los valores principales de la empresa en la que trabajan es que, el cliente, ante todo, es una persona y tiene que recibir el trato como tal.

2.2. OpenERP

Hace 6 meses, Electra Energía, S.A.U. decidió migrar el sistema de gestión empresarial hacia una solución basada en el software libre llamado OpenErp. La empresa contratada para llevar a cabo esta tarea fue GISCE-TI SL, una compañía de Servicios TI, especializada en proporcionar soluciones para la Gestión Operativa y la Administración Pública bajo un concepto de eficacia mediante un incremento de ingresos, una reducción de costes y una mejora de imagen de sus clientes.

Figura 2.2: Arquitectura

Gracias al amplio conocimiento de las actividades y procesos ligados a la actividad de comercialización eléctrica, Electra Energía S.A.U. en unión con GISCE-TI SL ha conseguido elaborar una solución para la gestión de todos los procesos internos de la empresa a través de la plataforma OpenERP.

El programa resultado, conocido como Electra-ERP, esta basado en la tecnología de cliente-servidor utilizando como base de datos PostgreSQL Y MongoDB. Dentro de la aplicación nos encontramos con los siguientes módulos:

- Módulo de facturación que incluye gestión de contratos, histórico de modificaciones contractuales, facturación
- Módulo de sincronización COMERDIS, entre comercializadora y distribuidora.

- Módulo de switching, gestión de procesos ATR entre comercializadora y distribuidoras externas.
- Módulo switching para gestión de facturas de distribuidores en formatos F1 y lecturas Q1, gestión de pagos de facturas de proveedor.
- Módulo Administración pública, para la obtención de informes para las diferentes administraciones (Modelo 347, 159, ITC-606/2011,...)
- Integración con la oficina virtual a través de XMLRPC, ópcional
- Módulo de gestión de cobros, (remesas, devoluciones...)
- Módulo power-email para envío de información y facturas por correo electrónico.

Electra-ERP cubre la totalidad de los requerimientos necesarios para la correcta gestión de un sistema de Planificación de Recursos Empresariales orientado al sector eléctrico ofreciendo una herramienta completa al servicio de Electra Energía S.A.U utilizando la siguiente estructura que comentaremos en el punto 2.2.1

Server ERP (Distribución y Comercialización):

- Mínimo 16 Gb de RAM
- Mínimo 16 procesadores arquitectura 64bits
- Almacenamiento 200GB
- I/O Disco bajo

Server Database NoSQL (Mongodb + Redis + Search index):

- Mínimo 32 Gb de RAM
- Mínimo 4 procesadores arquitectura 64bits
- Almacenamiento 1.5TB
- I/O Disco alto puntual

Server Database SQL (PostgreSQL)

- Mínimo 32 Gb de RAM
- Mínimo 8 procesadores arquitectura 64bits
- Almacenamiento 500GB
- I/O Disco alto

Server GIS:

- Mínimo 8 Gb de RAM
- Mínimo 12 procesadores arquitectura 64bits
- Almacenamiento 500GB
- I/O Disco alto

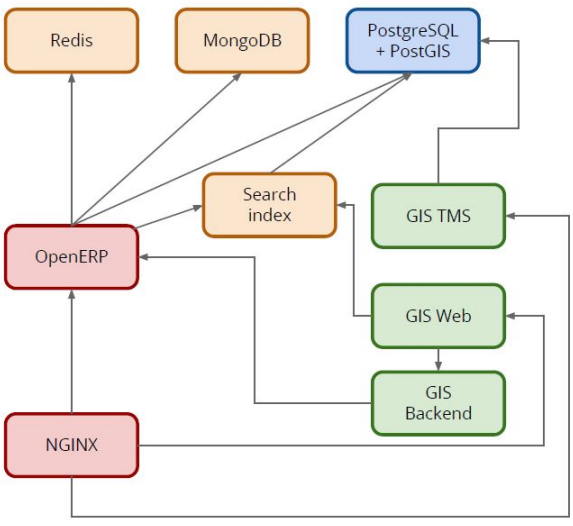


Figura 2.3: Estructura del OpenErp

2.2.1. Arquitectura Electra-ERP

OpenERP es un ERP libre, gratuito y multiplataforma. Su arquitectura web permite adaptarlo fácilmente e integrarlo con otros productos, como Business Intelligence, para optimizar la toma de decisiones en su empresa.

Sus principales características son:

1. **Libertad**: sin dependencia del proveedor.

2. **Código abierto:** Al ser software libre, podremos disponer del código para realizar cualquier mejora sobre los módulos ya existentes, o crear uno nuevo adaptado a sus necesidades.

3. **Conectividad:** visualización de informes en formato estándar PDF, importación/exportación con MS Office Excel o CSV, conectores ya existentes con Magento y Prestashop, y otras muchas plataformas, con la posibilidad de conexión con casi cualquier software utilizando webservices.

4. **Flexibilidad:** OpenERP es una solución modular. Puede comenzar por ejemplo utilizando solamente el módulo de CRM o Financiero, e ir integrando más módulos posteriormente, todos ellos compartiendo el mismo flujo de trabajo y la misma base de datos.

5. **Gratuito:** OpenERP es un producto que no tiene coste de licencias. No tiene que pagar dinero por usarlo en más puestos de trabajo o renovar las costosas licencias anualmente. Ese dinero puede aprovecharlo para otras mejoras informáticas o en otros departamentos. Solamente tiene el conste de la adaptación a la implementación y el mantenimiento.

6. **Multiplataforma:** la interfaz web de OpenERP le permite acceder desde cualquier ordenador independientemente del sistema operativo (GNU/Linux, Mac OS X o Windows), e incluso tablets y smartphones con Android o iOS. La versión “mobile” simplifica las vistas y lo hace más agradable y sencillo de manejar desde pantallas de tamaño reducido.

7. **OpenObject:** el framework de OpenERP permite un desarrollo rápido de funcionalidades o conectividad con otras plataformas. La extensa documentación facilita la integración y mejora del programa de manera muy ágil.

8. **Variedad:** OpenERP es un sistema en crecimiento que cuenta actualmente con centenares de módulos liberados que se pueden combinar y/o servir como base para construir casi cualquier tipo de solución vertical.

9. **Integración**: disfrute de módulos de gestión y aplicaciones web propias para construir portales web, tiendas online, foros o eventos. Todo dentro de plataforma única y un control unificado.

10. **PostgreSQL**, el motor de base de datos de OpenERP es un potente desarrollo libre dirigido por una comunidad de desarrolladores y organizaciones comerciales que cuenta con clientes como Skype, Sony Online, Departamento de Trabajo de U.S.A, IMDb (1), etc...

11. **Fácil migración**: la herramienta oficial importa y exporta datos maestros en formato .csv para que le resulte más sencillo seguir trabajando con los datos de su aplicación actual.

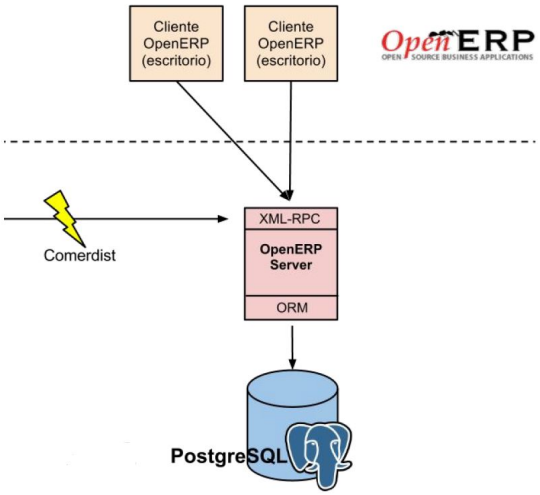


Figura 2.4: Arquitectura

Capítulo 3

Investigación sobre el desarrollo de Api de comunicación con el servidor.

Índice

Para llevar a cabo la oficina virtual, es necesario desarrollar un sistema de comunicación con el servidor con el que la aplicación se comunicará para recibir y enviar datos. La forma más habitual de comunicarse con el servidor es a través de una comunicación HTTP o HTTPS utilizando uno de los dos servicios más extendidos: SOAP y REST.

Este apartado hará una breve descripción de estas técnicas y explicara la opción escogida.

3.1. SOAP

Los servicios SOAP o mejor conocimos simplemente como Web Services, son servicios que basan su comunicación bajo el protocolo SOAP (Simple Object Access Protocol) el cual este definido por Wikipedia como "protocolo estándar que define cómo dos objetos en diferentes procesos pueden comunicarse por medio de intercambio de datos XML"[18]. Por lo tanto, queda claro

que la comunicación se realiza mediante XML. Los servicios SOAP funcionan por lo general por el protocolo HTTP, aunque pueden utilizar otros protocolos como el Ftp.

Normalmente, estos servicios los utilizaremos para la comunicación de Server to Server o Partner to Partner pues es un protocolo mucho más robusto que permite agregar metadatos mediante los atributos del propio sistema XML (cosa que JSON no tiene) y definir espacios de nombre con el objetivo de evitar la ambigüedad. Por lo mismo, SOAP es un formato más pesado, tanto en tamaño como en procesamiento, pues los XML tiene que ser parseado a un árbol DOM, resolver espacios de nombre (namespaces) antes de poder empezar a procesar el documento. Los XML además tienen métodos de validación muy potentes y ampliamente utilizados, a diferencia de JSON, el cual, si tiene forma de validar, pero no son tan potente y su utilización es menor.

Figura 3.1: Ejemplo comunicación SOAP

3.2. REST

REST es una tecnología flexible que transporta datos a través de los diversos medios que le proporciona el protocolo HTTP, como son:

- **GET**: Se utiliza para consultar, leer y en definitiva acceder a un recurso
- **POST**: Envía datos para crear un recurso. Como en cualquier petición POST, los datos deben ir incluidos en el cuerpo de la petición.
- **PUT**: Utilizado para editar un recurso. Al igual que el POST, los datos deben ir en el cuerpo de la petición.
- **DELETE**: Es la opción para eliminar un recurso

- **PATCH**: Se utiliza para modificar parcialmente un recurso, aunque se utiliza en muy pocas ocasiones. Normalmente se utiliza simplemente PUT

Y además, incorpora los códigos de respuesta nativos como:

- **200 OK Solicitud aceptada**; la respuesta contiene el resultado.
- **201 CREATED** Las operaciones PUT o POST devuelven este código de respuesta e indica que se ha creado un recurso de forma satisfactoria.
- **204 NO CONTENT** Indica que se ha aceptado la solicitud, pero no había datos para devolver.
- **400 BAD REQUEST** La solicitud no fue válida.
- **401 UNAUTHORIZED** El servidor de aplicaciones devuelve este código de respuesta cuando está habilitada la seguridad y faltaba la información de autorización en la solicitud.
- **403 FORBIDDEN** Indica que el cliente ha intentado acceder a un recurso al que no tiene acceso.
- **404 NOT FOUND** Indica que el recurso de destino no existe.
- **406 NOT ACCEPTABLE** El recurso de destino no admite el formato de datos solicitado en la cabecera de Accept o el parámetro accept.
- **409 CONFLICT** Indica que se ha detectado un cambio conflictivo durante un intento de modificación de un recurso.
- **415 UNSUPPORTED MEDIA TYPE** El recurso de destino no admite el formato de datos del cuerpo de la solicitud especificado en la cabecera de Content-Type.
- **500 INTERNAL SERVER ERROR** Se ha producido un error interno en el servidor.

El cliente inicia siempre una solicitud con la finalidad de solicitar al servidor un determinado recurso utilizando su propio lenguaje. Tanto el cliente como el servidor se comunican en un formato de intercambio de información tan flexible que permite transmitir prácticamente cualquier tipo de datos, ya que el tipo de datos está definido por el Header Content-Type, lo que nos permite mandar, XML, JSON, Binarios (imágenes, documentos), Text, etc. que contrasta con SOAP que solo permite la transmisión de datos en formato XML. A pesar de la gran variedad de tipos de datos que podemos mandar con REST, la gran mayoría transmite en JSON por un motivo muy importante, JSON

es interpretado de forma natural por JavaScript, lo que ha hecho que frameworks como Angular y React se aprovechen al máximo, pues pueden enviar peticiones directas al servidor por medio de AJAX y obtener los datos de una forma nativa [20].

Lo importante es que el cliente no recibe HTML para renderizar, sino simplemente los datos que se han generado como respuesta. Es decir, el servidor no escribe HTML, sino únicamente genera los datos para enviarlos al cliente.

Otro punto importante es la independencia con la tecnología escogida de programación o base de datos escogida. Podrás tener un servidor que trabaja con PHP, Java, Python, NodeJS o lo que prefieras, o te imponga el proyecto. Tanto el lenguaje como la base de datos son importantes, porque nos sirven para procesar la solicitud y generar la respuesta, pero no importa cómo lo hagas en el servidor. Simplemente que la respuesta la entregues en ese lenguaje de intercambio de información que estés usando, generalmente JSON.

Figura 3.2: Ejemplo comunicación REST

3.3. Conclusión

Gracias al estudio previo que hemos realizados, hemos podido analizar y diferenciar los dos servicios con el objetivo de escoger la mejor opción.

Por un lado tenemos en el servicio SOAP, un sistema de comunicación mediante archivos XML basado en la robustez y experiencia. Normalmente se suele utilizar a través protocolo HTTP, sin embargo, SOAP no está limitado a este protocolo, si no que puede ser enviado por FTP, POP3, TCP, Colas de mensajería (JMS, MQ, etc).

Ahora bien, entre sus principales problemas nos encontramos:

- Su complejidad.
- No funciona muy bien con dispositivos móviles y aplicaciones RIA basadas en javascript. [1]
- Es ineficiente desde el punto de vista del ancho de banda.El propio uso de este servicio implica un consumo mayor del ancho de banda justamente por la estructura que requieren estos paquetes (Cabecera, tipos, etc).
- No aprovecha la infraestructura web, como caches, negociación de formatos, negociación de idioma, seguridad, sistema de concurrencia optimista, etc.
- Problemas con el firewall. SOAP usa el protocolo HTTP como un mero medio de transporte. No aprovecha sus características, y subvierte la semántica de este último. Esto hace que las llamadas SOAP sean "opacas" desde el punto de vista de la infraestructura web, como caches o firewalls.

Por lo tanto, no seria la mejor opción para nuestro proyecto ya que nuestra aplicación móvil necesitaría una solución más sencilla, optimizada y adaptable.

Desde punto de vista del backend, necesitamos proporcionar al frontend una manera sencilla de obtener los datos que requieren las interfaces de usuario, como serian una app móvil Android, iOS o una interfaz web hecha en React[2].

[1] Una rich Internet application (RIA), aplicación de Internet enriquecida o aplicación rica de internet (ARI), es una aplicación web que tiene la mayoría de las características de las aplicaciones de escritorio tradicionales. Estas aplicaciones utilizan un navegador web estandarizado para ejecutarse y por medio de complementos o mediante una máquina virtual se agregan las características adicionales.

[2] React (también llamada React.js o ReactJS) es una biblioteca Javascript de código abierto para crear interfaces de usuario con el objetivo de animar al desarrollo de aplicaciones en una sola página. Es mantenido por Facebook, Instagram y una comunidad de desarrolladores independientes y compañías. `https://reactjs.org/`

Por lo tanto, la mejor solución para el proyecto seria implementar una API REST consistente, donde podríamos alimentar las distintas interfaces, y además, proporcionar nuestra fuente de servicios para el consumo por parte de terceros.

Buscando una definición sencilla, REST es cualquier interfaz entre sistemas que use HTTP para obtener datos o generar operaciones sobre esos datos en todos los formatos posibles, como XML y JSON. Y seria la alternativa perfecta al SOAP, que disponen de una gran capacidad pero también mucha complejidad.

Figura 3.3: Solución final

CAPÍTULO 4

Investigación sobre el desarrollo de aplicaciones para dispositivos móviles

Índice

Como hemos comentado ya en la sección 1.3 existe una gran diversidad de dispositivos móviles, cada uno con características de hardware y software diferentes. Esta diversidad genera la necesidad de realizar un estudio previo para elegir aquella opción que se adecue más a nuestras necesidades. Para empezar, en este momento existes dos tipos de tecnologías para dispositivos móviles muy diferencias, las aplicaciones web y las aplicaciones nativas.

Una aplicación web es, básicamente, un sitio web específicamente optimizado para un dispositivo móvil. Las características que definen una aplicación web son las siguientes: la interfaz de usuario se construye con tecnologías web estándar, está disponible en una URL2 (pública, privada o protegida por una contraseña) y está optimizada para los dispositivos móviles. Una aplicación web no está instalada en el dispositivo móvil. Por el contrario, las aplicaciones nativas están instaladas en el dispositivo móvil, tienen acceso al hardware (altavoces, acelerómetro, cámara, etc.) y están escritas en algún lenguaje de programación compilado (como, por ejemplo, el Objective-C).

Una vez definidas ambas posibilidades, hemos analizado los siguientes posibilidades:

- Phonegap 4.1
- Ionic 4.2
- Android 4.3
- Apple 4.3.2

4.1. Phonegap

PhoneGap permite desarrollar aplicaciones para Android mediante tecnologías web como HTML, CSS y JavaScript, y puede convertir esas aplicaciones web en aplicaciones nativas Android. De hecho, PhoneGap soporta múltiples plataformas (como Android, iPhone, Palm, Windows Mobile y Symbian), así que se puede usar el mismo código fuente para crear aplicaciones para múltiples plataformas. Pese a que se venden como "herramientas de tecnología web", lo que ofrecen PhoneGap u otros frameworks similares como Titanium es acceso al hardware de la máquina (se pueden hacer aplicaciones en HTML y JavaScript que usen la cámara, la brújula o el acelerómetro).

Por lo tanto, PhoneGap es una solución de código abierto diseñada para dar acceso JavaScript a los desarrolladores web a características populares de los dispositivos móviles como la cámara, el GPS, el acelerómetro o las bases de datos SQLite locales sin necesidad de tener que escribir aplicaciones completas. El objetivo es hacer más fácil el desarrollo de aplicaciones móviles.

Para conseguir esto, el framework PhoneGap actúa como un puente entre las aplicaciones web y los dispositivos móviles. Permite a los desarrolladores envolver aplicaciones web dentro de una aplicación nativa, lo que hace el desarrollo más fácil para aquellos que no están familiarizados con Objective-C y Cocoa.

Ahora bien, las desventajas que yo he encontrado al utilizar esta herramienta han sido:

- Dado que PhoneGap está diseñado para soportar diferentes plataformas, no contará con las características nuevas de cada plataforma tan pronto como estas se publiquen.
- Para acceder a características no disponibles desde el framework (PhoneGap) se deben realizar llamados a implementaciones nativas. Esto tiene

dos implicaciones; la primera es que la aplicación deja de ser multiplataforma debido a que se debe realizar una implementación nativa por funcionalidad no disponible para cada plataforma. La segunda es que el medio definido por PhoneGap para intercambiar datos con la plataforma nativa es JSON, lo que incrementa de forma considerable el parsing de este tipo de estructuras deteriorando sustancialmente el desempeño de la aplicación.

- Notas una diferencia entre una aplicación en Phonegap y una aplicación Nativa

4.2. Ionic

Ionic es una herramienta, gratuita y open source, para el desarrollo de aplicaciones híbridas basadas en HTML5, CSS y JS. Está construido con Sass y optimizado con AngularJS.

Sus principales características son:

- **Alto rendimiento:** Ionic está construido para ser rápido gracias a la mínima manipulación del DOM, con cero jQuery y con aceleraciones de transiciones por hardware.
- **AngularJS** Ionic utiliza AngularJS con el fin de crear un marco más adecuado para desarrollar aplicaciones ricas y robustas. Ionic no sólo se ve bien, sino que su arquitectura central es robusta y seria para el desarrollo de aplicaciones. Trabaja perfectamente con AngularJS.
- **Centro nativo** Ionic se inspira en las SDK de desarrollo móviles nativos más populares, por lo que es fácil de entender para cualquier persona que ha construido una aplicación nativa para iOS o Android. Lo interesante, como sabéis, es que desarrollas una vez, y compilas para varios.
- **Bonito diseño** Limpio, sencillo y funcional. Ionic ha sido diseñado para poder trabajar con todos los dispositivos móviles actuales. Con muchos componentes usados en móviles, tipografía, elementos interactivos, etc.
- **Un potente CLI** Con un sólo comando podrás crear, construir, probar y compilar tus aplicaciones en cualquier plataforma

4.3. Programación Nativa

En este momento, El 99.6 % del mercado móvil le pertenece a Android y iOS [16] y, por lo tanto, cuando hablamos de programación nativa tenemos que conocer y analizar ambos sistemas para tomar una decisión.

4.3.1. Android y la programción nativa

Android es una solución completa de software para dispositivos móviles. Incluye toda una pila de aplicaciones, como el sistema operativo, el middleware y las aplicaciones clave. También incluye herramientas para desarrollar en la plataforma, principalmente con el lenguaje de programación Java. Todo ello bajo licencia de código libre Apache, lo cual abre muchas posibilidades.

La pila de software del sistema operativo Android consiste en aplicaciones Java que se ejecutan en un framework de aplicaciones basado en Java y orientado a objetos encima de librerías base Java, que se ejecutan en una máquina virtual Dalvik, la cual realiza compilación JIT30. También hay librerías escritas en C que incluyen el gestor de superficie, el OpenCore Media Framework, el sistema gestor de base de datos relacional SQLite, la API gráfica 3D OpenGL ES 2.0, el WebKit layout engine, el motor gráfico SGL31, SSL, y Bionic libc. El sistema operativo Android consiste en doce millones de líneas de código que incluyen tres millones de líneas de XML, 2,8 millones de líneas de C, 2,1 millones de líneas de Java y 1,75 millones de líneas de C++.

En este caso, la programación se realiza en Java, un lenguaje bastante utilizado, optimizado y con una comunidad muy grande de usuarios. En el apartado 6.1.1 comento un poco las caracteristicas del lenguaje.

4.3.2. Apple

Apple es una empresa multinacional estadounidense que diseña y produce equipos electrónicos, software y servicios en línea, con sede en Cupertino (California, Estados Unidos) y la sede europea en la ciudad de Cork (Irlanda). Sus productos de hardware incluyen el teléfono inteligente iPhone, la tableta iPad, el ordenador personal Mac, el reproductor de medios portátil iPod, el reloj inteligente Apple Watch y el reproductor de medios digitales Apple TV.

Sus dispositivos móviles utilizan un sistema operativo llamado Ios, que únicamente se puede utilizar en sus dispositivos.

Los programadores que deseen programar aplicaciones para sus dispositivos tienen dos posibilidades que son:

- **Objective-C**. Objective-C es un lenguaje de programación orientado a objetos creado como un superconjunto de C para que implementase un modelo de objetos.
- **Swift** es un lenguaje de programación multiparadigma creado por Apple enfocado en el desarrollo de aplicaciones para iOS y macOS. Fue presentado en WWDC 2014 y está diseñado para integrarse con los Frameworks

Cocoa y Cocoa Touch, puede usar cualquier biblioteca programada en Objective-C y llamar a funciones de C. También es posible desarrollar código en Swift compatible con Objective-C bajo ciertas condiciones. Swift tiene la intención de ser un lenguaje seguro, de desarrollo rápido y conciso. Usa el compilador LLVM incluido en Xcode 6. En el año 2015 pasó a ser de código abierto.

4.4. Conclusiones

Después de cursar las asignaturas de desarrollo de aplicaciones en Android, Ios y Desarrollo front-end con frameworks Javascript considero que la mejor opción es aplicar un desarrollo nativo de nuestras aplicaciones móviles.

Como ya hemos comentado en el punto 1.3, después de una valoración interna sobre las necesidades de nuestra oficina virtual, hemos decidido optar por implementar aplicaciones nativas para Apple y Android y mantener una oficina virtual responsive accesible por navegador para el resto de dispositivos. Los motivos de esta elección han sido ideas que han aportado al proyecto desde nuestro departamento comercial donde sería necesario utilizar todas las características de los dispositivos (permitir pagos con la tecnología NFC, enviar fotos de los contadores con las lecturas del mes o alertar de alguna incidencia en la red, hacer eventos de geoposicionamento, etc.) que no son fáciles de implementar en el desarrollo web.

Con esta solución, esperamos obtener una aplicación final que otorgo al usuario una mejor experiencia aprovechando al máximo las capacidades de su dispositivo. Es cierto que, al tomar esta decisión aumentara el coste del proyecto y su posterior mantenimiento.

Otro punto a comentar en este apartado es la elección de programar primero para Android o para Apple. Analizando el mercado actual de los dispositivos móviles, y viendo el producto final a quien esta destinado, es más beneficio programar en Java que en Swift o Objective-C ya que más usuarios podrán beneficiarse de la App. Una vez este proyecto haya terminado pasaríamos a la siguiente fase, la elaboración de la Versión para Apple.

CAPÍTULO 5

Modelo de Análisis

Índice

El objetivo del análisis del sistema es obtener las especificaciones necesarias para servir de apoyo y definir con exactitud qué tareas va a realizar el nuevo sistema y así poder satisfacer las necesidades de información de la parte del diseño.

En este capítulo se describen los casos de uso como soporte a la definición de requisitos que permiten identificar los procesos que se desarrollan en el sistema y completar los requisitos de datos.

5.1. Usuarios

Para definir el contexto de uso de la aplicación es necesario establecer las características del actor que la utilizará. Está claro que una aplicación Android va a ser utilizada únicamente por usuario que posean un dispositivo con este sistema operativo, pero, como ya hemos comentado en el apartado 1.1 nuestro objetivo no es toda la comunidad Android, sino solamente aquellos usuarios que cumplan las siguientes tres características:

- Utilicen un teléfono con sistema operativo basado en Android.
- Tengan acceso a una cuenta de correo electrónico donde poder obtener las credenciales de acceso.
- Tengan un contrato vigente con la comercializadora eléctrica Electra Energía, S.A.U.

En un principio, esta aplicación tiene como publico objetivo aquellos clientes más ecologistas y ahorradores, que prefieran evitarse una llamada de teléfono o un desplazamiento a nuestras oficinas para cambiar la cuenta bancaria, el teléfono de contacto o el correo electrónico donde recibir las facturas.

En fases previas de análisis y consulta con los comerciales nos hemos dado cuenta que es importante potenciar la vertiente ecológica de este tipo de aplicaciones. Está claro que el ahorro de papel y sobres supone un beneficio económico a la empresa y una ayuda al medio ambiente, ya que evitamos la huella de CO_2 asociadas a la impresión y envió de las cartas.

Por último, la mayoría de nuestros clientes pertenecen o están relacionados a una zona geográfica muy determinada de España (el maestrazgo [1], situado al norte provincia de Castellón y sur de Teruel). Está zona se caracteriza por tener pueblos muy pequeños pero cercanos, que viven de la ganadería y el turismo rural. En futuras versiones se puede potenciar y ayudar el turismo rural de la zona informando a los usuarios sobre eventos o actos patrocinados por la empresa para potencias y ayudar la expansión de esta zona geográfica. También en caso de problemas eléctricos asociados al mal tiempo (nevadas o tormentas), podemos informar al cliente de incidencias en las redes, cortes programados o problemas en las carreteras.

Como conclusión a esta sección, nuestro potencial actor buscara una o varias de las siguiente características:

[1] El Maestrazgo (El Maestrat en valenciano) es una comarca histórica española que se extiende por el norte de la provincia valenciana de Castellón y el sureste de la provincia aragonesa de Teruel. El nombre de Maestrazgo deriva del término maestre, ya que estos territorios se encontraban bajo la jurisdicción del Gran Maestre de las órdenes militares del Temple, San Juan y Montesa. `https://es.wikipedia.org/wiki/Maestrazgo`

- Una energía limpia procedente de energías renovables como la hidráulica y la solar.
- Proximidad a la hora de tratar con la empresa.
- Tarifas adaptadas a cada tipo de cliente.
- Una oficina virtual donde gestionar tus facturas y tus contratos.
- Patrocinio de eventos culturales, sociales y deportivos.
- Experiencia
- Confianza
- Compromiso.

5.2. Contexto de uso

En la siguiente figura 5.1 se muestran los casos de uso que se describirán a continuación y que permitirán cubrir los procesos informáticos que darán soporte a los procesos que se plantean en el proyecto. En este proyecto solo existe un único actor, el usuario, el cual es el encargado de interactuar con la aplicación. A continuación, en las tablas del apartado 5.2.1 se describen cada uno de los casos de uso con todo detalle.

El siguiente diagrama se basa en la propuesta de Ivar Jacobson, Grady Booch y James Rumbaugh [19], los cuales tuvieron la idea inicial de escenario.

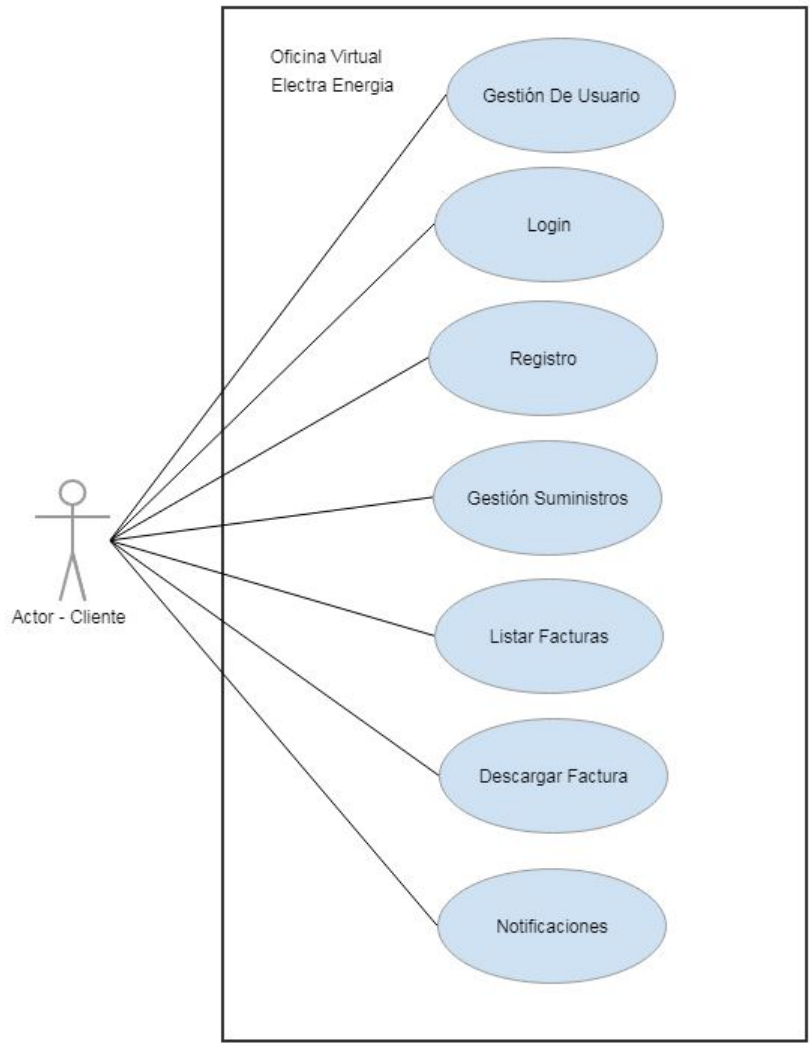


Figura 5.1: Logo Electra Energía

En la figura 5.1, el actor lo consideramos todo aquel cliente de la comercializadora eléctrica Electra Energía que quiera acceder a su contrato, a sus facturas o realizar algún tipo de modificación en la póliza a través de su teléfono Android.

Nombre:	Gloria
Edad:	25
Profesión:	Estudiante
Descripción de la persona:	Gloria es una estudiante de enfermería, que actualmente está preparando unas oposiciones para ser enfermera en hospitales de salud mental. Estudió en Huesca hace un año pero actualmente vive en Castellote. El horario habitual de esta persona es el siguiente: estudia 3 horas por la mañana, come en casa, estudia unas 3 horas más por la tarde, y a continuación intenta hacer algún tipo de actividad de ocio fuera de ella. Vive en un piso compartido en Castellote con bastantes dispositivos electrónicos en los que se encuentran: ▪ 3 ordenadores ▪ 1 nevera ▪ 1 plancha ▪ secador de pelo ▪ termo electrico ▪ microondas ▪ termo de agua . Entre todas las personas que se encuentra en casa reciben facturas de 50 euros de media anuales, con una potencia contratada de 3.3 KW
Descripción del escenario:	El sábado, después de una semana de duros estudios , deciden, junto con sus compañeras, analizar el consumo de su vivienda y intentar ver como consiguen disminuir la factura. Después de un rato, decide abrir la aplicación y consultar su consumo y analizar diferentes opciones que le ofrece la aplicación.

Nombre:	Isabel
Edad:	53
Profesión:	Profesora
Descripción de la persona:	Isabel es una profesora de Maestro de educación infantil. Profesora a tiempo completo en un colegio de Burgos en el que pasa la gran mayoría del día. Sale de casa a las 10 de la mañana y vuelve a las 8 de la tarde. Isabel tiene 53 años, esta casada y tiene 3 hijos. En su casa posee los siguientes dispositivos electrónicos en los que se encuentran: ▪ 2 ordenadores ▪ 2 nevera ▪ 1 congelador ▪ 1 plancha ▪ secador de pelo ▪ termo eléctrico ▪ microondas ▪ termo de agua ▪ aire acondicionado . La facturas que recibe de media en su casa es de un valore de 80 euros de media, con una potencia contratada de 5.5 KW
Descripción del escenario:	Es domingo por la tarde y Isabel llega a casa después de pasar el día con su mujer y los niños en Port Aventura. Mientras los niños se duchan y preparan para empezar la semana, ella y su espeso hablan de consultar una factura Descargan la aplicación y consultan la factura E5555123

Nombre:	Ismael
Edad:	74
Profesión:	Jubilada
Descripción de la persona:	Ismael es un hombre jubilado de Santander que tiene dos hijos y 5 nietos. Debido a problemas en su familia en su casa viven ahora el junto con uno de sus hijos, su esposa y los tres niños que poseen. Ismael como jubilado pasa gran parte del día en su casa. Solamente sale del bar para ir al almuerzo al bar y recoger a sus nietos y llevarlos al parque En su casa posee los siguientes dispositivos electrónicos en los que se encuentran: ▪ 1 ordenadores ▪ 1 nevera ▪ 1 congelador ▪ 1 plancha . La facturas que recibe de media en su casa es de un valore de 30 euros de media, con una potencia contratada de 0.89 KW
Descripción del escenario:	Un día, hablando con su hijo deciden intentar buscar una solución a un problema que han tenido en su factura electrónica. La aplicación ha enviado un aviso de que su factura eléctrica era muy elevada en el mes pasado y podría ser que existiera alguna anomalía. Al final de todo deciden acceder a la aplicación para ver la evolución de sus facturas y poner una reclamación.

5.2.1. Descripción de los casos de uso

Cada caso de uso va a ser detallado mediante un formulario tabular, en el que se detallaran los pasos necesarios para cada escenario. Este formulario tabular está basado en la teoría de Casos de Uso de Palmer, el cual inició la idea de especificar un sistema a partir de la interacción de los casos de uso con el entorno, y se describe a continuación:

- Identificador: identifica de forma unívoca un caso de uso siguiendo el formato de nombrado CU_X, donde X se corresponde a un número

empezando en la unidad.

- Descripción: explica que acciones se pueden realizar en cada caso de uso.
- Precondiciones: condiciones que deben darse para la realización del caso de uso.
- Post-condiciones: condiciones que son resultado de la ejecución del caso de uso.

Los casos de uso existentes son los siguientes:

Cuadro 5.1: Caso de Uso 1 - Login de usuarios

CU_1	**Login**
Descripción	La aplicación tiene que tener un sistema de seguridad. El usuario poseedor de la aplicación deberá introducir unas credenciales de acceso para acceder al contenido de la App.
Pre-Condiciones	El usuario deberá conocer sus credenciales de acceso a la aplicación. Para ello el usuario debería haber hecho uno de los dos procedimientos: ▪ Formulario de registro. ▪ Contacto con atención al cliente.
Post-Condiciones	Una vez acreditado, el usuario accederá a un menú principal que le permitirá disfrutar de las diferentes funcionalidades.

Cuadro 5.2: Caso de Uso 2 - Registro

CU_2	**Registro**
Descripción	El usuario que accede por primera vez a la aplicación tiene que tener la posibilidad de realizar un registro para crear nuevas credenciales de acceso.
Pre-Condiciones	El usuario deber ser cliente de Electra Energia y posser un correo electrónico.
Post-Condiciones	El usuario obtiene en su correo electronico una contraseña con la que acceder a la aplicación. El formulario de acceso se basara en los siguientes campos: ▪ Numero de Identificación Fiscal (NIF). ▪ Contraseña.

Cuadro 5.3: Caso de Uso 3 - Gestión De Suministro

CU_3	**Gestión De Suministro**
Descripción	El usuario puede acceder a cada uno de los diferentes contratos que posee con la empresa. Dentro de cada contrato puede realizar las siguientes modificaciones: ▪ Cambiar la cuenta bancaria. ▪ Cambiar la dirección de correo electrónico. ▪ Cambiar el teléfono de contacto.
Pre-Condiciones	El usuario deber estar autenticado en la aplicación y ser el titular de ese punto de suministro.
Post-Condiciones	El usuario puede haber modificado uno o varios de estos campos: ▪ IBAN. ▪ E-Mail. ▪ Telefono

Cuadro 5.4: Caso de Uso 4 - Gestión De Usuario

CU_4	**Gestión De Usuario**
Descripción	El usuario podrá acceder a una zona de gestión de credenciales, donde podrá modificar su contraseña o coreo electrónica en caso de que lo necesite.
Pre-Condiciones	El usuario deber estar autenticado en la aplicación y ser el titular de ese punto de suministro.
Post-Condiciones	El usuario puede haber modificado uno o varios de estos campos: ▪ Contraseña. ▪ E-Mail.

Cuadro 5.5: Caso de Uso 5 - Listar Facturas

CU_5	**Listar Facturas**
Descripción	El usuario podrá listar todas las facturas de la póliza que solicite.
Pre-Condiciones	El usuario deber estar autenticado en la aplicación y ser el titular de ese punto de suministro.
Post-Condiciones	El usuario podrá observar todas las facturas emitidas en el contrato seleccionado y ordenarlas como desee. Además también tendrá la posibilidad de observar como ha evolucionado su consumo en una gráfica configurable.

Cuadro 5.6: Caso de Uso 6 - Descarga Factura

CU_ 6	**Descarga Factura**
Descripción	El usuario podrá descargar la factura que desee en formato pdf a su dispositivo móvil Android .
Pre-Condiciones	El usuario deber estar autenticado en la aplicación, ser el titular de ese punto de suministro y acceder a la factura deseada a través de un listado.
Post-Condiciones	El usuario obtendrá un pdf con la factura que ha seleccionado.

Cuadro 5.7: Caso de Uso 7 - Notificaciones

CU_7	**Notificaciones**
Descripción	El usuario podrá configurar su propio sistema de alerta para decidir cuando desea recibir una notificación en su dispositivo. Las alertas podrán ser: ▪ Una nueva factura. ▪ Una factura superior a un determinado consumo o precio. ▪ Consejos o noticias respecto el sector.
Pre-Condiciones	El usuario deber estar autenticado en la aplicación, ser el titular de ese punto de suministro y acceder a la póliza que quiere monitorizar.
Post-Condiciones	El usuario obtendrá una notificación cuando exita: ▪ Una nueva factura. ▪ Un exceso de consumo. ▪ Una anomalía en el precio. ▪ Un evento o campaña publicitaria.

5.3. Requisitos de datos

Los requisitos de datos describen las entradas y salidas del sistema, así como la información que hay que almacenar. Hay que detallar las necesidades de información, así como también las posibles relaciones o estructuras existentes desde el punto de vista del usuario.

Los requisitos de datos son los siguientes:

- Factura
- Polissa
- Titular
- Pagador
- Usuario
- Direccio
- Iban
- CNAE
- Bank
- Tarifa
- Cups
- Preu

Antes de continuar con este apartado, he de explicar el motivo por el cual aparecen los nombres de algunos requisitos de datos en otro idioma diferente al que he escrito mi proyecto. Como hemos comentado en el apartado 2.2, OpenErp es un software libre, adaptado a las necesidades de la comercialización eléctrica por Gisce TI. Esta primera parte explica porque parte de la base de datos aparecen en ingles.

Si estuviera todo en inglés, creo que no habría ningún problema, incluso seria beneficioso ya que obtendría un carácter más global y podría ser interpretado por más gente.

El problema lo encuentro en que no es el único idioma utilizado, Gisce es una empresa catalana que en el momento de la implementación opto por utilizar el catalán para programar sus módulos. Electra Energía por una decisión de dirección ha obligado al departamento de informatica a implementar nuestros módulos en castellano.

Por ese motivo, coexisten tres idiomas en la implementación y eso es un problema importante a mi parecer. Cuando se pretende realizar una consulta nunca te acuerdas de que idioma tiene un determinado campo de una tabla y pierdes mucho tiempo buscando la tabla y viendo el nombre de sus campos.

En la elaboración de este proyecto he decidido mantener los mismos nombres de las tablas que utilizare en la API para hacer más entendible después el código en la supervisión.

Cuadro 5.8: Requisito de datos - Factura

Código	**RD_1**
Nombre	Factura
Versión	1.0
Autores	Rubén Pérez Ibáñez
Requisitos Asociados	Gestión De Suministros, Listar Facturas, Descargar Facturas
Datos Específicos	id, number, name, data_invoice, data_inici, data_final, energia_kwh, tarifa, cups, amount_total, reconciled, dies, polissa.
Importancia	Alta.

Cuadro 5.9: Requisito de datos - Polissa

Código	**RD_2**
Nombre	Polissa
Versión	1.0
Autores	Rubén Pérez Ibáñez
Requisitos Asociados	Gestión de Usuario, Gestión de Suministros, Listar Facturas, Notificaciones
Datos Específicos	id, name, data_alta, data_baixa, potencia, state, pending_amount, debt_amount, cups, tarifa, cnae, iban, titular, pagador, dirreccio_notificacio, facturas
Importancia	Alta.

Cuadro 5.10: Requisito de datos - Cups

Código	**RD_3**
Nombre	Cups
Versión	1.0
Autores	Rubén Pérez Ibáñez
Requisitos Asociados	Gestión Suministro.
Datos Específicos	id, name, direccio, polissa
Importancia	Media.

Cuadro 5.11: Requisito de datos - Tarifa

Código	**RD_4**
Nombre	Tarifa
Versión	1.0
Autores	Rubén Pérez Ibáñez
Requisitos Asociados	Gestión Suministros, Listar Facturas
Datos Específicos	id,name, llistat_de_preus
Importancia	Media.

Cuadro 5.12: Requisito de datos - CNAE

Código	**RD_5**
Nombre	CNAE
Versión	1.0
Autores	Rubén Pérez Ibáñez
Requisitos Asociados	Gestión Suministros
Datos Específicos	id, name, descripcio
Importancia	Baja.

Cuadro 5.13: Requisito de datos - Iban

Código	**RD_6**
Nombre	Iban
Versión	1.0
Autores	Rubén Pérez Ibáñez
Requisitos Asociados	Gestión Suministros
Datos Específicos	id, iban, printable_iban, bank, polizas
Importancia	Alta.

Cuadro 5.14: Requisito de datos - Bank

Código	**RD_7**
Nombre	Bank
Versión	1.0
Autores	Rubén Pérez Ibáñez
Requisitos Asociados	Gestión Suministros
Datos Específicos	id, nombre, vat
Importancia	Baja.

Cuadro 5.15: Requisito de datos - Preus

Código	**RD_8**
Nombre	Preus
Versión	1.0
Autores	Rubén Pérez Ibáñez
Requisitos Asociados	Gestión Suministros
Datos Específicos	id, periode, preu
Importancia	Baja.

Cuadro 5.16: Requisito de datos - Direccio

Código	**RD_9**
Nombre	Direccio
Versión	1.0
Autores	Rubén Pérez Ibáñez
Requisitos Asociados	Gestión Suministros
Datos Específicos	id, carrer, municipi, codi postal, provincia, pais, e-mail
Importancia	Baja.

Cuadro 5.17: Requisito de datos - Empresa

Código	**RD_10**
Nombre	Empresa
Versión	1.0
Autores	Rubén Pérez Ibáñez
Requisitos Asociados	Gestión Suministros
Datos Específicos	id, nom, nif, direccio
Importancia	Baja.

Cuadro 5.18: Requisito de datos - USUARIO

Código	**RD_11**
Nombre	USUARIO
Versión	1.0
Autores	Rubén Pérez Ibáñez
Requisitos Asociados	Login, Registro, Gestión de Usuario, Gestión Suministros, Notificaciones
Datos Específicos	id, nif, password, e-mail
Importancia	Alta.

5.4. Diagrama de clases a nivel conceptual, vista sin atributos ni operaciones

Un diagrama de clases describe la estructura de un sistema mostrando las clases, atributos y las relaciones entre ellos.

Como se puede observar en la figura 5.2, las clases que se han identificado son:

- Factura. Representa una factura asociadas a una póliza.
- Polissa. Representa un contrato de un cliente para un punto de suministro.
- Empresa. Representa toda aquella persona o empresa que esta inscrita un pagador. Una empresa puede tener diferentes roles en diferentes pólizas. Por ejemplo, en la póliza 1 puede ser el representante fiscal (titular) ya que es el propietario del domicilio y en la póliza 2 puede ser el pagador ya que es una casa alquilada.
- Usuario. Esta entidad representa las credenciales de acceso a la aplicación.
- Direccio. Representa una dirección .
- Iban. Representa una cuenta bancaria
- CNAE. La Clasificación Nacional de Actividades Económicas o CNAE de España permite la clasificación y agrupación de las unidades productoras según la actividad que ejercen de cara a la elaboración de estadísticas. Esta entidad pretende representar esta información.
- Bank. Representa un banco.
- Tarifa. Representa una determinada tarifa con un precio y una póliza asociada.

- Cups. Representa el Código Unificado del Punto de Suministro y siempre estará asociado a una póliza. Esta póliza puede cambiar ya que el CUPS va asociado a un punto de suministro no a una póliza, es decir, si una vivienda cambia de titular el CUPS seguiría siendo el mismo.

- Preu Representa el precio de una tarifa.

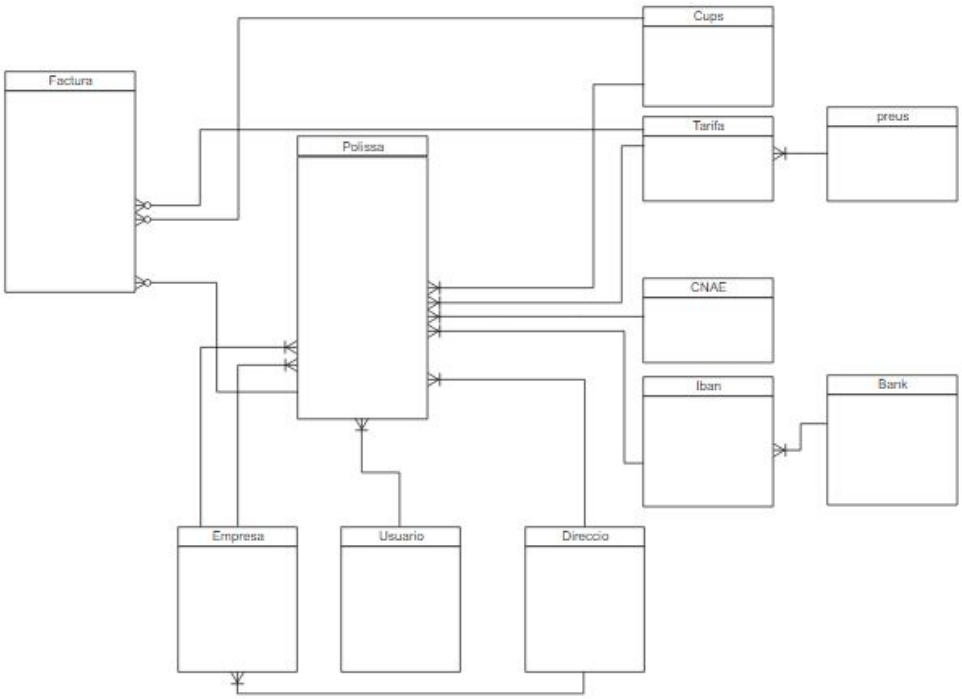


Figura 5.2: Diagrama de clases a nivel conceptual, vista sin atributos ni operaciones

5.5. Diagrama de clases a nivel conceptual, vista con atributos y operaciones

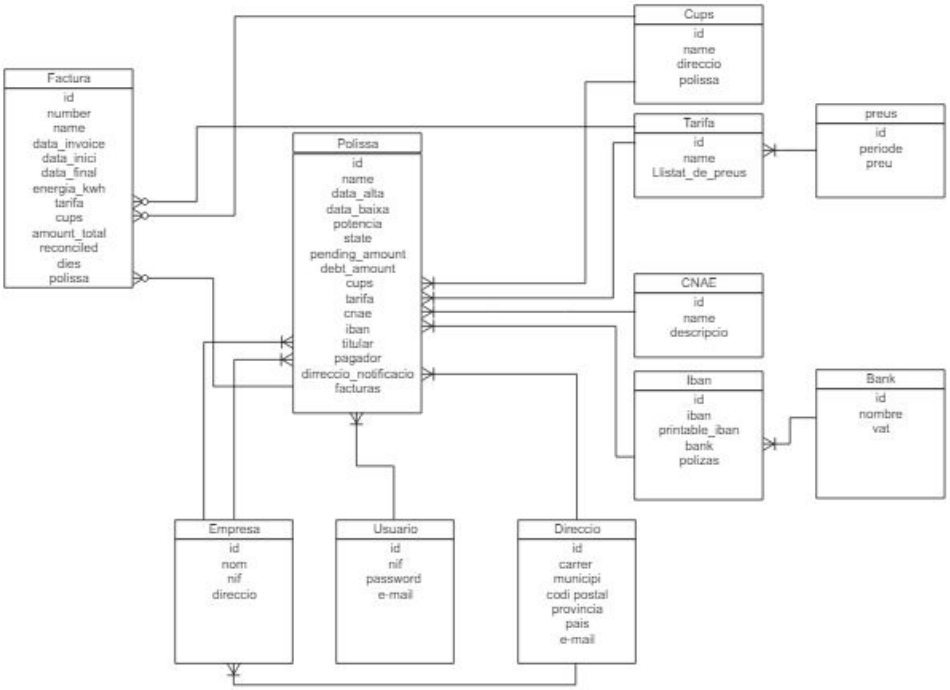


Figura 5.3: Diagrama de clases a nivel conceptual, vista con atributos y operaciones

5.6. Prototipo

El prototipo ha sido elaborado con Pencil, una herramienta de sofware libre recomendada por Francesc D'Assís Giralt. Esta aplicación permite la elaboración de un mockup donde podemos dibujar las interfaces necesarias para cumplir las funcionalidades expuestas en los apartados anteriores.

5.6.1. Identificación/ registro de usuarios

El usuario cuando acceda por primera vez en la aplicación deberá autenticarse para poder utilizar todos los servicios (figura 5.4a). Si el usuario ya poseía unas credencias de acceso para la plataforma, simplemente, tiene que introducirlas, pero, si no se encuentra registrado en el sistema se le ofrecerá la opción de darse de alta (figura 5.4b) introduciendo los siguientes datos:

- Número de contrato
- CUPS [2]
- Correo Electrónico

(a) Interfaz Login

(b) Interfaz Alta Usuario

Para mejorar la usabilidad del usuario, la aplicación otorga al usuario la facilidad de recordar sus credenciales y, así, no perder tiempo en la introducción del usuario y la contraseña.

Otro punto importante en este aspecto es la ayuda que les proporcionamos

[2]El CUPS (Código Unificado del Punto de Suministro), en España, es un código único e identificador de un punto de suministro de energía, ya sea de electricidad o gas canalizado.

a los usuarios para encontrar la información en la factura. En la figura 5.4b podemos observar el botón "Ver donde están los campos en la factura"que nos desplazara a la figura5.5. En esta interfaz el usuario podrá observar donde están localizados los campos del registro en una factura de ejemplo.

Figura 5.5: Ejemplo factura

Por último, si el usuario registrado desea modificar su contraseña o cambiar su correo electrónico podrá acceder a la interfaz expuesta en la figura 5.6

Figura 5.6: Interfaz gestión de usuario

Interfaz Principal

A través de la interfaz principal el usuario podrá acceder a las principales opciones de la aplicación:

- Gestión de Suministros
- Gestión de Usuario

En la figura 5.7 podemos observar como esta interfaz posee un botón para cada una de las funcionalidades.

Figura 5.7: Interfaz Menú

Guía

Para que un usuario primerizo puedan aprender a utilizar la aplicación desde cero, en la interfaz inicial de la aplicación existirá un botón que nos dirigirá a la guía de la aplicación. Esta pantalla servirá para que los nuevos usuarios aprendan a utilizar la aplicación tal y como se muestra en la figura 5.8 .

Figura 5.8: Interfaz Guía

Gestionar Suministros

La primera opción que vamos a estudiar es como gestionar nuestros suministros eléctricos. Como hemos comentado en las funcionalidades, el usuario puede modificar y consultar sus pólizas y sus facturas. Al presionar el botón Gestionar Suministros te aparecerá una interfaz como la que podemos observar en la figura 5.9a y, una vez hemos seleccionado el domicilio de la lista, nos proporcionara la información que podemos observar en la figura 5.9b.

(a) Interfaz Listado

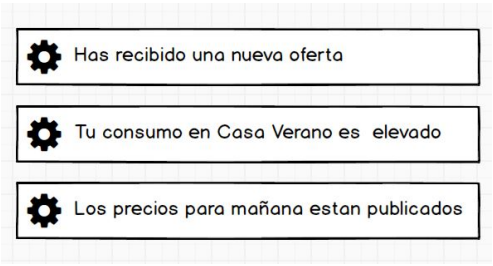
(b) Interfaz Contrato

Otro aspecto muy interesante es la alerta que tiene asociado nuestro domicilio. Entre los diferentes tipos de alertas que tiene la aplicación (por ejemplo de consumo, precio o incluso de análisis de facturas) nosotros podremos elegir aquella que más se adecue a nuestras necesidades y asociarla con el punto de

suministro. Si alguna vez la alerta se cumple el usuario recibirá una notificación en el dispositivo. Un ejemplo de estas notificación podría ser la figura 5.10b

Otra opción diferente es, por ejemplo, mostrar todos los puntos de suministro en un mapa en vez de un listado. Si presionamos el botón mapa nos aparecerá una interfaz tal y como muestra la figura 5.10a.

(a) Interfaz buscado por mapa



(b) Ejemplo Notificación

5.6.2. Interfaz Modificar Contrato.

En el caso que el usuario desee modificar alguno de los datos de su contrato como el número de teléfono o su cuenta bancaria dispondrá de la interfaz expuesta en la figura 5.11

Figura 5.11: Interfaz Modificar Contrato

5.6.3. Interfaz Listado de Facturas

En esta interfaz, el usuario puede tener acceso a sus últimas facturas y analizar la evolución de su consumo.

Normalmente, la gráfica de líneas se utiliza para representar la evolución de unos datos numéricos en el transcurso de un periodo de tiempo determinado. Nosotros la hemos utilizado para visualizar de manera rápida el sentido de la evolución, incrementos y reducciones en los consumos de los usuarios y además poder comparar si lo deseamos varios años . Este caso es mucho más visual y permite a simple vista analizar y comparar diferentes consumos en un periodo de tiempo que si fuera un simple gráfico de barras sobre el consumo. El resultado se puede analizar en la figura 5.12

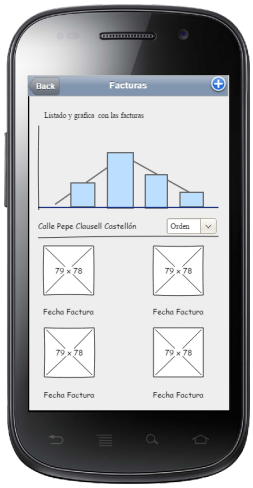


Figura 5.12: Interfaz listado de facturas

5.6.4. Flujo de la interfaz

El siguiente diagrama representa el flujo del prototipo.

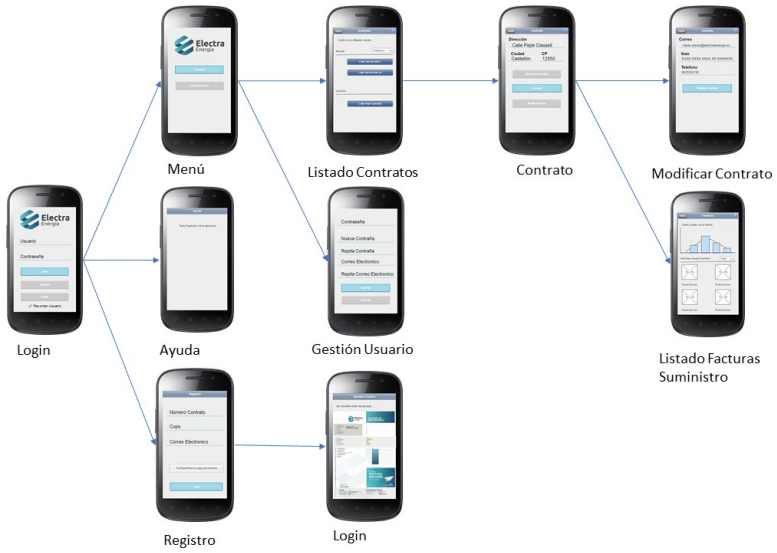


Figura 5.13: Flujo del prototipo

5.7. Diagrama Explicativo de la arquitectura del sistema

El modelo escogido en nuestro proyecto se basa en Modelo–Vista–Presentador (MVP), una derivación del patrón arquitectónico modelo–vista–controlador

(MVC), y es utilizado mayoritariamente para construir interfaces de usuario.

La intención de este proyecto es simplificar al máximo las vistas para que sea muy fácilmente modificable y extensible y en ese sentido este modelo nos ofrece esta solución.

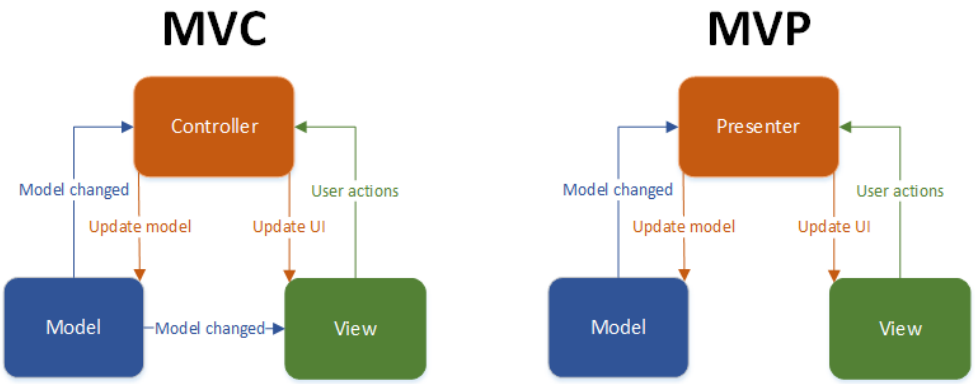


Figura 5.14: MVC vs MVP

Cabe destacar que el SDK de Android no nos ofrece de forma nativa un patrón MVP. Aún así podemos llevar a cabo alguna aproximación que nos permita desarrollar aplicaciones robustas, que sean más fáciles de mantener, escalar y testear.

Por ultimo, en la figura 5.15 se observa la arquitectura final que tendrá la aplicación.

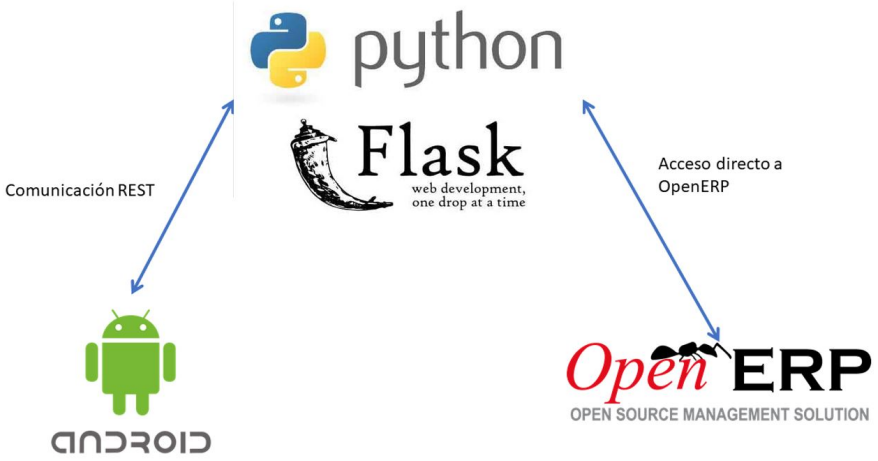


Figura 5.15: Arquitectura de la aplicación

CAPÍTULO 6

Herramientas y tecnologías usadas en el proyecto

Índice

En este apartado vamos a comentar las tecnologías empleadas para llevar a cabo el proyecto. Empezaremos en la primera sección eligiendo el lenguaje de programación que utilizaremos. En la segunda sección vamos a comentar que herramientas hemos empleado para elaborar el proyecto.

6.1. Elección del lenguaje de programación

Respecto al lenguaje de programación utilizado han sido:

- Python. Utilizado por Django para la generación de la Api
- Java. Utilizado por Android para crear las apps.

6.1.1. Python

Python es un lenguaje de programación poderoso y fácil de aprender. Cuenta con estructuras de datos eficientes y de alto nivel y un enfoque simple pero efectivo a la programación orientada a objetos. La elegante sintaxis de Python y su tipado dinámico, junto con su naturaleza interpretada, hacen de éste un lenguaje ideal para scripting y desarrollo rápido de aplicaciones en diversas

áreas y sobre la mayoría de las plataformas.[14]

El intérprete de Python y la extensa biblioteca estándar están a libre disposición en forma binaria y de código fuente para las principales plataformas desde el sitio web de Python, `https://www.python.org/`, y puede distribuirse libremente. El mismo sitio contiene también distribuciones y enlaces de muchos módulos libres de Python de terceros, programas y herramientas, y documentación adicional.

Entre sus principales carracteristicas nos encontramos:

- **Simplificado y rápido.** Es muy fácil hacer scripts en python utilizando muy pocas lineas.
- **Elegante y flexible**. Permite elaborar programas para solucionar cualquier problema de forma sencilla y de diversas formas.
- **Ordenado y limpio**. Ideal para trabajos colaborativos, ya que es muy facil de interpretar el código.
- **Portable**. Funciona en linux y en Windows.
- **Comunidad**. Existe una gran comunidad detrás de Python, ya que es el lenguaje más utilizado del mundo.

Java

La principal característica -y ventaja- de este lenguaje de programación es que se trata de un lenguaje independiente de la plataforma, es decir, cualquier programa creado a través de Java podrá funcionar correctamente en ordenadores de todo tipo y con sistemas operativos distinto. Ello es un beneficio para los programadores, pues les facilita el trabajo ya que ya no se ven obligados a crear un programa diferente que se adapte a Windows, Linux, Mac, etc.

Entre las principales ventajas nos encontramos:

- **Lenguaje Simple.** Una de las cosas más importantes que deses saber de Java, es que no es para nada complejo. De hecho la curva de aprendizaje del lenguaje es realmente corta, por lo que de inmediato podrás familiarizarte con los términos y las funciones que el lenguaje utiliza.
- **Lenguaje Orientado a Objetos** Los objetos se encargan de encapsular información, clases y funciones, las cuales se pueden manipular más adelante o se pueden agregar a distintos programas y existe lo que es la manipulación de datos entre objetos, por algo este tipo de lenguajes son mucho mas potentes.

- **Aplicaciones Distribuida**. Java tienes la posibilidad de hacer aplicaciones distribuidas. estas aplicaciones en red, lo que hacen son ejecutarse en una plataforma distribuida. Mantiene mucha estabilidad y el rendimiento se incrementa considerablemente.
- **Interpretado y Compilado**. Una de las principales ventajas de Java, es su compilación. Su compilación es tan buena, que se llega a asimilar al lenguaje ensamblador, es decir, desde la base puede ser interpretado. Esto ayuda muchísimo a la ejecución de aplicaciones compiladas en Java, pues se puede ejecutar básicamente en cualquier lugar sin mayor problema.
- **Es Seguro.** Una de las virtudes de Java, posiblemente sea su seguridad, además de que es un lenguaje a código abierto, pero sus programas están compilados tan perfecta y originalmente, que no tendrás ningún problema con filtros de seguridad.

6.2. Herramientas de desarrollo

Para desarrollar el proyecto se han utilizado las siguientes herramientas:

- Github: como herramienta para guardar copias de seguridad del proyecto y a la vez tener un control de versiones de cada nueva funcionalidad.
- Sharelatex: utilizado para escribir la memoria del proyecto.
- Trello.com : es una herramienta de Scrum que permite crear una lista de listas llena de tarjetas donde poner las diferentes tareas que consta el proyecto.
- Android Estudio: el entorno de desarrollo integrado oficial para la plataforma Android.
- Pycharm como entorno de programación para la Api de comunicación con el Servidor.

Cabe señalar que se han utilizado estas herramientas de desarrollo porque el alumno ha aprendido a utilizarlas durante sus estudios superiores en las distintas asignaturas que ha cursado.

6.2.1. Android Studio

Android Studio es un IDE muy útil en cuanto a la programación en Java se refiere, ya que proporciona muchísimas funcionalidades para que la programación sea mucho más cómoda. A continuación se nombran las características que han sido de más utilidad y que sin duda han reducido el tiempo empleado en el proyecto:

- Editor de texto con resaltado de sintaxis.
- Compilación en tiempo real.
- Auto completado inteligente.
- Plantillas para crear trozos de código, como nuevas clases, constructores, bucles, etc.
- Señalización automática de errores y avisos de compilación y sugerencias para resolverlos.
- Navegación entre clases y métodos.

6.2.2. Github

GitHub es una plataforma que permite alojar proyectos siguiendo un control de versiones Git. El código puede almacenarse de manera privada o pública. Normalmente, está abierto, para que los desarrolladores puedan trabajar en conjunto reparando bugs o implementando nuevas funcionalidades.

6.2.3. Trello.com

Trello es una herramienta colaborativa que organiza tus proyectos en tablones que te permite ver en que estas trabajando, que está realizando el resto y en que parte del proceso estas. Trello se basa en la metodología Kanban. Una metodología desarrollada por Toyota a principio de los años 40, dentro del sistema de gestión de producción JIT (just-in-time).

6.2.4. Pycharm

Pycharm es un IDE para la programación en Python . A continuación se nombran las características principales.

- Editor de texto con resaltado de sintaxis.
- Compilación en tiempo real.
- Auto completado inteligente.
- Señalización automática de errores y avisos de compilación y sugerencias para resolverlos.
- Navegación entre clases y métodos.

CAPÍTULO 7

Implementación

Índice

Una vez terminada la fase de análisis, se procede a modelar el sistema a nivel de arquitectura y componentes de software. Así pues, en este capítulo se expondrán los detalles de implementación, incluyendo algunas capturas de la aplicación desarrollada en funcionamiento.

7.1. Servidor

Desde el punto de vista del backend, busco proporcionar al frontend, una manera sencilla de obtener los datos que requieren las interfaces de usuario, concretamente, estos datos servirán tanto para una app móvil, como para una interfaz web hecha en React [1], por lo que nuestra solución debe alimentar ambos enfoques.

[1]React: React (también llamada React.js o ReactJS) es una biblioteca Javascript de código abierto para crear interfaces de usuario con el objetivo de animar al desarrollo de aplicaciones en una sola página `https://reactjs.org`

Con este proporsito, nos hemos propuesto crear una API (del inglés: Application Programming Interface) REST , que es un conjunto de subrutinas, funciones y procedimientos que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción.[2]

Buscando una definición sencilla, REST es cualquier interfaz entre sistemas que use HTTP para obtener datos o generar operaciones sobre esos datos en todos los formatos posibles, como XML y JSON.

Para conseguir este objetivo, he decidido buscar una herramienta que me permita generar una aplicación REST, fácil, estable y que pudiera permitirme aprovechar al máximo los recursos que ya disponía. Con esa idea en mente, decidimos centrarnos en framework en Phython y compatible con OpenErp.

Como resultado de esta búsqueda, encontramos FLASK y su paquete REST framework (Flask-RESTful) para el desarrollo de nuestra API. FLASK es un framework minimalista escrito en Python que permite crear aplicaciones web rápidamente y con un mínimo número de líneas de código. Está basado en la especificación WSGI de Werkzeug y el motor de templates Jinja2 y tiene una licencia BSD. Estas máximas están también en Flask-RESTful, una librería que nos permite construir un API REST sobre Flask de forma sencilla que nos ofrece una gama de métodos y funciones para el manejo, definición y control de nuestros recursos. Por lo que vamos a poder tener una API RESTful, en muy poco tiempo y de forma estable.

7.1.1. Implementación Básica

Para realizar esta parte del proyecto, es necesario crear un entorno de programación con los siguientes requerimientos:

```
Babel==2.4.0
ERPpeek==1.6.3
Flask==0.12.2
Flask-Cors==3.0.2
Flask-Login==0.4.0
Flask-RESTful==0.3.5
marshmallow==3.0.0b2
osconf==0.1.3
```

Una vez he instalado todos los paquetes para la ejecución de Python, configurado Flask junto con el paquete Flask-RESTful y configurado el servidor Apache hemos implementado los modelos definidos en en el fichero **oficina-virtual-api/models/___init___.py**.

En este documento podremos encontrar todos los objetos que se pueden obtener a través de la API. Un ejemplo ilustrativo de este documento lo podemos encontrar en la clase ContractModel, definida en el siguiente código extraído del fichero.

```
class ContractModel(Model):
    model = 'giscedata.polissa'
    serializer = ContractSchema()
    schema = unflatdot([
        'id',
        'name',
        'data_alta',
        'data_baixa',
        'potencia',
        'state',
        'pending_amount',
        'debt_amount',
        'tarifa.name',
        'cnae.name',
        'cnae.descripcio',
        'titular.name',
        'titular.vat',
        'pagador.name',
        'pagador.vat',
        'direccio_notificacio.street',
        'direccio_notificacio.phone',
        'direccio_notificacio.mobile',
        'direccio_notificacio.email',
        'cups.name',
        'cups.direccio',
        'llista_preu.name',
        'bank.iban',
        'bank.printable_iban',
        'bank.owner_id.name',
        'bank.owner_id.vat',
        'bank.bank.name'
    ])
```

El segundo paso es crear serializadores de nuestros modelos, los podremos encontrar en el archivo oficina-virtual-api/serializers/__init__.py de donde obtenemos el siguiente ejemplo:

```
class ContractSchema(ContractSimpleSchema):
    start_date = DateString(attribute="data_alta")
    end_date = DateString(attribute="data_baixa")
```

```
    tariff = fields.Nested(TariffSchema,
    attribute="tarifa")
    cnae = fields.Nested(CNAESchema)
    owner_partner = fields.Nested(PartnerSchema,
    attribute="titular")
    fiscal_partner = fields.Nested(PartnerSchema,
    attribute="pagador")
    notification_address =
    fields.Nested(AddressSchema,
    attribute="direccio_notificacio")
    contracted_power =
    fields.Float(attribute="potencia")
    cups = fields.Nested(CUPSSchema)
    pricelist = fields.Nested(PricelistSchema,
    attribute="llista_preu")
    bank_account = fields.Nested(PartnerBankSchema,
    attribute="bank")
    state = fields.Str()
    amount_pending = Float(attribute="pending_amount")
    amount_debt = Float(attribute="debt_amount")
```

Mediante este ejemplo, podremos obtener del servidor la información que vamos a proporcionar a los clientes de nuestra API REST.

Una vez tenemos todos estos datos definidos, ya podremos realizar la implementación del fichero oficina-virtual-api/api/__init__.py. Dentro de él, nos vamos a encontrar todo el sistema de enrutado de nuestra aplicación a través de la siguiente línea de código

```
resources = [
    (Invoices, '/invoices/'),
    (InvoicePdf, '/invoices/<int:invoice_id>/pdf/'),
    (Contracts, '/contracts/'),
    (ContractInvoices,
    '/contracts/<string:contract>/invoices/'),
    (CRMCategories, '/crm/categories/'),
    (CRMCases, '/crm/cases/'),
    (UserToken, '/get_token'),
    (UserTokenValid, '/is_token_valid'),
    (UserPassword, '/user/password'),
    (ApiCatchall, '/<path:path>/')
]
```

Flask-RESTful proporciona una clase Resource que nos permite definir el enrutamiento para uno o más métodos HTTP para una URL determinada.

Por ejemplo, para definir al modelo Contract el método Get hemos realizado la siguiente implementación:

```
class Contracts(ReadOnlyResource, SecuredResource):
    def get(self):
        try:
            search_params, limit, offset =
            ArgumentsParser.parse()
        except (ValueError, SyntaxError) as e:
            response = jsonify({
                'status': 'ERROR',
                'errors': {'filter': e.message}
            })
            response.status_code = 422
            return response
        try:
            search_params = [
                ('pagador.id', '=', g.partner_id)
            ]
            model = ContractModel()
            result = model.get(search_params,
            limit=limit, offset=offset)
        except ModelException:
            response = jsonify({'status': 'ERROR'})
            response.status_code = 422
            return response
        return jsonify(dict(result._asdict()))
```

7.1.2. Implematicón del login

Es normal encontrar aplicaciones donde el proceso de autenticar guarda en una sesión la información del usuario. Para ello se necesita almacenar esa información en una base de datos, podía ser una colección de MongoDB o en Redis.

Sin embargo, esto supone una pérdida de escalabilidad en nuestra aplicación, ya que el servidor debe almacenar un registro por cada vez que el usuario se autentique en el sistema. Además, hacer que el Backend se encargue de ello y de esta manera si queremos desarrollar una aplicación móvil, necesitamos otro backend diferente, no es posible reutilizarlo.

Por ello una de las nuevas tendencias es la autenticación por medio de Tokens a través de la API RESTful sin información de estado, stateless.

El funcionamiento es el siguiente. El usuario se autentica en nuestra aplicación

con un par usuario/contraseña. A partir de entonces, cada petición HTTP que haga el usuario va acompañada de un Token en la cabecera. Este Token no es más que una firma cifrada que permite a nuestro API identificar al usuario. Pero este Token no se almacena en el servidor, si no en el lado del cliente (por ejemplo, en localStorage o sessionStorage) y el API es el que se encarga de descrifrar ese Token y redirigir el flujo de la aplicación en un sentido u otro.

Como los tokens son almacenados en el lado del cliente, no hay información de estado y la aplicación se vuelve totalmente escalable. Podemos usar el mismo API para diferentes apliaciones (Web, Mobile, Android, iOS, ...) solo debemos preocuparnos de enviar los datos en formato JSON y generar y descrifrar tokens en la autenticación y posteriores peticiones HTTP a través de un middleware.

También nos añade más seguridad. Al no utilizar cookies para almacenar la información del usuario, podemos evitar ataques CSRF (Cross-Site Request Forgery) que manipulen la sesión que se envía al backend. Por supuesto podemos hacer que el token expire después de un tiempo lo que le añade una capa extra de seguridad.

Utilizando esta información, hemos implementado el fichero oficina-virtual-api/login/___init___.py donde se define el proceso de login que se realiza mediante un sistema de autenticación por token.

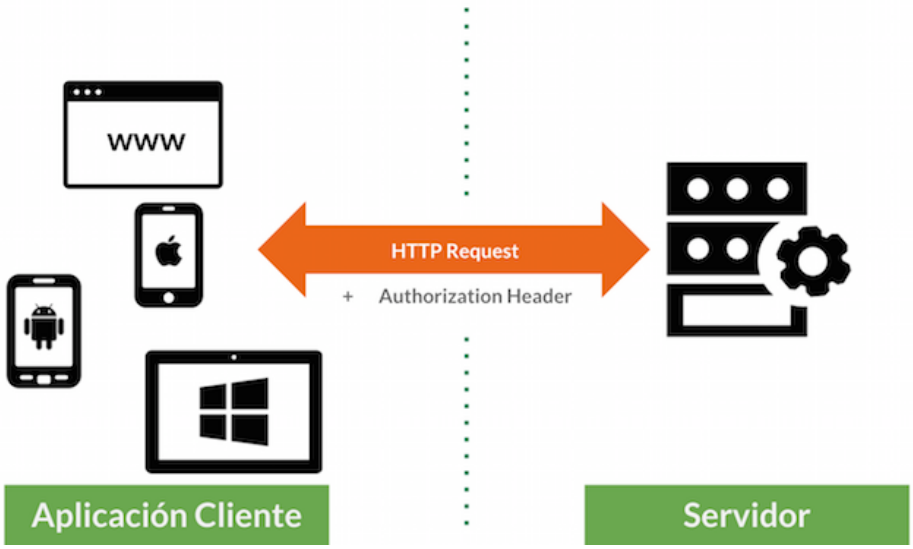


Figura 7.1: Autenticación basada en tokens

7.1.3. Resulta de la implementación

A continuación, voy a exponer un listado con todas las preticiones programas y un ejemplo de respuesta:

1. Obtener el listado de contractos

```
GET /api/v1/contracts
{
  "name": "43242",
  "amount_pending": 50.0,
  "amount_debt": 0.0,
  "start_date": "2016-01-01",
  "end_date": "2017-12-31",
  "tariff": {
    "id": 3,
    "name": "2.0A"
  },
  "cnae": {
    "id": 4535,
    "code": "9820",
    "description": "Actividades de los hogares como
    productores..."
  },
  "owner_partner": {
    "id": 5,
    "name": "Marisol Lopez",
    "vat": "ESXXXXXXX"
  },
  "fiscal_partner": {
    "id": 5,
    "name": "Marisol Lopez",
    "vat": "ESXXXXXXX"
  },
  "notification_address": {
    "id": 456,
    "street": "Sylvie Lelitre, 69 rue de Chimay  5478
    Wavre",
    "phone": "54564353",
    "mobile": "354353453",
    "email": "marisol@example.com"
  },
  "contracted_power": 5.463,
  "cups": {
    "id": 32,
```

```
    "name": "ES1234000000000001JN0F",
    "street": "carrer inventat 1 1 1 1 aclaridor 00001
    (Poble de Prova)"
  },
  "pricelist": {
    "id": 3,
    "name": "Public Pricelist"
  },
  "bank_account": {
    "id": 325,
    "iban": "ES9604030706712594221547",
    "printable_iban": "ES96 0403 0706 7125 9422 1547",
    "owner": {
      "id": 342,
      "name": "Pepito palotes",
      "vat": "ESXXXXXXX"
    },
    "bank": {
      "id": 342,
      "name": "LA FAIXA"
    }
  }
}
```


2. Listado de facturas asociadas a un contrato.

```
Getting invoices from a contract
GET /contracts/<name>/invoices
```

3. Factura

```
GET /api/v1/invoices
```


```
{
  "id": 43,
  "number": "0007/F",
  "date": "2017-03-13",
  "partner": {
    "id": 5,
    "name": "Marisol Lopez",
    "vat": "ESXXXXXXX"
  },
  "contract": {
    "id": 34,
    "name": "354352"
```

```
    },
    "energy_consumed": 453,
    "amount_total": 34.43,
    "amount_total_printable": "34,43 euros",
    "paid": true,
    "number_of_days": 31,
    "start_date": "2016-01-01",
    "end_date": "2016-01-31",
    "tariff": {
      "id": 3,
      "name": "2.0A"
    },
    "cups": {
      "id": 32,
      "name": "ES1234000000000001JN0F",
      "street": "carrer inventat 1 1 1 1 aclaridor 00001
      (Poble de Prova)"
    },
    "pricelist": {
      "id": 3,
      "name": "Public Pricelist"
    },
  }
```


4. Descarga de facturas

   GET /invoices/<id>/pdf

5. Devuelve información del usuario descargado.


```
   GET /api/v1/user
  {
    "user": "123456789R",
    "partner": {
      "id": 5,
      "name": "Marisol Lopez",
      "vat": "ESXXXXXXX"
    },
  }
```


7.2. Aplicación Android

Una vez hemos realizado la implementación de la API REST nos queda analizar la implementación del cliente Android a través de un análisis de su

estructura, funcionalidades e interfaces.

Con este objetivo, voy a empezar analizando el sistema de conexión con la API REST, posteriormente analizare el modelo de arquitectura utilizado para acabar analizando las diferentes interfaces que nos podemos encontrar en el proyecto.

7.2.1. Comunicación con la API REST

Dentro de nuestro proyecto Java, nos encontraremos con un paquete muy importante, llamado ComunicacionApi que contiene la clase ApiRestOpenErp. En esta clase se han definido los métodos necesarios para hacer las request al servidor que hemos implementado en el apartado 7.1.

En la primera versión en PreProducción hemos utilizado un certificado expedio por letsencrypt.org [2]. Esta web está gestionada por una autoridad de certificación. Su finalidad es proporcionar certificados X.509 gratuitos para el cifrado de Seguridad de nivel de transporte (TLS) a través de un proceso automatizado diseñado para eliminar el complejo proceso actual de creación manual, la validación, firma, instalación y renovación de los certificados de sitios web seguros.

Cuando utilizamos esta clase, lo que realmente ejecutamos es la implementación necesaria para cargar el certificado en la máquina virtual java permitiendo utilizar este tipo de certificados y, por lo tanto, poder establecer conexiones https.

```
CertificateFactory cf =
CertificateFactory.getInstance(this._context.getString
(R.string.comunication_tipo_certificado));
        InputStream caInput
        =this._context.getResources().openRawResource(
        R.raw.comeredmgiscenet);
        Certificate ca;
        try {
            ca = cf.generateCertificate(caInput);
        } finally {
            caInput.close();
        }

        // Crea un KeyStore que contiene las
        autoridades certificadores que nosotros
        confiamos
```

[2] https://letsencrypt.org/

```
String keyStoreType =
KeyStore keyStore =
KeyStore.getInstance(keyStoreType);
keyStore.load(null, null);
keyStore.setCertificateEntry(this._context.get
String(R.string.comunication_autoridad_certifi
cadora), ca);

// Creaa un  TrustManager donde anyadimos
nuestro certificado para que el cliente
Android confie en el certificado.
String tmfAlgorithm =
TrustManagerFactory.getDefaultAlgorithm();
TrustManagerFactory tmf =
TrustManagerFactory.getInstance(tmfAlgorithm);
tmf.init(keyStore);

//Almacena en la variable _sslContext el
TrustManager parar realizar posteriomente las
diferentes conexions
_sslContext = SSLContext.getInstance("TLS");
_sslContext.init(null, tmf.getTrustManagers(),
null);
```

Una vez hemos configurado la máquina virtual con el certificado, es posible realizar peticiones al servidor a través de la variable ___sslContext como vemos en el siguiente ejemplo:

```
URL url = new
URL(this._context.getString(R.string.comunicatio
n_host)+"contracts/"+codabonado+"/"+this._contex
t.getString(R.string.comunication_funcion_invoic
es));
  HttpsURLConnection urlConnection =
          (HttpsURLConnection)
          url.openConnection();
  urlConnection.setSSLSocketFactory(_sslContext.
  getSocketFactory());
  urlConnection.setRequestMethod("GET");
  urlConnection.setRequestProperty("Authorizatio
␣␣␣␣␣␣␣␣n", this.leerToken());
```

7.2.2. Utils

Dentro de nuestro proyecto Java, nos encontraremos con un paquete compuesto de las siguientes clases:

1. AlertasPersonalizadas: El objetivo de esta clase es crear diálogos de alerta para informar a los usuarios cuando se produzca un problema.

2. Internet: Su objetivo es analizar si el dispositivo tiene o no acceso a internet.

3. RandomLetras: Esta clase se encarga de crear un texto de confirmación en el proceso de login.

4. Mail: Utilizamos estos objetos para enviar correos electrónicos a las direcciones que se obtengan a través de los parámetros de los diferentes objetos.

7.2.3. Modelos de datos.

En este paquete, crearemos todos los objetos que la aplicación utiliza para recopilar los datos del servidor. Unos objetos pueden contener a otros como, como, por ejemplo, Items_Contratos contiene un listado de Contract lo que permite reutilizar los objetos en diferentes clases.

Este paquete contiene los siguientes elementos principales:

- Items_Contratos Modelo de datos necesario para representar un Contrato.
- Items_Invoices Modelo de datos necesario para representar una Factura.
- Token Modelo de datos necesario para representar un Token.
- Token_is_valid Modelo de datos necesario para representar un Token.
- Usuario Modelo de datos necesario para representar un Usuario.

7.2.4. Presenter

Como hemos comentado en el apartado 5.7, el modelo escogido en nuestro proyecto se basa en Modelo–Vista–Presentador. En el inicio del proceso de desarrollo del proyecto, me di cuenta que, todas las clases Presentador que implementaba compartían cierta parte del código.

Con este objetivo, desarrolle una clase abstracta para que todos los objetos que implementaran está función ya tuvieran parte del código implementado.

7.3. Interfaz de usuario

Puesto que el proyecto desarrollado consta de una aplicación para móviles Android, las diferentes interfaces de usuario de ésta se han de implementado, como se ha comentado anteriormente, con el lenguaje de marcas XML y Java.

En el proceso de implementación de la interfaz de usuario se han tomado como pilares fundamentales los siguientes principios de diseño y usabilidad:

- Diseño sencillo. La interfaz no debe abrumar al usuario con elementos innecesarios. La atención del usuario debe estar constantemente puesta en lo que quiere hacer.

- Interfaz familiar. Se ha pretendido en todo momento facilitar al usuario la navegación, aprovechando al máximo los componentes comunes a la mayoría de sitios web. Es el caso, por ejemplo, del menú superior de navegación.

- Metáforas visuales. La mayoría de los botones que conducen a acciones van acompañado s de un icono que representa su funcionalidad. De esta manera, el usuario puede hacerse una idea a priori de lo que va a suceder.

- Consistencia. Las distintas páginas que componen la interfaz poseen un aspecto muy parecido en cuanto a colores y a posición de los componentes. Además, acciones similares producen efectos similares.

- Jerarquía visual. Cuando se desea realizar una tarea, es importante que la interfaz permita al usuario centrarse en los elementos más relevantes para llevarla a cabo.

  La jerarquía visual se consigue mediante el tamaño, color, contraste, proximidad o la alineación de los diferentes elementos que intervienen.

- Feedback. Es absolutamente indispensable que la aplicación informe en todo momento al usuario de lo que está sucediendo. De esta forma se consigue evitar la sensación de incertidumbre que se produce, por ejemplo, cuando no se sabe si una acción se ha podido llevar a cabo con éxito. El feedback aporta mucho valor a la aplicación como producto.

- Reflejar estado. Es importante no obligar al usuario a recordar, por lo que en todo momento la interfaz debe reflejar el estado en que se encuentra.

- Mensajes de confirmación. Se debe pedir confirmación antes de realizar operaciones importantes, como por ejemplo eliminar un usuario del sistema.

Como resultado de tomar como base esta serie de principios, se ha conseguido una interfaz intuitiva, estéticamente agradable y fácil de utilizar. A continuación, se presentan las interfaces de las vistas más representativas que conforman la aplicación desarrollada y sus principales métodos y propiedades.

7.3.1. Login

Esta primera clase voy a realizar un análisis más exhaustivo, para presentar el modelo estudiado en el apartado 5.7.

Dentro del paquete Actividades.Login, nos encontraremos con las clases Java que conforman el proceso de autenticación de un cliente. Estas clases son:

- LoginActivity. Subapartado 7.3.1
- LoginPresenter. Subapartado 7.3.1
- LoginTask. Subapartado 7.3.1

Como podemos ver, esta primera actividad utiliza el sistema MVP (Modelo-Vista-Presentador), explicado en el apartado 5.7. La intención de este punto es exponer los principales métodos e implementación para cada una de las clases que conforman la tarea de autenticación.

LoginActivity

La finalidad de esta clase es generar la interfaz necesaria para permitir al usuario autentificarse en la aplicación. Aquellos clientes que esten registrados podrán ver sus pólizas, facturas y datos. Para ello el usuario deberá introducir las credenciales de acceso en dos objetos EditText.

Los principales métodos que podemos encontrar en esta clase son:

- **protected void onCreate(android.os.Bundle savedInstanceState)**
  Esta función se encarga de inicializar todos los elementos de la interfaz almacenados en **R.layout.activity_login**. La finalidad del método es asociar a todas las variables gráficas de la clase sus respectivos elementos gráficos del layout, el comportamiento que van a tener y sus funcionalidades. En este método inicializaremos:
  - private AutoCompleteTextView **mDNIView**: Se encarga de obtener y gestionar el comportamiento del campo DNI de la vista.

- private EditText **mPasswordView**: Se encarga de obtener y gestionar el comportamiento del campo Contraseña de la vista.
- private View **mProgressView**: Se encarga de gestionar el funcionamiento del proceso de carga y su interacción con la interfaz. Para ello, durante el proceso de autenticación, el cliente podrá observar un elemento que indica el progreso de la operación según la figura7.2

Figura 7.2: Ejemplo de ejecución del elemnto mProgressView

- private View **mLoginFormView**: Esta variable se encarga de gestionar la visibilidad del formulario cuando se produce la autenticación.
- private Button **mEmailSignInButton**: Se encarga de gestionar la acción de presionar el botón de Login. Una vez se ha presionado, la actividad ejecutará el método attemptLogin() ;
- private Button **mregistro** Se encarga de gestionar la acción de presionar el botón de Registro. Una vez se ha presionado, la actividad ejecutará el método go_to_registro().
- View **__focusView** : Gestión la posición del cursor en la vista.

Dentro de la clase nos vamos a encontrar más variables internas relacionadas con el funcionamiento de la aplicación y la interacción con el usuario que son:

- boolean **debug**: Necesaria para la fase de test, ayuda a realizar pruebas, y permite el acceso al contenido completo de la aplicación. Cuando esta variable contiene el valor *true*, los campos de la vista están rellenados, permitiendo el acceso a todos los contenidos de la App.
- LoginPresenter **__presenter**: Variable que contiene un objeto del tipo LoginPresenter analizado en el apartado 7.3.1
- private SharedPreferences **sharedPref**: Datos que una aplicación debe guardar para personalizar la experiencia del usuario.

Para terminar con este método, el resultado de su ejecución es la figura 7.3

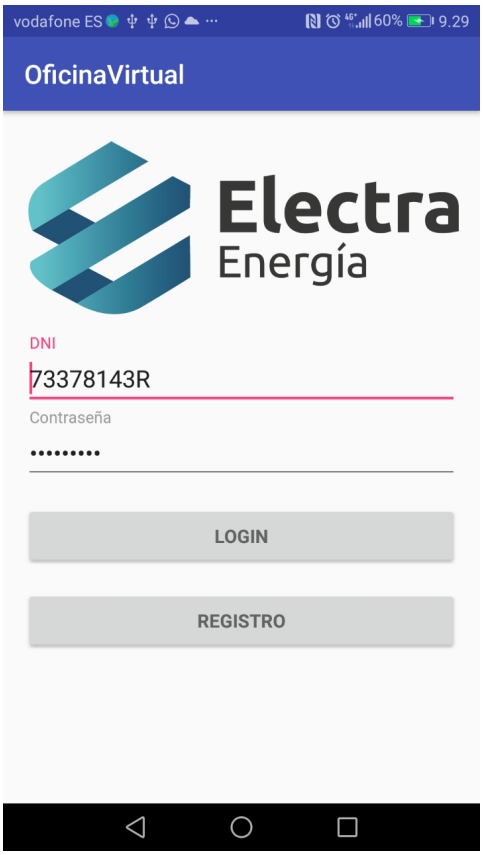


Figura 7.3: Login aplicación en Huawei P10 Lite

- **private void attemptLogin()**: Se encarga de reiniciar las variables mDNIView y mPasswordView de los posibles errores de una introducción de datos previa, almacenar los datos introducidos para futuros procesos de logueo e iniciar el proceso de login del presenter.

- **activarPresenter()**: El objetivo de esta función es permitir a la actividad realizar varias invocaciones a la tarea de LoginTask. Las clases AsyncTask solamente se pueden ejecutar una vez, por lo tanto, es necesario reiniciar la memoria cada vez que se produzca una invocación a este tipo de objeto.

- **void desactivarBotones()**: Esta función se encarga de deshabilitar la aplicación, eliminar la actividad y liberar sus recursos, es decir, su función es cerrar la aplicación en el caso que sea invocada. Solamente la he utilizado para el caso que el usuario no tenga acceso a internet (figura 7.4)

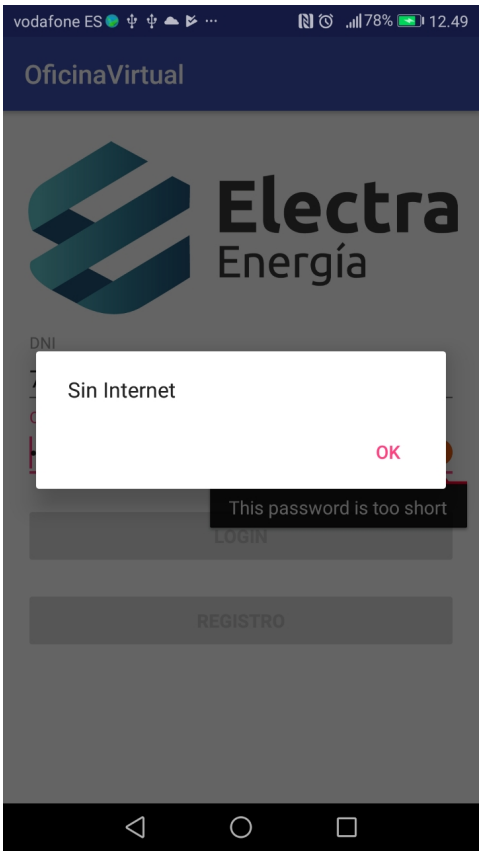


Figura 7.4: Aviso conexión Internet en Huawei P10 Lite

- **static LoginActivity getInstance():** Sirve para utilizar una instancia de la propia actividad desde otro objeto Java.

- **void showProgress(final boolean show):** Se encarga de gestionar y controlar las variables **mLoginFormView** y **mProgressView**. Depen-

diendo del valor del parametro **show** mostrara o no el elemento mostrado en la figura 7.2 o la figura 7.3

- **void dniInvalid()** Alerta al usuario de un error en la introducción del DNI. Para ello muestra un texto en el objeto **mDNIView**.
- **void passwordInvalid()** Alerta al usuario de un error en la introducción del Password. Para ello muestra un texto en el objeto **mPasswordView**.
- **void go_to_registro()** Se encarga de redirigir al usuario a la actividad destinada para el registro.
- **void siguienteActividad(java.lang.String s)** Se encarga de redirigir al usuario a la actividad destinada para mostrar el listado de polizas.

LoginPresenter

Es la clase encargada de recoger eventos desde la vista, ejecutar una acción y si es necesario, volver a interactuar con la vista a través de la interfaz. Para ello, esta clase dispone de los siguientes métodos principales:

- **void procesoLogin(java.lang.String usuario, java.lang.String password)** Este el principal método de la actividad. Su finalidad es ejecutar la tarea asincrona de autenticación. Para ello ejecuta la variable mAuthTask del tipo LoginTask siempre y cuando se haya introducido correctamente el nombre de usuario y la contraseña en la interfaz.
- **void controlShowProgress(boolean gestionProgres)** Este método se encarga de gestionar el proceso de carga de la interfaz, invocando la función showProgress de la actividad LoginActivity.
- **private boolean isEmailValid(String email)** Se encarga de comprobar si el correo tiene un formato valido.
- **private boolean isPasswordValid(String password)** Su función es analizar si la contraseña introducida posee los criterios básico de la aplicación.
- **public void siguienteActividad_ListadoPolizas()** Este método ordena a la interfaz ejecutar la siguiente actividad, el listado de pólizas.
- **errorLogin()** Ordena a la interfaz mostrar un error en el proceso de autenticación.

Como podemos observar, este objeto gestiona toda la parte lógica de la actividad e interactúa con los componentes visuales a través de los métodos

que posee la interfaz. De esta forma, un cambio en el funcionamiento de la aplicación implica analizar donde afectará el cambio. Si corresponde al funcionamiento lógico sera necesario modificar o crear nuevos métodos para resolver el problema dentro de esta clase.

LoginTask

LoginTask es una clase java que extiende el objeto AsyncTask. Debido a los problemas comentados anteriormente, el proceso de autenticarse debe realizarse en un hilo secundario, ya que, aparte de evitar la saturación del hilo principal de la aplicación nos permite alertar al usuario de la carga de los datos y mantener activa la interfaz gráfica.

Analizado este aspecto, los método principal que nos podemos encontrar en esta clase son:

- **protected Boolean doInBackground(Void... params)** Esta método se encarga de ejecutar e invocar los métodos que enviar la petición al servidor REST.
- **protected void onPostExecute(final Boolean success)** Este método se encarga de alertar al Presenter de que la ejecución del hilo ha terminado.
- **protected void onCancelled()** La finalidad de esta función es alertar al objeto Presenter que la ejecución del hilo se ha cancelado.

7.4. Registro

El objetivo de esta actividad es proporcionar al usuario la capacidad de registrarse en la aplicación. Debido a un estudio legal que estamos elaborando en este momento, aún no se ha podido implementar una conexión directa entre el registro en la aplicación y el acceso a la misma ya que no están definidos por parte de la empresa que información es necesaria obtener para validar el cliente.

En sustitución de ese proceso hemos implementado un formulario de login a través de Firebase [6] que nos permite obtener las siguientes funcionalidades en el proyecto:

- Una visión profunda sobre el uso de la aplicación por parte de los usuarios.
- Una plataforma para mensajes y notificaciones para Android, iOS, y aplicaciones web que actualmente puede ser usada de forma gratuita.

- Un servicio que puede autenticar los usuarios utilizando únicamente código del lado del cliente

- Una base de datos en tiempo real y back-end.

Utilizando la base de datos que nos proporciona Firebase, hemos implementado un formulario de login para crear un usuario cada vez que se rellenen los datos mostrados en la figura 7.5. Una vez los datos son válidos, se envía un correo a la dirección introducido con un texto.

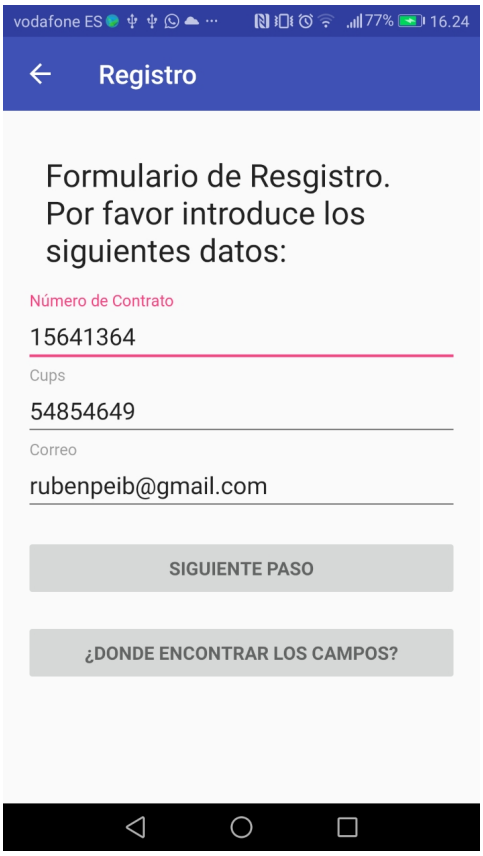


Figura 7.5: Registro en Huawei P10 Lite

En el caso que, el usuario ya este registrado en la aplicación o el contrato ya este dado de alta, se notifica un error al usuario como se muestra en la figura 7.6.

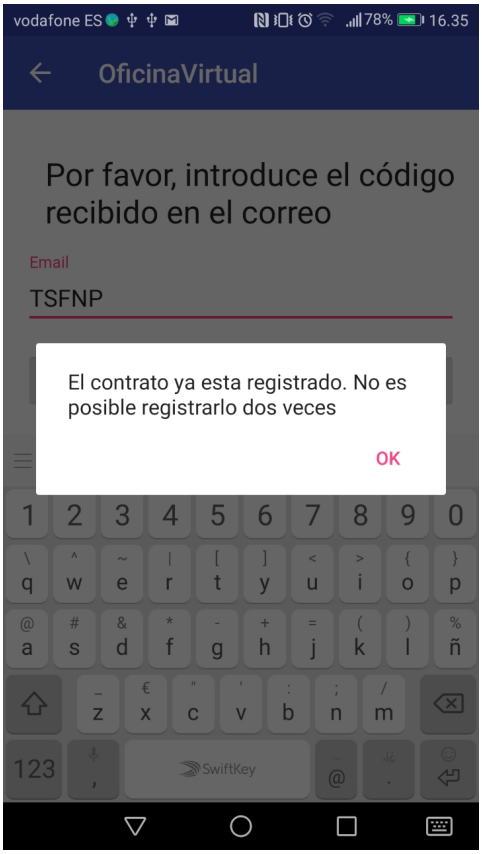


Figura 7.6: Error registro en Huawei P10 Lite

7.5. Listado de polizas

La finalidad de esta actividad es mostrar el listado de pólizas asociado al cliente. En la figura 7.7 se puede observar que, el cliente logueado con el DNI 73378143R, posee 2 contratos activos, el número 50010084 y el 50011011.

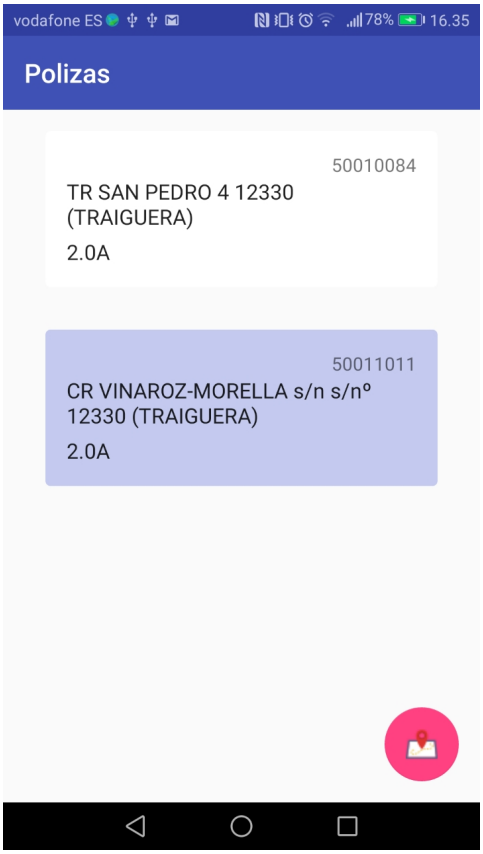


Figura 7.7: Listado de pólizas en Huawei P10 Lite

Como podemos observar desde el punto de vista del diseño, se ha escogido un listado, donde dependiendo de la posición que ocupe en la lista el elemento tendrá un color diferente.

También en la parte inferior derecha hemos activado un botón flotante por si el usuario quiere analizar donde están localizadas sus contratos en un mapa. Este botón es importante, ya que muchos de los clientes de la empresa tienen contratada la luz de sus segundas o terceras viviendas y, por lo tanto, no son sus viviendas habituales y no conocen su localización o nombre de la calle. En estas situaciones un mapa es bastante más útil que un listado.

7.6. Listado de pólizas en un Mapa

Como hemos comentado en el punto 7.5, para Electra Energia es importante ofrecer al usuario la posibilidad de localizar sus puntos de suministro en un mapa. Para ello hemos implementado la interfaz mostrada en la figura 7.8, donde se puede observar el resulatado de la implementación.

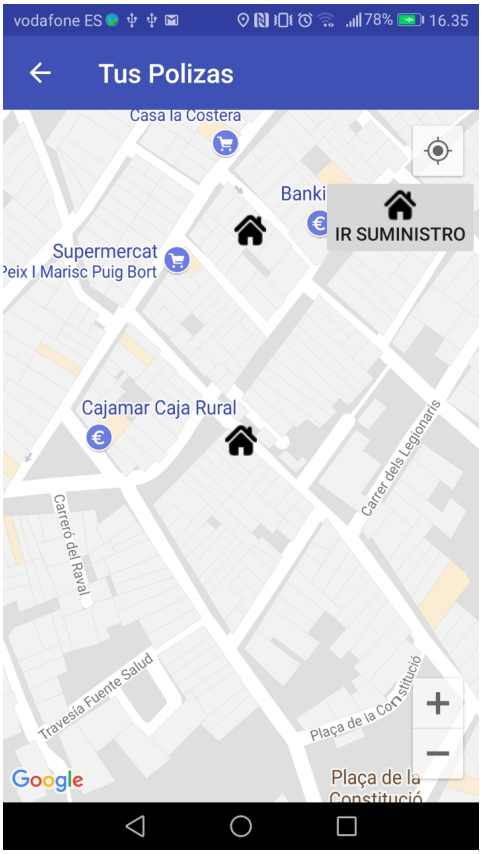


Figura 7.8: Listado de pólizas en un mapa utilizando un Huawei P10 Lite

Mediante el botón Ir Suministros, el usuario puede desplazarse entre los diferentes puntos de suministro.

7.7. Listado de facturas

El objetivo de la activada es poder mostrar al usuario un listado con las ultimas 10 facturas emitidas. Para ello, hemos implementado una actividad compuesta por dos listview. Dependiendo del tipo de objeto que va a mostrar, nos proporcionara un elemento del tipo factura o un elemento del tipo gráfica de consumo.

El motivo de aplicar esta implementación es devido a que, mediante un único listview no funcionaba adecuadamente. El objeto ChartDataAdapter de la clase ListViewBarChartActivity no se adaptaba adecuadamente al movimiento de la gráfica, ya que reproducía elementos del tipo gráfico en posiciones que correspondían a las facturas cuando se producía un desplazamiento en la vista.

Por último, en la figura 7.9 nos encontramos una gráfica creada utilizando la libreria MPAndroidChart[8] en la versión v3.0.3

Figura 7.9: Listado de facturas utilizando un Huawei P10 Lite

7.8. Enviar Factura

Para terminar, en la figura 7.10, se observa un campo modificable, que almacena en Firebase el correo donde se quiere recibir las facturas a través de la aplicación.

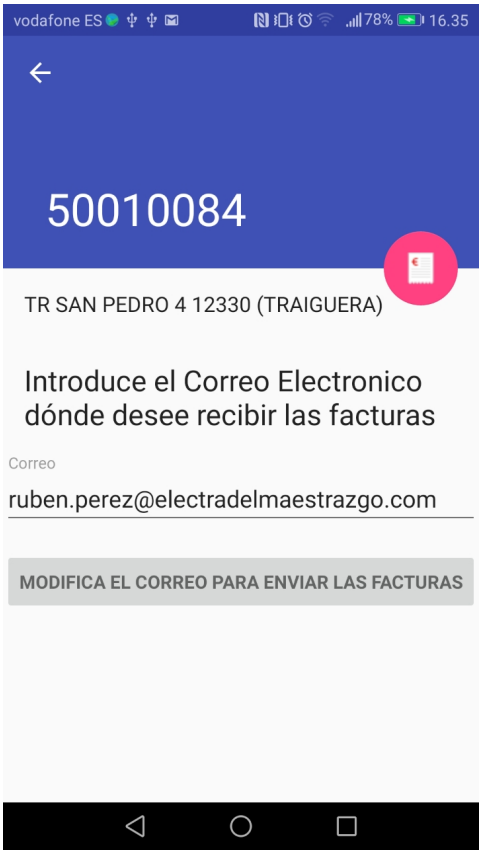


Figura 7.10: Modificar correo en Huawei P10 Lite

Una vez hemos seleccionado la factura deseada del aparatado 7.7 podremos acceder a la figura 7.11, donde, utilizando la librería de Google, PdfRenderer-Basic hemos conseguido transformar un Pdf en una imagen y mostrarla en la pantalla.

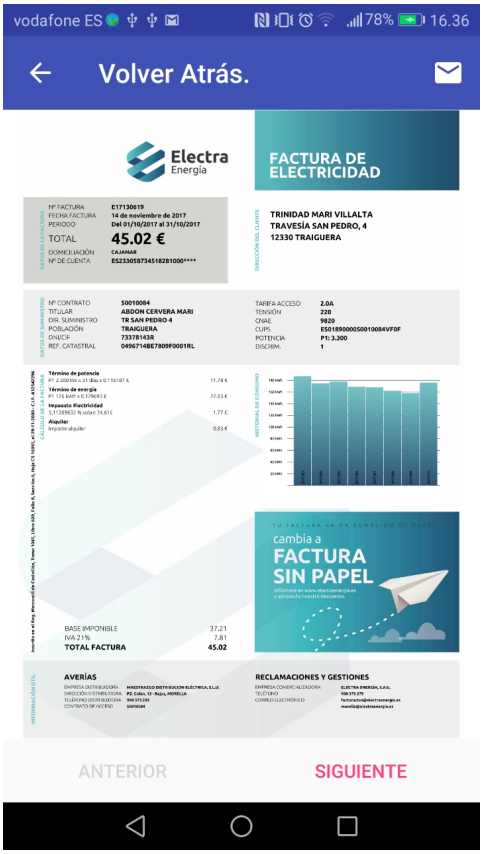


Figura 7.11: Factura en Huawei P10 Lite

CAPÍTULO 8

Verificación y validación

Índice

El objetivo de la fase de verificación y validación del sistema es comprobar el correcto funcionamiento de éste a todos los niveles. Esta fase es esencial en el proceso de desarrollo de software ya que proporciona información sobre la calidad del sistema que se está desarrollando.

Existen varios tipos de pruebas según su alcance o el tipo de componente que se está probando:

- Pruebas unitarias
- Pruebas de integración
- Pruebas de carga y estrés
- Pruebas de aceptación

El objetivo de todo desarrollo software no es llevar a cabo todos los tipos de pruebas, sino que para cada caso concreto hay que realizar un estudio con el objetivo de definir qué pruebas son útiles y viables, teniendo en cuenta variables como el tiempo o el presupuesto. En este proyecto se han desarrollado pruebas unitarias y de aceptación.

8.1. Pruebas unitarias

Las pruebas unitarias son aquellas que se encargan de comprobar el correcto funcionamiento de cada módulo o componente que constituye el sistema. La prueba de cada módulo es independiente de los demás, de esta forma nos aseguramos de que cada uno de los módulos trabaja como se espera por separado.

Para desarrollar las pruebas unitarias en este proyecto se ha utilizado la herramienta JUnit. JUnit es una librería de código abierto que permite definir casos de pruebas unitarias para programas escritos en Java.

Como se puede apreciar en el ejemplo siguiente, los métodos que constituyen test unitarios se identifican con la anotación @Test y es habitual asignarles nombres muy descriptivos.

```
@Test
    public void get_token() throws Exception
    {
        Context appContext = InstrumentationRegistry.getTargetContext();
        ApiRestOpenErp comunicacion = new ApiRestOpenErp(appContext);
        comunicacion.getToken("20488355R","20488355R");
        assertNotEquals(null,comunicacion.leerToken());
        System.out.println(comunicacion.leerToken());
    }
```

Todas las pruebas implementadas se han estado ejecutando con mucha frecuencia para asegurar que cada cambio no ha afectado a la lógica implementada hasta el momento.

Por esto, las pruebas que se han desarrollado han sido lo más independientes y simples posible, ya que el objetivo es consumir el menor tiempo posible en ejecución de test unitarios.

No obstante, se debe tener en cuenta que en algunos casos la implementación de las pruebas es más costoso que la ganancia que obtendría la calidad de la aplicación, por lo que sólo se han realizado pruebas para aquellos módulos o funciones en los que realmente ha merecido la pena invertir tiempo en ello.

8.2. Pruebas de aceptación

Las pruebas de aceptación de una aplicación las realizan generalmente un conjunto de usuarios finales y sirven para asegurarse de que la aplicación es capaz de llevar a cabo todas las historias de usuario establecidas durante la etapa de requisitos.

En este proyecto las pruebas de aceptación las han realizadas por el desarrollador. Se ha detallado una lista con todas las pruebas a pasar para cada módulo implementado antes de la entrega del producto final, mediante las cuales se comprueban los siguientes casos:

- **Caso de Uso 1 - Login de usuarios:** Una vez introducido los datos, el usuario accede a las pólizas asociadas correctamente.
- **Caso de Uso 2 - Registro:** Mediante un formulario de acceso, el usuario puede realizar la petición para que le den acceso a la aplicación. Si el usuario ha sido previamente registrado, le es denegada esta opción.
- **Caso de Uso 3 - Gestión De Suministro:** El usuario puede modificar la dirección donde envía la factura.
- **Caso de Uso 5 - Listar Facturas** El usuario tiene la capacidad de observar en un listado las diferentes facturas y, además, observar en una gráfica la evolución de su consumo.
- **Caso de Uso 6 - Descarga Factura:** El usuario tiene la posibilidad de observar la factura y enviarse por correo un duplicado.

CAPÍTULO 9

Conclusiones y trabajo futuro

Índice

En este capítulo, se comentan las conclusiones que se ha obtenido el alumno a partir de la realización del proyecto, así como posibles ampliaciones o mejoras que se podrían aplicar en un futuro al producto final.

9.1. Conclusiones

El propósito de este proyecto ha sido implementar una aplicación Android enfocada a la gestión de una oficina virtual. La consecución de los objetivos no ha sido satisfactoria ya que no se ha completado con éxito todo el trabajo previsto. Los casos C_4 Gestión De Usuario y C_7 Notificaciones no han podido llevarse a cabo por falta de tiempo. El motivo de este descuadro a sido un error en la valoración de la complejidad del proyecto.

Como experiencia, decir que es muy satisfactorio el poder ver cómo se pueden aplicar conjuntamente algunos de los conceptos que se han ido viendo durante el Máster. La realización del proyecto ha sido posible gracias a los conocimientos adquiridos en las diferentes asignaturas recibidas en la carrera y al estudio individual mediante la información web que hay en Internet y a que a medida que vas resolviendo problemas que van surgiendo o aplicando nuevas ideas, vas también aprendiendo cosas nuevas.

Me gustaría comentar también la facilidad con que se visualiza desde el exterior el desarrollo de aplicaciones móviles y la complejidad real que tiene desde dentro hasta la más mínima función.

Al finalizar este proyecto, puedo decir que he aprendido a crear una aplicación móvil Android desde el principio hasta el final, he conocido nuevos entornos de desarrollo que no había usado nunca, he podido practicar con técnicas y herramientas que ya conocía gracias a la carrera cursada pero que después de finalizar este proyecto he podido consolidar los conocimientos que tenia de ellas.

Por otra parte, este proyecto me ha servido para crecer profesionalmente también en cuanto a tecnologías y herramientas de desarrollo. He aprendido a identificar soluciones para distintos problemas en el ámbito de las aplicaciones para dispositivos móviles, así como a desarrollar mi capacidad para llevarlas a cabo de manera competente.

9.2. Trabajos futuros

El objetivo de este proyecto es muy ambicioso, ya que no se centra en un caso particular, sino que se deja abierta su aplicación para distintas funciones. Por lo tanto, es evidente que en la duración estimada para realizar esta asignatura no se han podido cumplir todas las posibles funciones que se quisieran implementar.

Durante la elaboración de la aplicación base, se aparecían muchas ideas para introducir en la aplicación final, pero debido al tiempo tanto personal como establecido, no se ha podido buscar la manera de implementarlas y añadirlas a la funcionalidad.

Por ello, la extensión del proyecto se podría mejorar con los siguientes puntos:

- Permitir al cliente realizar cambios en sus pólizas como podrían se:
  - Cambios del número de la cuenta bancaria.
  - Cambios de direcciones.
  - Cambios de métodos de pagos
- Pago de facturas.
- Módulo de reclamaciones.
- Módulo de consejos al cliente.

- Creación de un sistema de notificaciones y alertas como por ejemplo avisos de una nueva factura.
- Módulo de eventos
- Módulo de noticias
- Pagos por TPV.

Otro trabajo futuro bastante interesante seria la elaboración de la misma aplicación para dispositivos IOS.

Bibliografía

[1] Google Analytics.
https://www.google.com/analytics/

[2] Interfaz de programación de aplicaciones Wikipedia
https://es.wikipedia.org/wiki/Interfaz_de_programaci%C3%B3n_de_aplicaciones

[3] ELECTRA ENERGÍA, S.A.U.
http://electraenergia.com/

[4] Android
https://www.android.com/

[5] Apple
www.apple.com/es

[6] Firebase
https://firebase.google.com/?hl=es-419

[7] PdfRendererBasic
https://developer.android.com/samples/PdfRendererBasic/index.html

[8] MPAndroidChart
https://github.com/PhilJay/MPAndroidChart

[9] flask
http://flask.pocoo.org/

[10] Introducción a los dispositivos móviles *Julián David Morillo Pozo*
PID_00176753

[11] Entornos de programación móviles *Julián David Morillo Pozo*
PID_00176754

[12] Desarrollo de aplicaciones basadas en Android *Robert Ramírez Vique*
PID_00178750

[13] Wikipedia *Sistema de planificación de recursos empresariales*
https://es.wikipedia.org/wiki/Sistema_de_planificaci%C3%B3n_de_recursos_empresariales

[14] Python
http://docs.python.org.ar/tutorial/3/real-index.html Ventajas

[15] Web mongodb.org *Características de MongoDb*
http://www.mongodb.org/

[16] PCMag *El 99.6 % del mercado móvil le pertenece a Android y iOS*
http://latam.pcmag.com/sistemas-operativos-moviles/18490/news/el-996-del-mercado-movil-le-pertenece-a-android-y-ios

[17] Omicrono *Estos son los lenguajes de programación más usados de 2017*
http://omicrono.elespanol.com/2017/07/lenguaje-programacion-mas-usado-2017/

[18] Wikipedia *Simple Object Access Protocol*
https://es.wikipedia.org/wiki/Simple_Object_Access_Protocol

[19] El lenguaje unificado de modelado. Isbn 84-7829-037-0 *Ivar Jacobson James Rumbaugh Grady Booch*
https://ingenieriasoftware2011.files.wordpress.com/2011/07/el-lenguaje-unificado-de-modelado-manual-de-referencia.pdf

[20] Miguel Angel Alvarez *Ventajas e inconvenientes de API REST para el desarrollo*
https://desarrolloweb.com/articulos/ventajas-inconvenientes-apirest-desarrollo.html