_ TELECOM ESCUELA TÉCNICA VLC SUPERIOR DE UPV INGENIEROS DE TELECOMUNICACIÓN

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

APLICACIÓN MÓVIL MULTIPLATFORMA Y WEB PARA LA GESTIÓN DE UNA COMUNIDAD DE VECINOS

Autor: Niny Johanna Plata Sánchez

Tutor: Francisco José Martínez Zaldívar

Trabajo Fin de Grado presentado en la Escuela Técnica Superior de Ingenieros de Telecomunicación de la Universitat Politècnica de València, para la obtención del Título de Graduado en Ingeniería de Tecnologías y Servicios de Telecomunicación

Curso 2015-16

Valencia, 04 de julio de 2016

Resumen

El presente TFG se centra en la realización de una aplicación multiplataforma para dispositivos móviles y de una aplicación web para la gestión de una comunidad de vecinos. Para alcanzar nuestro objetivo se intentarán extraer conclusiones sobre las tecnologías que nos permitan realizar este tipo de aplicaciones y que funcionen en entornos tanto móviles como de escritorio. Para ello, hemos hecho un estudio de los *frameworks* destinados a su desarrollo, seleccionando los que más se adaptan a las necesidades que hemos determinado. Una vez conocidos los *frameworks*, hemos estudiado las tecnologías que utilizan (HTML, CSS, JavaScript, etc.).

El desarrollo de este trabajo, como explicaremos más adelante, se divide en dos partes. La primera parte es la aplicación web, donde hemos hecho uso de JSF (JavaServer Faces) para Java. En la segunda parte, es la aplicación para dispositivos móviles, la herramienta utilizada ha sido PhoneGap, que nos permite poder abarcar casi todas las plataformas existentes en el mercado. Para todo ello se ha incluido las funcionalidades básicas y necesarias para poder verificar la viabilidad de la aplicación como futuro producto comercial.

Resum

El present TFG es centra en la realització d'una aplicació multiplataforma per a dispositius mòbils i d'una aplicació web per a la gestió d'una comunitat de veïns. Per a aconseguir el nostre objectiu s'intentaran extraure conclusions sobre les tecnologies que ens permeten realitzar aquest tipus d'aplicacions i que funcionen en entorns tant mòbils com d'escriptori. Per a açò hem fet un estudi dels *frameworks* destinats al seu desenvolupament, seleccionant els que més s'adapten a les necessitats que hem determinat. Una vegada conegut els frameworks, hem estudiat les tecnologies que utilitzen (HTML, CSS, Javascript, etc.).

El desenvolupament d'aquest treball, com explicarem més endavant, es divideix en dues parts. La primera part és l'aplicació web, on hem fet ús del *framework* JSF (JavaServer Faces) per a Java. En la segona part, l'aplicació per a dispositius mòbils, l'eina utilitzada ha sigut PhoneGap, que ens permet poder abastar quasi totes les plataformes mòbils existents en el mercat. Per a tot açò s'han inclòs les funcionalitats bàsiques i necessàries per a poder verificar la viabilitat de l'aplicació com a futur producte comercial.

Abstract

This TFG focuses on making a multiplatform application for mobile devices and a web application for managing a community of neighbors. To achieve our goal will try to draw conclusions about the technologies that allow us to perform this type of application and operating in both mobile and desktop environments. To do this, we have made a study of the development *frameworks* intended for selecting the best adapted to the needs we have identified. Once the *frameworks* were known, we studied using technologies (HTML, CSS, JavaScript, etc.).

The development of this work, as explained below, is divided into two parts. The first part is the web application, where we made use of JSF (JavaServer Faces) for Java. In the second part, the application for mobile devices, the tool used has been PhoneGap, which allows us to cover almost all existing platforms on the market. For all this has included the Basic and necessary ideas to verify the feasibility of the application as future commercial product functionality.

Índice

Índice de figuras

Índice de tablas

Capítulo 1. Introducción

En este capítulo se quiere manifestar la motivación del trabajo técnico realizado para este Trabajo de Fin de Grado, así como una breve descripción del mismo.

1.1 Motivación

Es interesante observar el desarrollo tecnológico que ha ido experimentado la sociedad desde mediados del siglo XX hasta hoy, no cabe duda de que más que un avance se ha producido una verdadera revolución. El descubrimiento de la informática, su aplicación en todo tipo de áreas de conocimiento y de producción ha cambiado a la sociedad y a su economía, de una forma más rápida incluso que cualquier otro hecho o descubrimiento anterior. La aparición de Internet, y sobre todo su apertura al público general, ha reforzado la importancia de los ordenadores en la vida social, laboral o académica de cualquier persona. Simultáneamente a esto, existe otro fenómeno que ha ido cada vez más formando parte de la vida de las personas de una forma que hoy en día se hace indispensable para todo, esto es, el teléfono móvil. El boom en la implantación de Internet, junto al furor de la telefonía móvil, confirma que esta revolución tecnológica no sólo afecta a la investigación o la actividad económica, sino que implica un fenómeno sociológico donde la comunicación y el acceso a la información en cualquier lugar y momento son sus pilares básicos. Haciendo un análisis de todo lo que esto ha implicado, surgió la motivación de formar parte de esta nueva era tecnológica y por ello se pensó la idea de desarrollar una aplicación web de gestión y su app para dispositivos móviles que pudiese realizar la misma función.

La funcionalidad de la aplicación se ha basado en la experiencia de vivir en una comunidad de vecinos, en donde se ha podido observar la cada vez más marcada necesidad de poder tener acceso a toda la información relacionada con la administración de una finca por parte de un propietario de una forma rápida e intuitiva. Este hecho resulta sumamente importante para un propietario que en muchas ocasiones desconoce en tiempo real, los movimientos, gestiones y transacciones que se realizan en su comunidad por parte de los administradores de fincas. La mala gestión, el no presentar información clara y detallada a los propietarios, muchas veces desembocan en quejas, reclamos, inconformidades que hace que las personas implicadas en una comunidad se sientan estafadas y decidan no seguir cumpliendo con sus obligaciones.

Es importante resaltar que haciendo uso de estas herramientas tecnológicas como son las aplicaciones para web y móvil, el trabajo para realizar la gestión de una comunidad de vecinos por parte de un administrador por ejemplo, se vuelve más sencillo, práctico y sobre todo útil a la hora de presentar todo tipo de información de una forma transparente a sus propietarios. Sin embargo, este proyecto en su primera versión que es la abordada en este TFG, cuenta con las funcionalidades básicas tanto para un propietario como para un administrador. Dejando pendiente llegar a completar esta aplicación en todas sus funcionalidades por si en un futuro pudiese ser vendible a gestorías que se dedican a la administración de comunidades de vecinos y a los mismos propietarios que son los que más se pueden beneficiar de este tipo de aplicaciones, ya que se vive en una sociedad donde es tan importante poder tener siempre a la mano todo tipo de información que sea relevante. Estas líneas futuras también representa una gran motivación para el desarrollo a priori y posterior de este proyecto.

1.2 Breve descripción del proyecto

La aplicación se desarrollará para su acceso mediante navegadores web. A su vez, también se realizará una app para dispositivos móviles multiplataforma. ¿Por qué Multiplataforma? Esto es, porque se pretende que la aplicación desarrollada sea una solución viable para el mayor número de sistemas operativos móviles existentes, o al menos para los de uso más extendido, con lo que se pueda llegar a cubrir el mayor número de usuarios posibles en el mercado. Para ello, vamos a emplear un *framework* que nos permita esta funcionalidad con el mayor ahorro de código posible.

De esta manera la aplicación desarrollada permite a los propietarios de la comunidad de vecinos obtener información relevante del sistema de información.

Para introducirnos y entender mejor el por qué se ha querido dedicar este proyecto en desarrollar una aplicación móvil y una aplicación para su acceso mediante navegadores web. Es interesante hacer un pequeño estudio acerca de la gran demanda del uso del Internet y el papel fundamental que desempeña en todos los ámbitos de la vida. Así mismo, la gran penetración que están teniendo hoy en día los teléfonos inteligentes, llamados habitualmente "Smartphone" como puerta de acceso a todo tipo de servicios

Antes del año 2014 el ordenador era el dispositivo a través del cual se accedía fundamentalmente. En 2014 se produjo un empate entre el ordenador y el teléfono móvil, y en el 2015 el *Smartphone* continuó ganando peso entre los internautas hasta el día de hoy.

Merece la pena también destacar que España muestra una posición aventajada en el grado de adopción de la Sociedad de la Información y, en concreto, en el uso de dispositivos. Así, en un estudio realizado entre los principales países en los que opera Telefónica (Reino Unido, Alemania, Brasil, Argentina y España) los internautas españoles muestran el mayor grado de adopción respecto a los diferentes dispositivos existentes en el mercado. Estos datos se recogen en la siguiente figura:

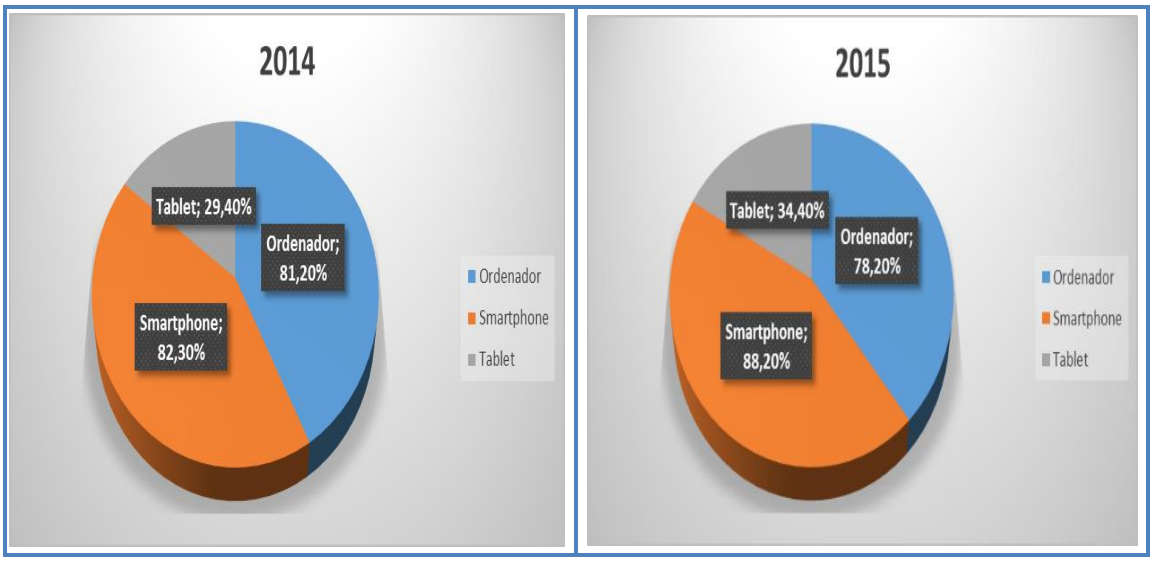


Figura 1. Porcentaje de utilización de los diferentes dispositivos

Otro fenómeno cuyo avance se observó durante el año 2015, ha sido cómo Internet se ha ido empleando en todas las esferas de la vida y su uso sigue creciendo en los tres ámbitos importantes de cada persona: trabajo y educación, ocio y comunicación. Hace años que Internet dejó de ser algo especial y novedoso al que se accedía a través de un solo dispositivo y en momentos concretos. Hoy en día Internet se utiliza en la realización de todo tipo de actividades cotidianas y se accede a él a través de un número cada vez mayor de aparatos. En este sentido, el ordenador personal con el tiempo dejó de ser el dispositivo de preferencia para acceder a la red y su puesto lo ocupó el *Smartphone*, aunque en una situación prácticamente de paridad.

Por otro lado, existe otro dato interesante y es que en España existen 23 millones de usuarios activos de «app» que realizan 3,8 millones de descargas diarias de aplicaciones. De media, cada usuario tiene instaladas 39 aplicaciones, por 33 de los usuarios de «Tablet». Son datos que se recoge en la edición del informe «La Sociedad de la Información en España», correspondiente a 2015 [1].

Otra dato importante a tener en cuenta es la cuota de mercado de los sistemas operativos que se utilizan en los diferentes dispositivos móviles actualmente.

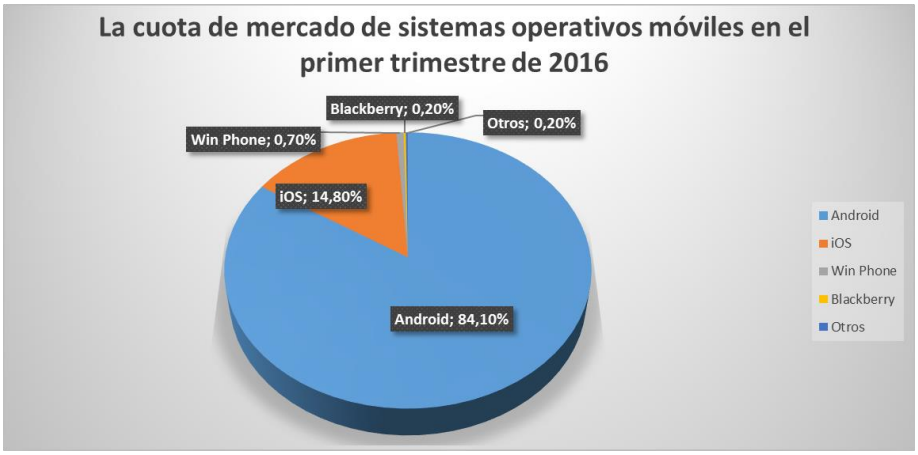


Figura 2. Cuota de mercado de los sistemas operativos móviles.

Este estudio ha sido realizado por la consultora "Gartner" y en ellos se puede ver algo muy claro, Windows Phone está hundido y como es de suponer, Android y iOS son los líderes en este aspecto.

Siguiendo con esta línea, se ha hecho otra comparativa respecto a estos sistemas operativos, la cual se ha recogido del "Google Trends" [2] y donde se puede observar la tendencia que se tiene sobre las app de Android con respecto a las aplicaciones desarrolladas para las demás plataformas.

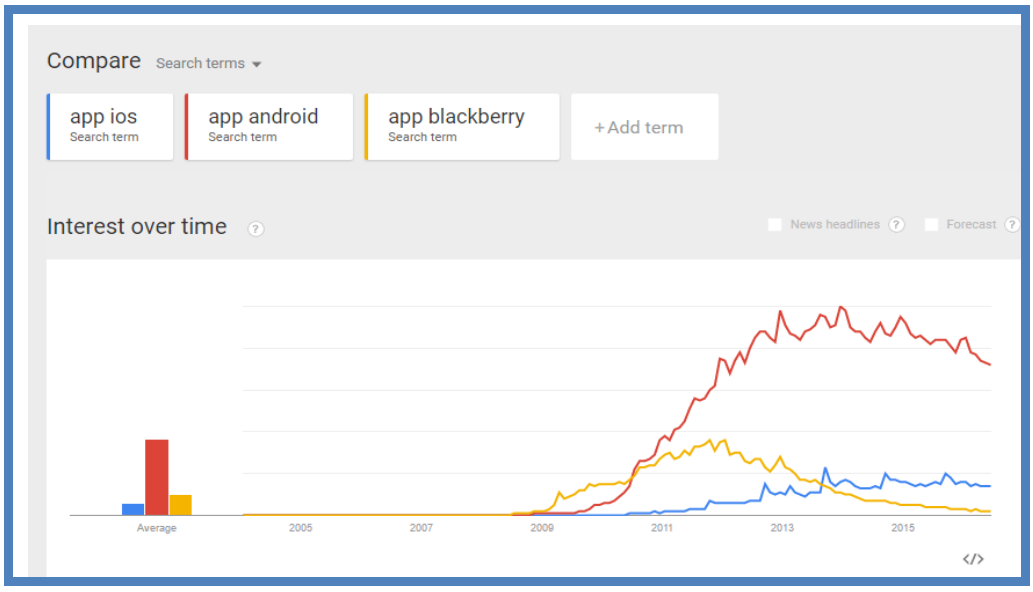


Figura 3. Comparativa de aplicaciones para los diferentes sistemas operativos móviles.

Por último, en la memoria se tratarán de describir todas las herramientas empleadas en el desarrollo y cómo han sido utilizadas durante la implementación de la aplicación.

Capítulo 2. Objetivos

Este proyecto pretende poner en práctica los conocimientos adquiridos a lo largo de toda la carrera, de este modo su objetivo es el planteamiento, desarrollo e implementación de la solución tecnológica más adecuada para el desarrollo de una aplicación en entorno web y móvil para la gestión de una comunidad de vecinos. Para ello se tienen que cumplir los siguientes objetivos clave:

1. Diseñar una arquitectura de integración tecnológica de la aplicación móvil y web (cliente), y su contraparte servidor, que pueda satisfacer los requerimientos presentes y también los futuros.

2. Desarrollar una aplicación web que implemente todos los casos de usos expuestos en el diseño.

3. Desarrollar una aplicación móvil multiplataforma, que permita ofrecer información útil al usuario final.

4. Diseñar – implementar una base de datos relacional que contenga toda la información necesaria para el desarrollo de la aplicación, siguiendo a su vez todos los criterios necesarios que eviten la redundancia y protejan la integridad de los datos.

5. Reducir en la medida de lo posible el coste de nuestro proyecto, para ello se pretende usar programas y tecnologías de código libre.

De este modo, teniendo todos los objetivos y documentación implicada para el desarrollo de este proyecto, se realizó el planteamiento de una solución que fuera acorde con los objetivos anteriormente planteados en forma de tareas que serán detallados en el próximo capítulo.

Capítulo 3. Metodología del trabajo

En este capítulo 3, se detalla la metodología utilizada para llevar a cabo el Trabajo de Final de Grado, haciendo una explicación detallada de cómo se ha gestionado el proyecto

3.1 Gestión del proyecto

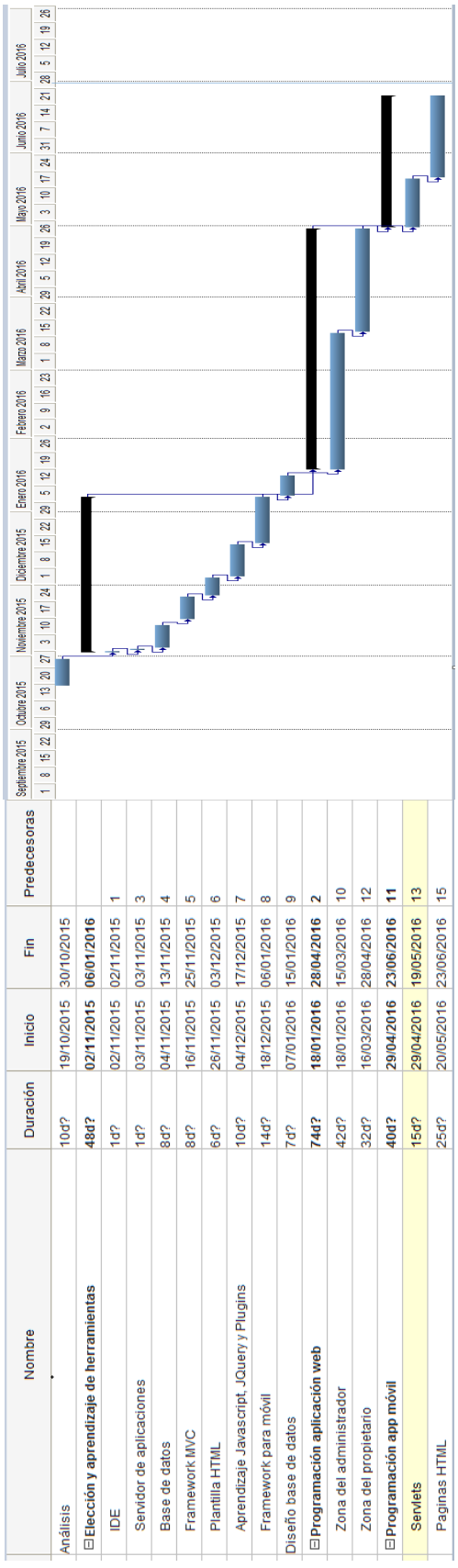
El trabajo técnico que recoge este proyecto se ha venido realizando en paralelo por un lado, con otras actividades externas a la universidad (trabajo laboral) y por otro lado, con la realización de las últimas asignaturas de la carrera. Es por ello, que este proyecto se ha ido extendiendo a lo largo de los últimos meses, siendo como objetivo principal el aprendizaje de una gran mayoría de tecnologías necesarias en el campo de la Telemática y la programación con los que se había tenido en algunos casos muy poco contacto hasta el momento, y en otros, ha servido para profundizar en conceptos estudiados en la carrera. Esto ha servido como base en muchas de las fases de elaboración de este TFG. Por tanto, este trabajo ha ido avanzado al ritmo de las distintas fases, en donde, en algunas de ellas la curva de aprendizaje ha sido más lenta teniendo que afrontar y resolver los distintos problemas que se fueron presentando a medida que se iba avanzando en el proyecto.

3.2 Distribución de las diferentes tareas a realizar

A continuación se detallan todas las tareas que se han realizado en este TFG, las cuales se han clasificado en diferentes fases que se exponen a continuación:

1. Análisis. Captura de requerimientos.
2. Elección y aprendizaje de las herramientas a utilizar en la programación.
3. Elección del sistema de gestión de bases de datos e instalación del mismo.
4. Elección del IDE y aprendizaje de su funcionamiento.
5. Elección del servidor de aplicaciones.
6. Elección del sistema de gestión de bases de datos e instalación del mismo.
7. Elección del *framework* MVC.
8. Elección de una plantilla HTML para la vista de la web.
9. Aprendizaje de lenguaje JavaScript. Aprendizaje de JQuery y *plugins* empleados.
10. Elección del *framework* en el que realizar la app móvil. Instalación y aprendizaje del modo de funcionamiento de PhoneGap.
11. Diseño la base de datos y creación del script inicial.
12. Programación de la aplicación web.
13. Programación de la zona del administrador.
14. Programación de la zona de los propietarios.
15. Programación de la app móvil.

16. Programación de los *servlets* receptores de las llamadas Ajax.



Nombre	Duración	Inicio	Fin	Predecesoras
Análisis	10d?	19/10/2015	30/10/2015	
Elección y aprendizaje de herramientas	**48d?**	**02/11/2015**	**06/01/2016**	
IDE	1d?	02/11/2015	02/11/2015	1
Servidor de aplicaciones	1d?	03/11/2015	03/11/2015	3
Base de datos	8d?	04/11/2015	13/11/2015	4
Framework MVC	8d?	16/11/2015	25/11/2015	5
Plantilla HTML	6d?	26/11/2015	03/12/2015	6
Aprendizaje Javascript, JQuery y Plugins	10d?	04/12/2015	17/12/2015	7
Framework para móvil	14d?	18/12/2015	06/01/2016	8
Diseño base de datos	7d?	07/01/2016	15/01/2016	9
Programación aplicación web	**74d?**	**18/01/2016**	**28/04/2016**	**2**
Zona del administrador	42d?	18/01/2016	15/03/2016	10
Zona del propietario	32d?	16/03/2016	28/04/2016	12
Programación app móvil	**40d?**	**29/04/2016**	**23/06/2016**	**11**
Servlets	15d?	29/04/2016	19/05/2016	13
Paginas HTML	25d?	20/05/2016	23/06/2016	15

Figura 4. Diagrama de Gantt.

Capítulo 4. Sistema de información para una comunidad de vecinos

En este capítulo se detalla el análisis y diseño para la aplicación web y móvil que se pretende crear, la cual recibe el nombre de “e-vecinos”.

4.1 Análisis

4.1.1 Captura y especificación de requisitos

En esta fase se estudia los casos de uso y los escenarios a realizar y se detallan los requisitos que se deben cumplir. De estos requisitos se identifican dos actores, el administrador y el propietario. Habrá un súper usuario que será el que dé de alta a los diferentes administradores, y a partir de ahí se parte de que es el administrador el encargado de dar de alta en el sistema a los propietarios de cada finca que se administre.

Para la realización del diagrama de casos de uso estructurado y del diagrama de clases, se ha hecho uso de la herramienta “StarUML”, el software se licencia bajo una versión modificada de GNU GPL en versión beta y el cual ha sido instalado y ejecutado previamente en el ordenador de trabajo [3].

En el siguiente diagrama los círculos de color azul son los casos de uso asignados al administrador, los de amarillo asignados al propietario, los verdes son los casos de uso en común y los blancos serán los propuestos para el trabajo futuro de este proyecto en su próxima versión.

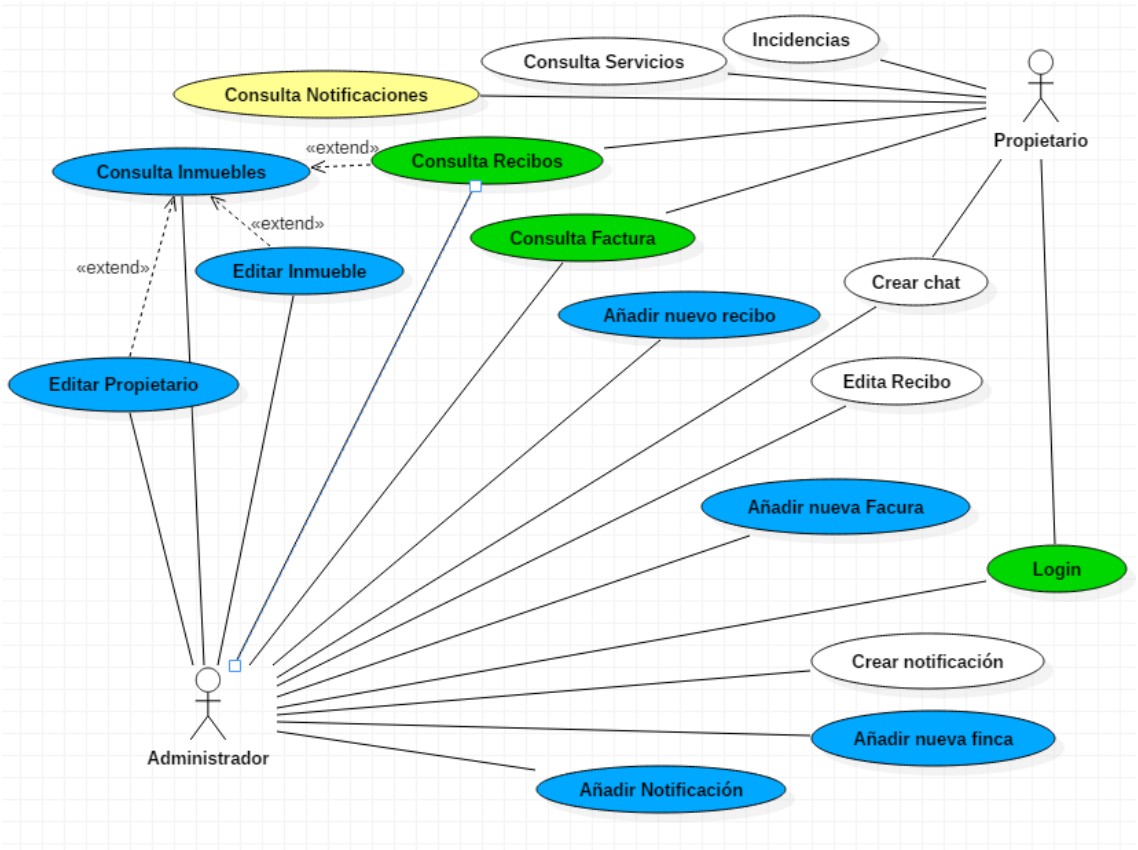


Figura 5. Diagrama de casos de uso estructurado del ciclo.

En base al diagrama de casos de uso del ciclo, se ha creado la especificación expandida de cada uno de estos casos de usos, como se verá a continuación.

Login
Actores: administrador y propietario
Precondiciones: Ninguna
Secuencia de eventos típica **Acción de un actor:** 1. Los actores entran a la página de login. 3. Los actores introducen 'usuario' y 'contraseña'. **Acción del sistema:** 2. El sistema presenta un formulario de login. 4. El sistema va a la página principal mostrando el nombre del usuario que ha entrado y su foto.
Secuencias alternativas Acción 3: Si se introduce mal el usuario o la contraseña, el sistema presenta un mensaje de error indicando la causa y vuelve a 2.
Postcondiciones: El usuario de la sesión queda autentificado ya sea como administrador o propietario en el dominio de seguridad de la aplicación.

Tabla 1. Especificación expandida de 'login'.

Consulta Inmuebles
Actores: administrador
Precondiciones: 1. El administrador ha hecho el login.
Secuencia de eventos típica **Acción de un actor:** 1. El administrador selecciona una finca en concreto. **Acción del sistema:** 2. El sistema presenta un listado con todos los inmuebles correspondientes a esa finca. <Editar inmueble> <Editar propietario> <Ver recibos>
Secuencias alternativas Ninguna
Postcondiciones: Ninguna

Tabla 2. Especificación expandida de 'consulta inmueble'.

Editar Inmueble
Actores: administrador
Precondiciones: 1. El administrador ha seleccionado una finca en concreto o la opción 'inmueble'. 2. El administrador ha hecho consulta inmuebles.
Secuencia de eventos típica **Acción de un actor:** 1. El administrador selecciona 'editar inmueble' del inmueble que desee. 3. El administrador cambia los campos que desee. **Acción del sistema:** 2. El sistema abre una ventana para poder editar los campos del inmueble. 4. El sistema registra el nuevo cambio del inmueble y vuelve al listado de inmuebles.
Secuencias alternativas Acción 4: Si los datos del inmueble no pasan la validación, se vuelve al punto 2 advirtiendo al usuario de que se ha producido un error.
Postcondiciones: El nuevo cambio del inmueble queda registrado en el sistema.

Tabla 3. Especificación expandida de 'editar inmueble'.

Editar Propietario
Actores: administrador
Precondiciones: 1. El administrador ha seleccionado una finca en concreto. 2. El administrador ha hecho consulta inmuebles.
Secuencia de eventos típica **Acción de un actor:** 1. El administrador pincha sobre el nombre del propietario que desee en la lista de inmuebles. 3. El administrador cambia los campos que desee. **Acción del sistema:** 2. El sistema abre una ventana para poder editar los campos del propietario. 4. El sistema registra el nuevo cambio del propietario y vuelve al listado de inmuebles.
Secuencias alternativas Acción 4: Si los datos del propietario no pasan la validación, se vuelve al punto 2 advirtiendo al usuario de que se ha producido un error.
Postcondiciones: El nuevo cambio del propietario queda registrado en el sistema.

Tabla 4. Especificación expandida de 'editar propietario'.

Consulta Recibo
Actores: administrador y propietario.
Precondiciones: 1. El administrador y propietario han hecho el login. 2. El administrador y propietario han seleccionado una finca.
Secuencia de eventos típica **Acción de un actor:** 1. El actor selecciona la opción 'recibos'. 3. El actor rellena los parámetros de búsqueda que desee. **Acción del sistema:** 2. El sistema muestra un formulario de búsqueda pudiendo elegir por el inmueble, por fecha o por los recibos que estén pendientes de pago. 4. El sistema muestra la información de los recibos que cumplen con los criterios de búsqueda definidos.
Secuencias alternativas Ninguna
Postcondiciones: Ninguna

Tabla 5. Especificación expandida de 'consulta recibo'.

Añadir Recibo
Actores: administrador
Precondiciones: 1. El administrador ha hecho login. 2. El administrador ha seleccionado la finca que desea administrar.
Secuencia de eventos típica **Acción de un actor:** 1. El administrador selecciona la opción 'recibos'. 3. El administrador selecciona en el submenú, la opción 'nuevo'. 5. El administrador rellena los campos del nuevo recibo y le da a 'generar recibo'. **Acción del sistema:** 2. El sistema presenta la página de recibos. 4. El sistema presenta un formulario con los campos necesarios para dar de alta un nuevo recibo relacionado al inmueble que se elija. 6. El sistema registra el nuevo recibo, lo marca como pendiente y muestra un mensaje de confirmación.
Secuencias alternativas Acción 6: Si los datos del recibo no pasan la validación, se vuelve al punto 4 advirtiendo al usuario de que se ha producido un error.
Postcondiciones: El recibo queda registrado.

Tabla 6. Especificación expandida de 'añadir recibo'.

Consulta Factura
Actores: administrador y propietario
Precondiciones: 1. El administrador y propietario han hecho el login. 2. El administrador y propietario han seleccionado una finca.
Secuencia de eventos típica **Acción de un actor:** 1. El actor selecciona la opción 'factura'. 3. El actor rellena los parámetros de búsqueda que desee. **Acción del sistema:** 2. El sistema muestra un formulario de búsqueda pudiendo elegir por proveedor, fecha o por facturas que estén pendientes de pago. 4. El sistema muestra la información de los recibos que cumplen con los criterios de búsqueda definidos.
Secuencias alternativas Ninguna
Postcondiciones: Ninguna

Tabla 7. Especificación expandida de 'consulta factura'.

Añadir Factura
Actores: administrador
Precondiciones: 3. El administrador ha hecho login. 4. El administrador ha seleccionado la opción 'factura'.
Secuencia de eventos típica **Acción de un actor:** 1. El administrador selecciona la opción 'nueva factura'. 3. El administrador rellena los campos de la factura. **Acción del sistema:** 2. El sistema presenta un formulario con los campos de la factura. 4. El sistema registra la nueva factura y muestra un mensaje de confirmación.
Secuencias alternativas Acción 4: Si los datos de la factura no pasan la validación, se vuelve al punto 2 advirtiendo al usuario de que se ha producido un error.
Postcondiciones: La factura queda registrada.

Tabla 8. Especificación expandida de 'añadir factura'.

Añadir Finca
Actores: administrador
Precondiciones: 5. El administrador ha hecho login.
Secuencia de eventos típica **Acción de un actor:** 1. El administrador selecciona la opción 'nueva finca'. 3. El administrador rellena los campos de la finca. **Acción del sistema:** 2. El sistema presenta un formulario con los campos de la finca. 4. El sistema registra la nueva finca y muestra un mensaje de confirmación.
Secuencias alternativas Acción 4: Si los datos de la finca no pasan la validación, se vuelve al punto 2 advirtiendo al usuario de que se ha producido un error.
Postcondiciones: La finca queda registrada.

Tabla 9. Especificación expandida de 'añadir finca'.

Añadir Notificación
Actores: administrador
Precondiciones: 1. El administrador ha hecho login. 2. El administrador ha seleccionado la opción 'notificación'.
Secuencia de eventos típica **Acción de un actor:** 1. El administrador selecciona la opción 'nueva notificación'. 3. El administrador rellena los campos de la notificación. **Acción del sistema:** 2. El sistema presenta un formulario con los campos de la notificación. 4. El sistema registra la nueva notificación y muestra un mensaje de confirmación.
Secuencias alternativas Acción 4: Si los datos de la notificación no pasan la validación, se vuelve al punto 2 advirtiendo al usuario de que se ha producido un error.
Postcondiciones: La notificación queda registrada.

Tabla 10. Especificación expandida de 'añadir notificación'.

Consulta Notificaciones
Actores: administrador y propietario
Precondiciones: 1. El administrador y propietario han hecho el login.
Secuencia de eventos típica **Acción de un actor:** 1. El actor selecciona la opción 'notificaciones'. 3. El actor selecciona una notificación. **Acción del sistema:** 2. El sistema muestra el listado de notificaciones leídas y pendientes por leer. 4. El sistema muestra los detalles de la notificación.
Secuencias alternativas Ninguna.
Postcondiciones: La notificación queda registrada como leída.

Tabla 11. Especificación expandida de 'consulta notificaciones'.

Otra tarea es identificar las clases del análisis como se detalla a continuación:

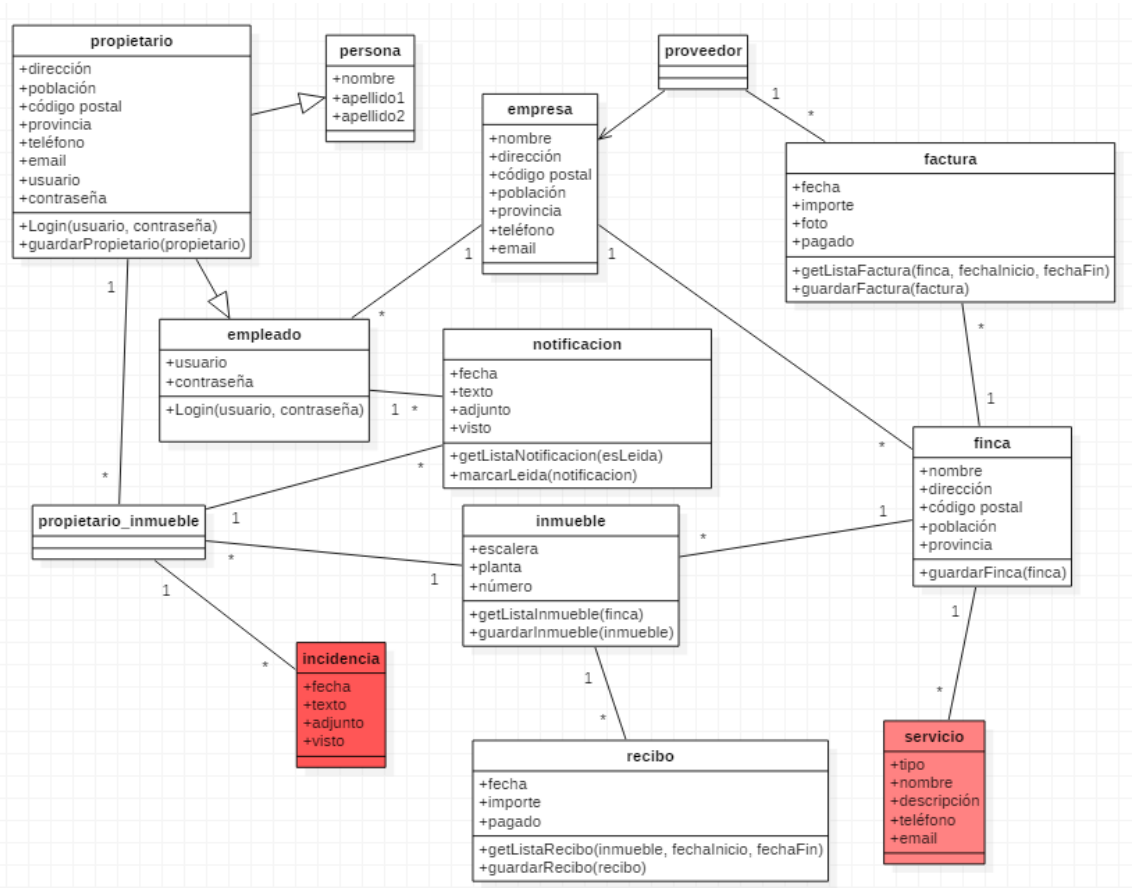


Figura 6. Diagrama de clases del análisis.

4.2 Diseño

El diseño de la aplicación web y móvil recogido en este trabajo pretende en primer lugar abstraer aquella parte de la aplicación que pueda resultar en funcionalidades comunes a las dos aplicaciones, lo que nos permitirá la reutilización del mayor número posible de código: la lógica que accede a la base de datos será común para las dos partes. Es decir, un mismo método de la lógica podrá ser llamado desde la parte móvil o de la web.

4.2.1 Esquema por capas

A continuación se describe como sería el esquema en el que se basa la funcionalidad de este proyecto.

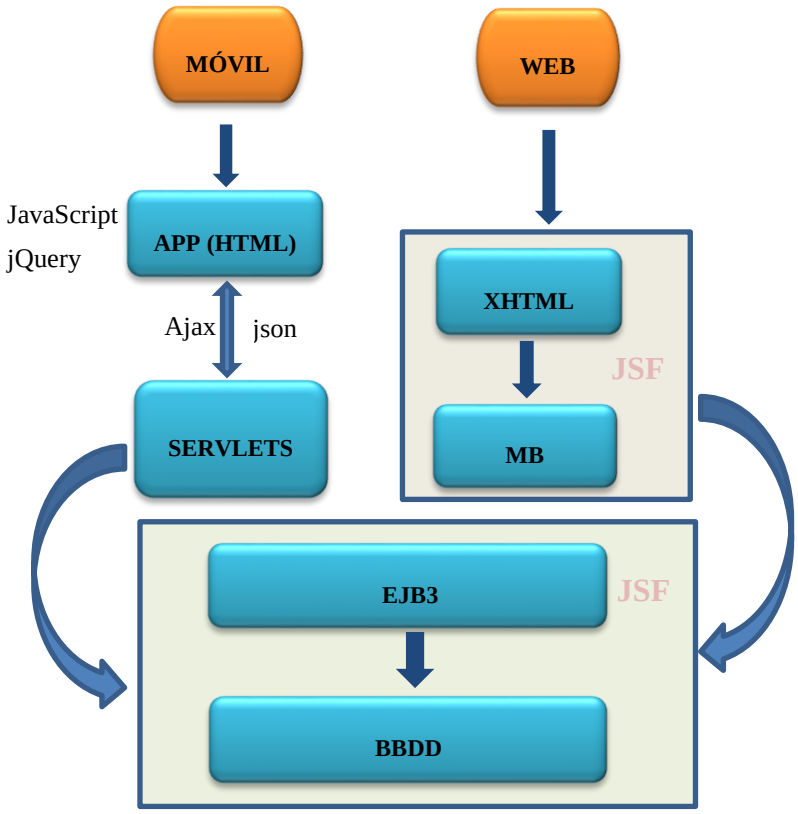


Figura 7. Esquema de las diferentes capas del proyecto e-vecinos.

Estos componentes y demás tecnologías implicados en este esquema, se irán estudiando con más detalle a lo largo de este capítulo.

4.2.2 Diseño de la base de datos

El diseño de las bases de datos es uno de los procesos más importantes en cualquier proyecto, teniendo que dedicarle siempre muchas horas hasta encontrar el diseño más eficiente y optimizado para la aplicación o sitio web que se vaya a programar. Por ello, se ha optado por la utilización de la herramienta "DBDesigner4" para la realización del diseño de la base de datos de este trabajo.

El DBDesigner4 es un sistema totalmente visual de diseño, modelado, creación y mantenimiento de bases de datos, que combina características y funciones profesionales con un diseño simple, muy claro y fácil de usar, a fin de ofrecer un método efectivo para gestionar bases de datos. Actualmente se encuentra en la versión 4, de licencia libre (GNU) y que puede ser descargado desde su web oficial. [4].

Esta herramienta está disponible para los sistemas operativos de Windows y Linux. Además, es desarrollada y optimizada para quienes utilizan MySQL como motor de base de datos.

Tras descargar e instalar DBDesigner4 en el ordenador de trabajo, se ha podido comprobar que esta completa aplicación incluye todas las herramientas que se necesitan a la hora de trabajar con cualquier base de datos.

Se pueden establecer relaciones entre las tablas, teniendo en cuenta la cardinalidad de 1:1 / 1:n (1:*) y n:n.

El resultado del diseño de la base de datos de la cual se parte para este trabajo se observa en el siguiente diagrama, en él se recoge todas las tablas y relaciones necesarias para el correcto funcionamiento de las aplicaciones a desarrollar.

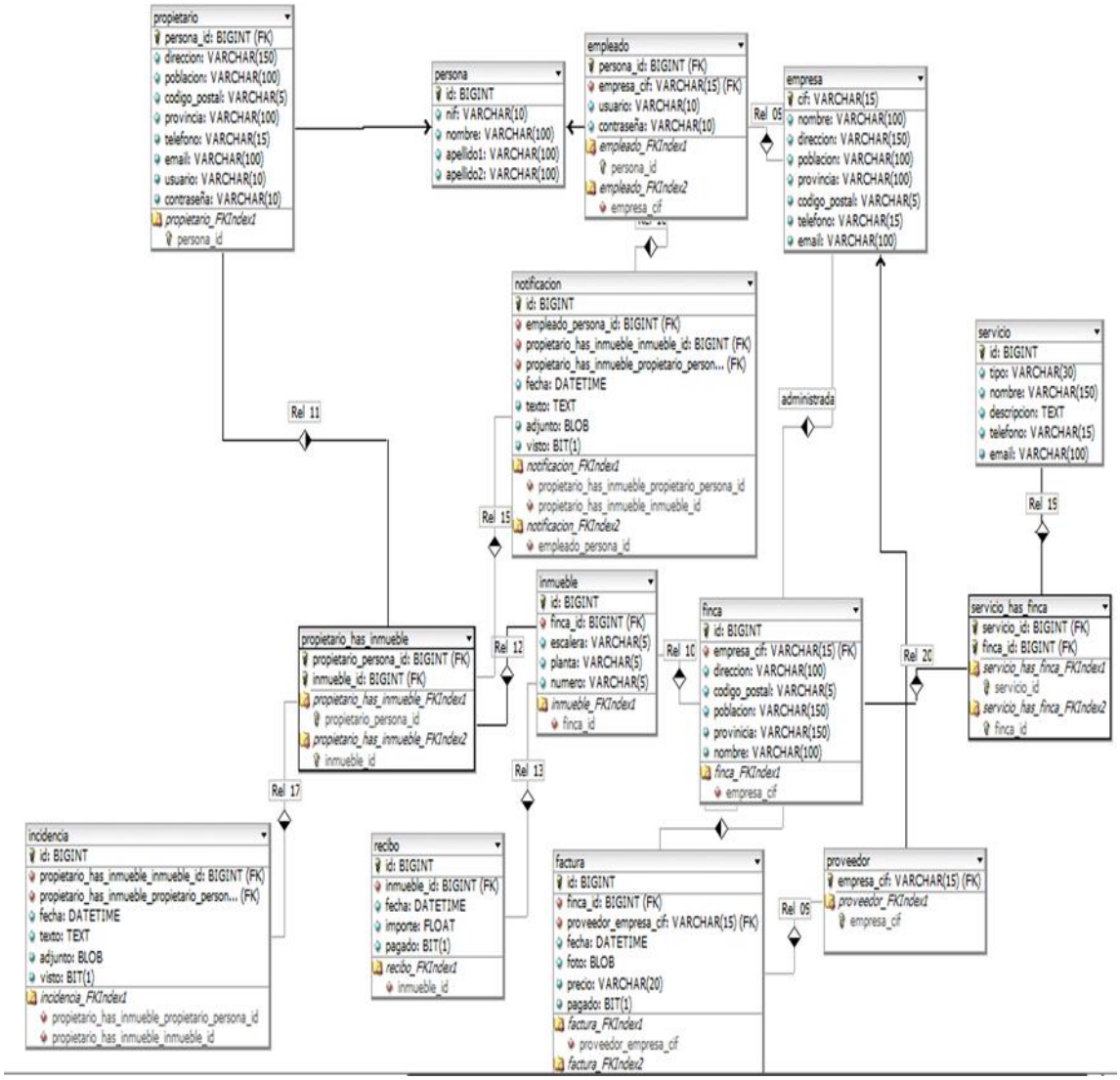


Figura 8. Modelo Entidad-Relación.

Una vez diseñada la base de datos, hay algo muy importante que ofrece esta herramienta y es el poder crear un script SQL, para luego ser ejecutado en el MySQL.

Esto se logra de la siguiente manera:

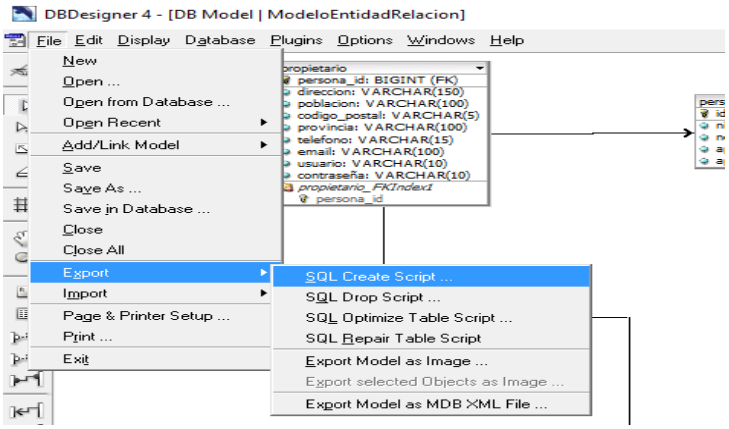


Figura 9. Ejemplo de cómo exportar un Script para SQL.

Por otro lado, existe la herramienta "SQLyog". Esto es un gran GUI (interfaz gráfica de usuario) desarrollado por Webyog, Inc. SQLyog permite trabajar gráficamente para MySQL, y también utiliza código SQL de una manera entendible y ordenada [6].

En MySQL, su herramienta de trabajo es una consola, por lo que para trabajar visualmente mejor se debe utilizar una herramienta externa, existen varios programas gratuitos, pero para este trabajo se ha hecho uso de esta herramienta por su previo conocimiento y fácil uso. Funciona en la plataforma Windows y Linux. Utiliza MySQL C API para comunicarse con los servidores MySQL. Sin dependencias de 'capas de abstracción de base de datos' (como ODBC / JDBC) [8]. Una de sus grandes ventajas es el ahorro de tiempo al escribir consultas con la comprobación de sintaxis.

Tras la instalación del programa, se configura la conexión a MySQL, como se muestra a continuación:

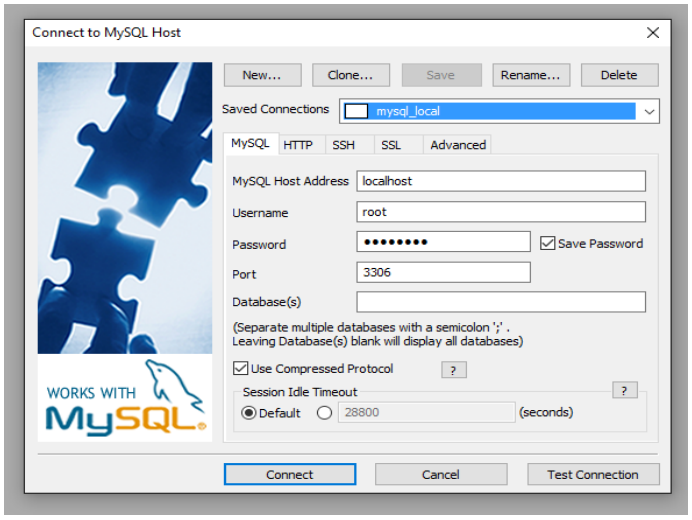


Figura 10. Conexión a la base de datos MySQL.

En el host Address se puede cambiar de dirección si el servidor esta en otro lugar o red, sino se deja por defecto.

Capítulo 5. Desarrollo y resultados

En este capítulo se expone el desarrollo y el resultado final de cada tarea realizada para obtener la aplicación web y móvil para e-vecinos.

5.1 IDE utilizado: Netbeans

NetBeans es un entorno de desarrollo integrado libre y gratuito sin restricciones de uso, de gran éxito y con una gran base de usuarios. Hecho principalmente para el lenguaje de programación Java, el cual no requiere un SDK adicional. La plataforma NetBeans permite que las aplicaciones sean desarrolladas a partir de un conjunto de componentes de software llamados *módulos*. Un *módulo* es un archivo Java que contiene clases de java escritas para interactuar con las API de NetBeans y un archivo especial (manifest file) que lo identifica como módulo [9].

El NetBeans IDE soporta el desarrollo de todos los tipos de aplicación Java (J2SE, web, EJB y aplicaciones móviles).

La plataforma ofrece servicios reusables comunes para las aplicaciones de escritorio, permitiendo a los desarrolladores centrarse en la lógica de sus aplicaciones. Algunas de las características de la aplicación son:

- Gestión de la interfaz de usuario (menús y barras de herramientas)
- Gestión de configuración de usuario
- Gestión de almacenamiento (guardar o cargar algún tipo de dato)
- Gestión de ventana
- Marco Asistente (soporta diálogos para a paso)
- Librería visual de Netbeans
- Herramientas de desarrollo integrado

5.2 Aplicación web

Como es bien sabido Java es uno de los lenguajes más populares para la construcción de *back-ends* para aplicaciones web. Con Java y *framewoks* basados en él, los desarrolladores web pueden crear aplicaciones web escalables para un gran número de usuarios. Java también es uno de los lenguajes principales a la hora de desarrollar aplicaciones nativas de Android para los diferentes dispositivos más usados actualmente como son teléfonos inteligentes y tabletas.

Además de esto, para la realización de este TFG, se han utilizado las siguientes herramientas que estudiaremos a lo largo de todo el desarrollo del mismo:

- HTML: ***HyperText Markup Language*** (lenguaje de marcas de hipertexto). Hace referencia al lenguaje de marcado para la elaboración de páginas web. Define una estructura básica y un código (denominado código HTML) para la definición de contenido de una página web, como texto, imágenes, videos, juegos, entre otros. Se considera el lenguaje web más importante siendo su invención crucial en la aparición, desarrollo y expansión de la World Wide Web (WWW). Es el estándar que se ha impuesto en la visualización de páginas web y es el que todos los navegadores actuales han adoptado. [11]
- XHTML: **eXtensible Hypertext Markup Language** (lenguaje de marcas de hipertexto extensible). Es básicamente HTML expresado como XML válido. Es más estricto a nivel técnico, pero esto permite que posteriormente sea más fácil al hacer cambios o buscar errores, siendo aconsejable para la modelación de páginas web. [12]

- CSS: ***Hoja de estilo en cascada o CSS***. Es un lenguaje usado para definir y crear la presentación de un documento escrito en HTML, XML o XHTML. Es decir, es un mecanismo simple que describe cómo se va a mostrar un documento en la pantalla, o cómo se va a imprimir, o incluso cómo va a ser pronunciada la información presente en ese documento a través de un dispositivo de lectura. Esta forma de descripción de estilos ofrece a los desarrolladores el control total sobre estilo y formato de sus documentos. [13].

- JavaScript: ***JS*** es un lenguaje de programación interpretado. Se utiliza principalmente en su forma del lado del cliente (client-side), implementado como parte de un navegador web permitiendo mejoras en la interfaz de usuario y páginas web dinámicas. Todos los navegadores modernos interpretan el código JavaScript integrado en las páginas web. Para interactuar con una página web se provee al lenguaje JavaScript de una implementación del Document Object Model (DOM) [14].

  Tradicionalmente se venía utilizando en páginas web HTML para realizar operaciones y únicamente en la parte del cliente, sin acceso a funciones del servidor. No obstante, a partir de mediados de la década de los 2000, ha habido una proliferación de implementaciones de JavaScript para el lado servidor, surgiendo así "Node.js". Actualmente JavaScript es ampliamente utilizado para enviar y recibir información del servidor junto con ayuda de otras tecnologías como AJAX.

HTML	CSS	JavaScript
Para definir el contenido de las páginas web	Para especificar el diseño de las páginas web	Para programar el comportamiento de las páginas web

Tabla 12. Resumen de las herramientas de trabajo.

5.2.1 Paradigma MVC – Modelo, Vistas y Controlador

Para este proyecto se aplica el modelo MVC. Es un patrón de arquitectura de software que separa los datos de una aplicación, la interfaz de usuario, y la lógica de control. Sus componentes son:

- Vista: Presenta el modelo en un formato adecuado para interactuar con el usuario (HTML, XHTML, JavaServer Pages, etc).

- Modelo: Es la representación de la información con la cual el sistema opera. Por tanto, gestiona todos los accesos a dicha información, tanto consultas como actualizaciones. Envía a la 'vista' aquella parte de la información que en cada momento se le solicita para que sea mostrada (típicamente a un usuario). Las peticiones de acceso o manipulación de información llegan al 'modelo' a través del 'controlador'. Engloba por tanto, los sistemas de Gestión de Base de Datos, la lógica de negocio, etc.

- Controlador: Responsable de recibir los eventos de entrada desde la vista e invoca peticiones al 'modelo' cuando se hace alguna solicitud sobre la información (por ejemplo, editar un documento o un registro en una base de datos). Así pues, el controlador es un intermediario entre la 'vista' y el 'modelo'.

Aunque originalmente MVC fue desarrollado para aplicaciones de escritorio, ha sido ampliamente adaptado como arquitectura para diseñar e implementar aplicaciones web en los principales lenguajes de programación. Se han desarrollado multitud de *frameworks* que implementan este patrón. Estos *frameworks* se diferencian básicamente en la interpretación de como las funciones MVC se dividen entre cliente y servidor. Cómo las tecnologías web han madurado, ahora existen *frameworks* como JavaScriptMVC, Backbone o jQuery que permiten que ciertos componentes MVC se ejecuten parcial o totalmente en el cliente (AJAX).

5.2.2 Elección del Framework J2EE

Antes de estudiar y elegir la herramienta de trabajo, se va a definir qué es y para qué sirve un *framework*, ya que conociendo este tipo de herramientas, será mucho más fácil desarrollar una aplicación como la expuesta en este proyecto.

Un *framework* define en términos generales, un conjunto estandarizado de conceptos, prácticas y criterios para enfocar un tipo de problemática particular que sirve como referencia, para enfrentar y resolver nuevos problemas de índole similar [15].

Más coloquialmente, un *framework* es un soporte de programas, bibliotecas y un lenguaje interpretado que nos permite desarrollar y unir de una manera más fácil y eficiente los diferentes componentes de un proyecto. Es decir, es sólo la herramienta base para construir una aplicación.

Entre las ventajas que podemos destacar al trabajar con un *framework*, es que es una herramienta común a muchos otros programadores y por tanto es posible beneficiarse de utilidades ya desarrolladas por estos, y que ya están adaptadas al propio *framework*.

Como se ha mencionado anteriormente, Java es una de las plataformas más extendidas en el entorno corporativo. Es una tecnología muy madura y popular que cuenta con innumerables herramientas de todo tipo. Para el desarrollo de aplicaciones de negocio se utiliza frecuentemente el patrón de diseño MVC (Modelo Vista Controlador) ya estudiado y que además, es sencillo de implementar en las aplicaciones web.

Existen muchos *frameworks* MVC para Java destacados, como por ejemplo:

- Spring MVC
- Struts (1.x y 2.x)
- GWT (Google web toolkit)
- JSF
- Aurora

Una vez sabemos qué es y para qué nos va a servir el *framework*, se va a indicar cuales son las preferencias que se tienen a la hora de elegir entre los posibles *frameworks* existentes para el desarrollo de esta aplicación y para ello se va a tener en cuenta los siguientes aspectos:

- Experiencia en el uso del framework, ya que esto reduce la curva de aprendizaje y nos ahorrará tiempo a la hora de trabajar con él porque no se tendrá que empezar desde cero.
- Que sean programas cuya licencia está basada en la licencia GNU GPL, es decir, software libre de carácter público, garantizando al usuario la utilización del mismo sin coste alguno.
- Disponibilidad de librerías y aplicaciones de terceros.
- Rendimiento y escalabilidad.

Tras una intensa búsqueda y el estudio entre los diferentes *frameworks* MVC para Java, se ha seleccionado el que cumple principalmente los puntos definidos anteriormente y el que más conviene para solución de esta primera parte del proyecto (aplicación web).

5.2.3 JavaServer Faces (JSF)

Es una tecnología y *framework* para aplicaciones Java basadas en web que simplifica el desarrollo de interfaz de usuario en aplicaciones Java JEE basadas en el patrón MVC. La tecnología JavaServer Faces incluye un conjunto de *API* para representar componentes de una interfaz de usuario y administrar su estado, manejar eventos, validar entrada, definir un esquema de

navegación de las páginas y dar soporte para internacionalización y accesibilidad. Además, incluye un modelo de eventos en el lado del servidor y un conjunto por defecto de componentes para la interfaz de usuario. [16] y [17].

JavaServer Faces define claramente una separación entre la lógica de aplicación y presentación, al tiempo que facilitan la conexión de la capa de presentación a el código de la aplicación.

Ventajas de usar el JSF:

- Forma parte del Java EE Standard.
- El código JSF con el que se crean las vistas es muy parecido al HTML estándar, conocido como XHTML. Lo pueden utilizar fácilmente desarrolladores y diseñadores web.
- Resuelve validaciones, conversiones, mensajes de error e internacionalización.
- Permite introducir JavaScript en la página, para acelerar la respuesta de la interfaz en el cliente (navegador del usuario).
- Muchos componentes.
- Muchas librerías y herramientas.
- Buen soporte por varios IDE.
- Alta demanda laboral.

Normalmente las aplicaciones web se construyen cómo un conjunto de pantallas con las que va interactuando el usuario. Estas pantallas contienen textos, botones, imágenes, tablas y elementos de selección que el usuario modifica. Todos estos elementos estarán agrupados en formularios HTML, que es la manera en que las páginas web envían la información introducida por el usuario al servidor. Por tanto, la principal función del controlador JSF es asociar a las pantallas, clases Java que recogen la información introducida y que disponen de métodos que responden a las acciones del usuario.

Las aplicaciones JSF están formadas por los siguientes elementos principales:

- Páginas que incluyen los formularios JSF. En este trabajo se usan páginas “xhtml”. Estas páginas generarán las vistas de la aplicación.
- Beans Java que se conectan con los formularios JSF.
- Clases Java para la lógica de negocio y utilidades.
- Ficheros de configuración, componentes a medida y otros elementos del *framework.*
- Resto de recursos de la aplicación web: recursos estáticos, JavaScript y otros elementos.

Para facilitar la visualización y recogida de los datos de los formularios, JSF introduce un lenguaje de expresiones (EL). Este lenguaje de expresiones permite acceder de manera muy sencilla a las propiedades de los Managed Beans.

A continuación vamos a detallar algunos de estos elementos que han formado parte del estudio de esta arquitectura.

5.2.3.1 *Managed Beans*

Managed Bean (MB) es una clase Java Bean regularmente registrada con JSF. El Bean gestionado contiene los métodos getter y setter, la lógica de negocio o incluso un Bean de respaldo [19].

Así pues los MB son el elemento principal en el que se incluyen los datos de registro del Bean. Será necesario añadir un bloque *<managed-bean>* por cada Bean que quiera ser gestionado por el *framework.*

EL MB trabaja con anotaciones "scope", Las anotaciones para el manejo de alcances JSF son las siguientes: @RequestScoped, @SessionScoped, @ViewScoped, @ApplicationScoped.

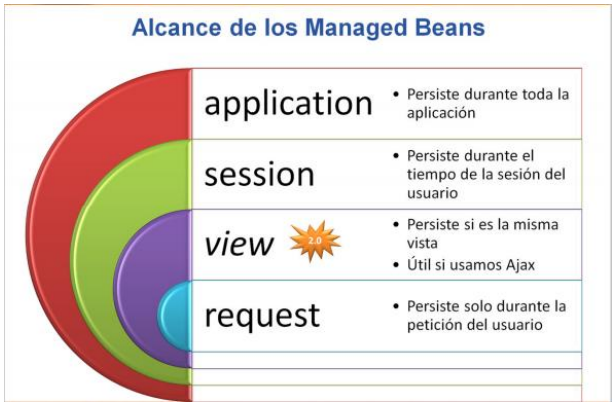


Figura 11. Enlaces de los MB.

Existen varias formas de declarar un Managed Bean en JSF. Aunque se recomienda, en caso de utilizar un contenedor JEE 6 (Java Enterprise Edition 6) utilizar la anotación de CDI [20].

CDI (Contexts and Dependency Inyection) ofrece un modelo más flexible para administrar los Beans, como parte de la especificación JEE 6. CDI especifica un mecanismo para inyectar Beans, interceptar, filtrar u observar eventos.

El paquete para las CDI es: *javax.enterprise.context.*

A continuación se muestra un pequeño ejemplo de una de las Managed Beans creada para el desarrollo de la aplicación web "e-vecinos". Esta MB se ha llamado "RecibosMB":

```
import java.util.List;
import javax.ejb.EJB;
import javax.inject.Named;

import javax.enterprise.context.SessionScoped;
import javax.inject.Inject;

/**
 *
 * @author niny
 */
@Named(value = "recibosMB")
@SessionScoped
public class RecibosMB implements Serializable {

    @EJB
    RecibosEJB recibosEJB;

    @Inject
    private SesionMB sesionMB;
```

Figura 12. Ejemplo de una Managed Bean - 'recibos.MB'.

Para la creación de las Managed Bean se ha hecho uso de las anotaciones:

- *@SessionScoped:* Las Managed Beans empleadas permanecen en la sesión ya que mantienen muy poca información, lo que no carga la memoria del servidor pero permite

el mantenimiento de la información de formularios de búsqueda y otros elementos creados durante la navegación de los usuarios: los datos del mismo usuario o la información de qué elemento del menú está activo.

- *@Named:* Dan un nombre a la Managed Bean de forma que luego pueda ser llamada desde los XHTML mediante EL.

También es interesante ver como se inyectan los EJB que forman parte de la lógica en los Managed Beans. Esta acción se realiza usando simplemente la anotación @EJB. De esta forma el servidor de aplicaciones se encargará de gestionar el acceso al EJB.

5.2.3.2 *Facelets y el uso de plantillas*

Facelets es un *framework* basado en el servidor que permite definir la estructura general de las páginas (su *layout*) mediante plantillas.

Facelets se adapta perfectamente al enfoque de JSF y se incorpora a la especificación en la última revisión 2.1. La estructura de la página se define utilizando las etiquetas Facelets y los componentes específicos que deben presentar los datos de la aplicación utilizando etiquetas JSF [21].

Entre las características de Facelets destacan:

- Definición de plantillas (como en Tiles)
- Composición de componentes
- Etiquetas para definir funciones y lógica
- Desarrollo de páginas amistoso para el diseñador
- Posibilidad de crear librerías de componentes

Para usar los tags de facelets, habrá que añadir la siguiente declaración en las páginas JSF:

xmlns:ui="http://java.sun.com/jsf/facelets"

La siguiente tabla explica brevemente las etiquetas definidas por Facelets que se han utilizado:

Etiqueta	Descripción
<ui:include>	Incluye contenido de otro fichero XHTML. Permite la reutilización de contenido para múltiples vistas.
<ui:composition>	Cuando se usa sin el atributo template una composición es una secuencia de elementos que se pueden insertar en algún otro lugar (mediante la etiqueta<ui:include> por ejemplo). La composición puede tener partes variables especificadas con el elemento hijo <ui:insert>. Cuando se usa con el atributotemplate, se carga la plantilla especificada. Los elementos hijos (etiquetas<ui:define>) determinan las partes variables de la plantilla. El contenido de la plantilla reemplaza esta etiqueta.
<ui:define>	Define el contenido que se inserta en una plantilla con un <ui:insert> que empareja

Tabla 13. Etiquetas Facelets.

Facelets nos permite usar un mecanismo de plantillas para encapsular los componentes comunes en una aplicación. Así también se podrá modificar el aspecto de una página aplicando cambios sobre la plantilla, y no individualmente sobre las páginas.

Para el proyecto e-vecinos, hemos hecho uso de un modelo de plantillas con sus respectivos CSS ya definidos [22]. De esto modo, se tiene una plantilla para la página de login definida en el fichero *templates/LoginTemplate.xhtml*, y una plantilla principal también definida en el fichero *templates/BasicTemplate.xhtml*, en donde el usuario siempre va a estar en la misma página y sólo va a cambiar el contenido del centro de la pantalla.

Además de esto, se ha hace uso de las hojas de estilos CSS descargadas y en las que se definen entre otras cosas, el color, el tamaño de fuente del título, botones, las dimensiones, separación del menú y la zona de página principal, etc.

5.2.3.3 *Vistas*

Las páginas de JSF se han creado con el lenguaje XHTML. Debido a que están escritas en XML, los documentos XHTML son más fáciles de validar y de procesar. Se pueden editar con herramientas genéricas orientadas a XML. Y se pueden transformar utilizando hojas de estilo y otras características de XML. Una de las características del XHTML es la posibilidad de utilizar en un mismo documento distintos lenguajes de etiquetas utilizando espacios de nombres.

Estos espacios de nombres se definen al comienzo del documento XHTML definiendo el prefijo que utilizan las etiquetas de cada uno de los lenguajes. Utilizando distintos prefijos para distintos lenguajes se elimina el problema de la posible ambigüedad a la hora de procesar el documento. Sólo es necesario declarar los que se utilizan en el documento.

El siguiente ejemplo muestra una cabecera de un documento de este proyecto “recibos.xhtml” en el que se declaran todos los espacios de nombres que se pueden utilizar.

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:ui="http://java.sun.com/jsf/facelets"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:f="http://java.sun.com/jsf/core"
      xmlns:c="http://xmlns.jcp.org/jsp/jstl/core">

    <ui:composition template="/WEB-INF/templates/BasicTemplate.xhtml">
        <ui:define name="content">
```

Figura 13. Ejemplo de cómo se declaran leguajes de etiquetas.

Los 4 primeros espacios de nombres son los que se utilizarían en cualquier proyecto de JSF y los que permiten múltiples funcionalidades. Las etiquetas más destacadas que se emplean en este proyecto son:

- ***ui:composition*:** permite tener una plantilla para diferentes ficheros XHTML donde se guarde la parte de las páginas que es común a todos ellos, por ejemplo cabecera, menú y pie de página.
- ***h:form, h:inputText, h:commandButton, h:commandLink,*** etc: definen elementos HTML, tanto elementos de formularios como botones o anclas. Permiten la comunicación con los Managed Beans tanto para pasarles información creada por el usuario como para obtener información de ellos que se mostrará al usuario.

Existe una librería de etiquetas que se utiliza normalmente en las páginas JSF (JavaServer Faces), como se ha mencionado antes, y cuyo prefijo y URI es el siguiente:

xmlns:c=”http://xmlns.jcp.org/jsp/jstl/core”

Estas etiquetas nos permiten las siguientes funcionalidades:

- **c:if:** permite la realización de operaciones condicionales, por ejemplo: <c:if test=”${usuario.nombre==”pepe”}>El usuario es pepe</c:if>.

- **c:foreach:** permite la realización de bucles, por ejemplo <c:foreach items="#{inmueblesMB.getListaInmuebles()}" var="inmueble">, itera sobre una lista de inmuebles y para acceder a la información de cada inmueble se usará la variable inmueble.

Como ejemplo se muestra la creación de la página "login.xhtml", en la que se pide al usuario el nombre y la contraseña. El código empleado queda así:

```
        xmlns:f="http://java.sun.com/jsf/core">

<ui:composition template="/WEB-INF/templates/LoginTemplate.xhtml">
        <ui:define name="content">
            <div class="header grey">
                <img src="img/lock.png" width="16" height="16" />
                <h3>Login</h3>
            </div>
            <h:form>
                <div class="content no-padding with-actions grey">
                    <div class="section _100">
                            <label><h:outputText value="Usuario"></h:outputText></label>
                            <div>
                                    <h:inputText value="#{loginMB.name}"></h:inputText>
                            </div>
                    </div>
                    <div class="section _100">
                            <label> <h:outputText value="Contraseña"></h:outputText> </label>
                            <div>
                                    <h:inputSecret value="#{loginMB.password}"></h:inputSecret>
                            </div>
                    </div>
                    <div class="actions">
                            <div class="actions-right">
                                    <h:commandButton value="Login" action="#{loginMB.login()}"></h:commandButton>
                            </div>
```

Figura 14. Código 'login.xhtml'.

Como puede apreciarse en el código empleado, se ha hace uso de JSF con 'xhtml'. En este caso, para crear el campo para la introducción del usuario y la contraseña, se utiliza el componente *< h:inputText>* y *<h:inputSecret>*. Posteriormente para poder vincularlo con el Managed Bean correspondiente, se utiliza *value="#{loginMB.name}"* y *value="#{loginMB.password}"*.

En el anexo se recoge información relacionada con las librerías de etiquetas de los distintos componentes que se usan.

5.2.3.4 *Fichero web.xml*

El motor de JSF es el *servlet*, que procesa las peticiones de los navegadores y realiza el procesamiento de las páginas JSF para generar el HTML resultante. Es necesario declarar este *servlet* en el fichero "web.xml" y definir ciertas propiedades de la aplicación web, como el mapeado de nombres o las páginas de bienvenida.

En concreto, el fichero WEB-INF/web.xml de la aplicación ejemplo es el siguiente:

```
<web-app version="3.1" xmlns="http://xmlns.jcp.org/xml/ns/javaee" xmlns:xsi="ht
    <context-param>
        <param-name>javax.faces.PROJECT_STAGE</param-name>
        <param-value>Production</param-value>
    </context-param>
    <servlet>
        <servlet-name>Faces Servlet</servlet-name>
        <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>Faces Servlet</servlet-name>
        <url-pattern>/faces/*</url-pattern>
    </servlet-mapping>
    <session-config>
        <session-timeout>
            30
        </session-timeout>
    </session-config>
    <welcome-file-list>
        <welcome-file>faces/inicio.xhtml</welcome-file>
    </welcome-file-list>
    <security-role>
        <description/>
        <role-name>clientes</role-name>
    </security-role>
</web-app>
```

Figura 15. Fichero 'web.xhtml'.

Podemos destacar los siguientes elementos:

- El servlet que procesa todas las peticiones es *javax.faces.webapp.FacesServlet*.
- Se definen dos patrones de nombres de petición que se redirigen a ese servlet: *.jsf y /faces/*.Cuando se realiza una petición (por ejemplo, */faces/login.xhtml* o*/login.jsf*) el *servlet* elimina el prefijo o la extensión *.jsf* y procesa la página XHTML definida en la ruta de la petición (*/login.xhtml*).

5.2.4 Lógica – EJB3

EJB es un modelo de programación que permite construir aplicaciones Java mediante objetos ligeros como POJO's (Plain Old Java Object). Cuando se construye una aplicación, son muchas las responsabilidades que se deben tener en cuenta, como la seguridad, transaccionalidad, concurrencia, etc. El estándar EJB permite centrarse en el código de la lógica de negocio del problema que se desea solucionar y deja el resto de responsabilidades al contenedor de aplicaciones donde se ejecutará la aplicación [23].

El hecho de estar basado en componentes permite que éstos sean flexibles y sobre todo reutilizables. Existen 3 tipos de EJB:

1. **EJB de Entidad (*Entity Beans*):** su objetivo es encapsular los objetos del lado del servidor que almacena los datos, es decir, los componentes *Entity Beans (EB)* son representaciones de información (en forma de POJO's) que es almacenada en una base de datos. El encargado de gestionar los EB es EntityManager. Los EJB de entidad presentan la característica fundamental de la persistencia.
2. **EJB de Sesión (*Session EJB*):** gestionan el flujo de la información en el servidor. Generalmente sirven a los clientes como una fachada de los servicios proporcionados por otros componentes disponibles en el servidor. Puede haber dos tipos, los con estados (*stateful*) y los sin estado (*stateless*).

3. **EJB Dirigidos por Mensajes (*Message-driven EJB*):** son los únicos Beans con funcionamiento asíncrono.

Los EJB de entidad se tratarán en el apartado de la persistencia. Aquí se ocupará de los EJB de sesión, ya que éstos son los componentes que contienen la lógica de negocio de la aplicación en desarrollo. Son accedidos a través de un proxy (también llamado *vista*) tras realizar una solicitud al contenedor. Tras dicha solicitud, el cliente obtiene una vista del *Session Bean*, pero no el *Session Bean* real. Esto permite al contenedor realizar ciertas operaciones sobre el Session Bean real de forma transparente para el cliente (como gestionar su ciclo de vida, solicitar una instancia a otro contenedor trabajando en paralelo, etc). Existen básicamente dos tipos:

- Stateless Session Beans: EJBs que no guardan información.
- Stateful Session Beans: EJBs que guardan información.

En el siguiente ejemplo se muestra un EJB para este proyecto, "RecibosEJB":

```
import javax.ejb.Stateless;
import javax.persistence.PersistenceContext;
import javax.persistence.TypedQuery;
import javax.persistence.EntityManager;

/**
 *
 * @author niny
 */
@Stateless
public class RecibosEJB {

    @PersistenceContext(unitName = "EvecinosPU")
    private EntityManager em;

    public List<Recibo> getListaRecibos(Propietario propietario, Long inmuebleId, Date fechaDesde, Date fechaHasta,
        HashMap<String,Object> parametros = new HashMap<String, Object>();
        String q = "SELECT r FROM Recibo r LEFT JOIN r.inmueble i LEFT JOIN i.propietarioHasInmuebleCollection hi "
                + "LEFT JOIN hi.propietario p WHERE" ;
        boolean conAnd = false;
```

Figura 16. Ejemplo de un EJB - 'recibosEJB'.

Como se observa, se utiliza el componente de sesión *@Stateless* (sesión sin estado). Estos componentes no requieren mantener un estado entre diferentes invocaciones. Un cliente debe asumir que diferentes solicitudes al contenedor de una misma sesión pueden devolver vistas a objetos diferentes. Por todo esto, estos componentes son creados y destruidos a discreción del contenedor, y puesto que no mantienen estado son muy eficientes a nivel de uso de memoria y recursos en el servidor. Toda la lógica de negocio de e-vecinos se encuentra programada con EJB de sesión sin estado. Además, EJB recoge un objeto *EntityManager* que le permite acceder a la capa de persistencia para realizar consultas y actualizaciones en la base de datos.

5.2.4.1 *JPA (API de Persistencia en Java)*

En Java a menudo se suelen solucionar problemas de negocio a través de objetos, los cuales tienen estado y comportamiento. Sin embargo, las bases de datos relacionales almacenan la información mediante tablas, filas, y columnas, de manera que para almacenar un objeto hay que realizar una correlación entre el sistema orientado a objetos de Java y el sistema relacional de la base de datos sobre la cual se trabaja. Por tanto, JPA es una abstracción sobre JDBC que nos permite realizar

dicha correlación de forma sencilla, realizando por parte del programador toda la conversión entre los objetos y las tablas de una base de datos. Esta conversión se llama ORM (Object Relational Mapping - Mapeo Relacional de Objetos), y puede configurarse a través de metadatos (mediante XML o anotaciones en las propias clases de entidad) [24]. Por supuesto, JPA también permite seguir el sentido inverso, creando objetos a partir de las tablas de una base de datos, y también de forma transparente. A estos objetos se les llama *entidades* (entities).

La persistencia de Java fue desarrollada por expertos de EJB 3.0, aunque su uso no se limita a los componentes software EJB se puede utilizar en aplicaciones web y aplicaciones clientes.

La API de persistencia de Java se basa en el uso de Entity Beans de EJB3. Básicamente cada Entity Bean representa los campos de cada una de las tablas de la base de datos. Para ello contiene los campos de cada tabla mediante una correspondencia entre tipos de datos de Java y de la base de datos.

Esta es una tabla que se muestra en la documentación de MySQL:

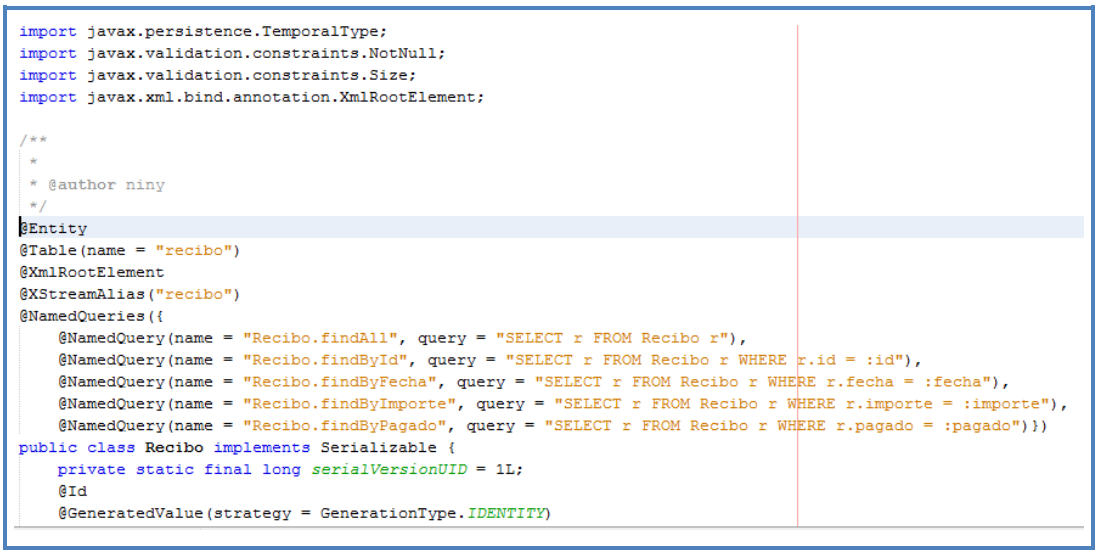
Java Data Type	MySQL Column Type
boolean, Boolean	BIT(1)
byte, Byte	BIT(1) to BIT(8), TINYINT
short, Short	BIT(1) to BIT(16), SMALLINT, YEAR
int, Integer	BIT(1) to BIT(32), INT
long, Long	BIT(1) to BIT(64), BIGINT, BIGINT UNSIGNED
float, Float	FLOAT
double, Double	DOUBLE
java.math.BigDecimal	NUMERIC, DECIMAL
java.math.BigInteger	NUMERIC (precision = 0), DECIMAL (precision = 0)

Tabla 14. Tipos de datos Java y correspondencia en MySQL.

Además se usan anotaciones que permiten definir elementos típicos de una base de datos. Estas son algunas de las utilizadas:

- *@Id*: el campo pertenece a la clave privada.
- *@NotNull*: el campo no puede ser nulo.
- *@Column*: indica el nombre del campo en la base de datos.
- *@Size*: se emplea sobretodo en cadenas de texto para indicar su tamaño máximo.
- *@OneToMany*: indica que el campo nace de una relación 1:n. Su valor es una lista de objetos del lado n de la relación.
- *@ManyToOne*: indica que el campo nace de una relación n:1. Su valor es un objeto Entity Bean del tipo del lado 1 de la relación.

A continuación se muestra como ejemplo una de las clases Entity Bean del proyecto:



```
import javax.persistence.TemporalType;
import javax.validation.constraints.NotNull;
import javax.validation.constraints.Size;
import javax.xml.bind.annotation.XmlRootElement;

/**
 *
 * @author niny
 */
@Entity
@Table(name = "recibo")
@XmlRootElement
@XStreamAlias("recibo")
@NamedQueries({
    @NamedQuery(name = "Recibo.findAll", query = "SELECT r FROM Recibo r"),
    @NamedQuery(name = "Recibo.findById", query = "SELECT r FROM Recibo r WHERE r.id = :id"),
    @NamedQuery(name = "Recibo.findByFecha", query = "SELECT r FROM Recibo r WHERE r.fecha = :fecha"),
    @NamedQuery(name = "Recibo.findByImporte", query = "SELECT r FROM Recibo r WHERE r.importe = :importe"),
    @NamedQuery(name = "Recibo.findByPagado", query = "SELECT r FROM Recibo r WHERE r.pagado = :pagado")})
public class Recibo implements Serializable {
    private static final long serialVersionUID = 1L;
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
```

Figura 17. Clase de entidad - 'recibo.java'.

Para llevar a cabo consultas y actualizaciones con estos Entity Bean, el servidor de aplicaciones ofrece un objeto de tipo: *javax.persistence.EntityManager*.

@PersistenceContext(unitName = "EvecinosPU")

private EntityManager em;

Ésta es la interfaz principal de JPA utilizada para la persistencia de las aplicaciones. Cada Entity Manager puede realizar operaciones CRUD (Create, Read, Update, Delete) sobre un conjunto de objetos persistentes. Es un objeto único, no compartido que representa una unidad de trabajo particular para el acceso a datos. Proporciona métodos para gestionar el ciclo de vida de las instancias entidad y para crear instancias Query.

La interfaz *javax.persistence.Query* está implementada por cada vendedor de JPA para encontrar objetos persistentes manejando cierto criterio de búsqueda. JPA estandariza el soporte para consultas utilizando Java Persistence Query Language (JPQL) y Structured Query Language (SQL). Podemos obtener una instancia de Query desde una instancia de un Entity Manager.

Los métodos de Query que se han utilizado en la aplicación son:

- *getResultList*: ejecuta una sentencia JPQL y devuelve una lista de un determinado tipo de Entity Bean.
- *merge*: método al que se le pasa un Entity Bean para que sea persistido en base de datos. Dependiendo del valor de los campos que forman la clave privada la persistencia realizará una inserción o una actualización.

Las unidades de persistencia se definen en el fichero de configuración persistence.xml. Aquí se muestra un ejemplo de este fichero:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.1" xmlns="http://xmlns.jcp.org/xml/ns/persistence" xmlns:xsi="http://www.w3.org/2001/XMLSchema-i
  <persistence-unit name="EvecinosPU" transaction-type="JTA">
    <provider>org.eclipse.persistence.jpa.PersistenceProvider</provider>
    <jta-data-source>EvecinosDS</jta-data-source>
    <exclude-unlisted-classes>false</exclude-unlisted-classes>
    <properties>
      <property name="javax.persistence.schema-generation.database.action" value="create"/>
    </properties>
  </persistence-unit>
</persistence>
```

Figura 18. Configuración del fichero 'persistencia.xml'.

Este archivo define una unidad de persistencia llamada “EvecinosPU”, con una fuente de datos que utiliza el contenedor también definido como “EvecinosDS”.

5.2.5 Implementación

Se resume todos los componentes JEE usados para el desarrollo web del proyecto de e-vecinos.

- Páginas JavaServer Faces (JSF), basadas en XHTML:

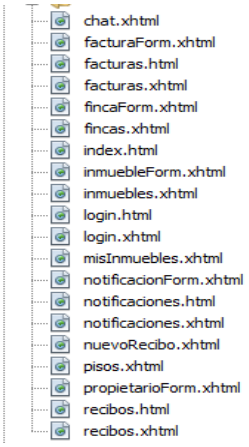


Figura 19. Páginas JSF.

- Entidades:

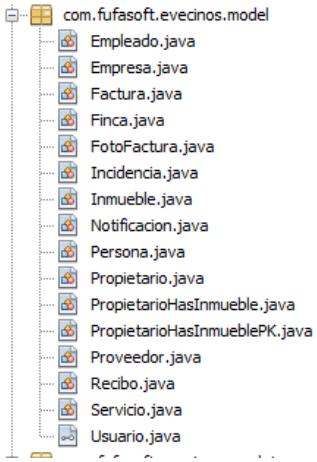


Figura 20. Entidades.

- Enterprise java beans (EJB):

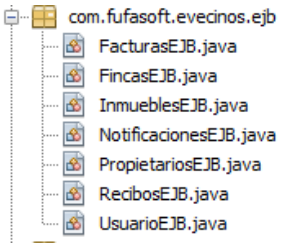


Figura 21. EJB

- Managed Beans (MB):

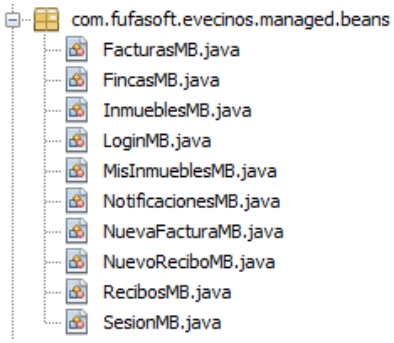


Figura 22. MB

5.2.6 GlassFish

GlassFish es un servidor de aplicaciones de código abierto realizado por Oracle. Su integración con Netbeans funciona muy bien y su instalación es muy sencilla ya que la descarga del servidor se puede realizar desde el mismo IDE [25].

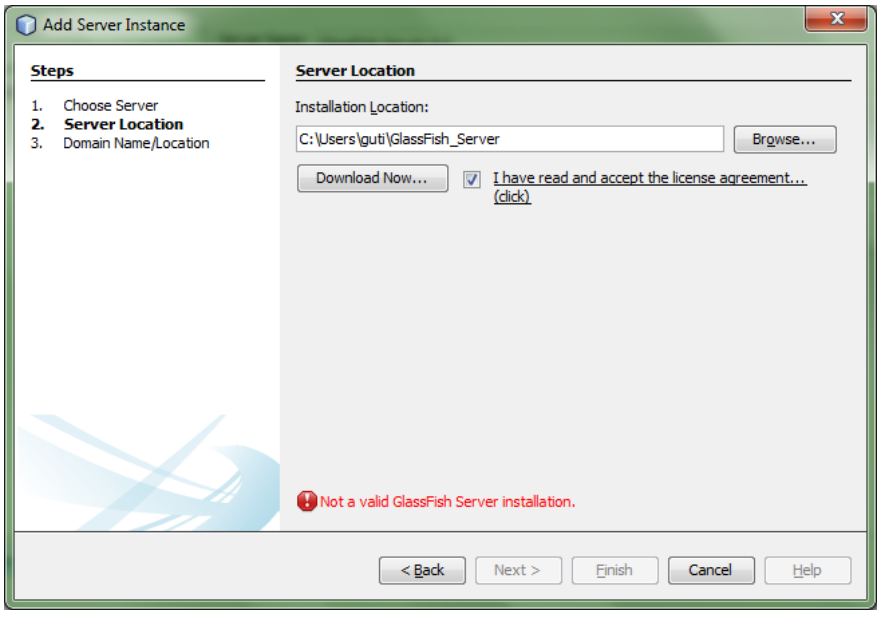


Figura 23. Paso 1 para añadir GlassFish.

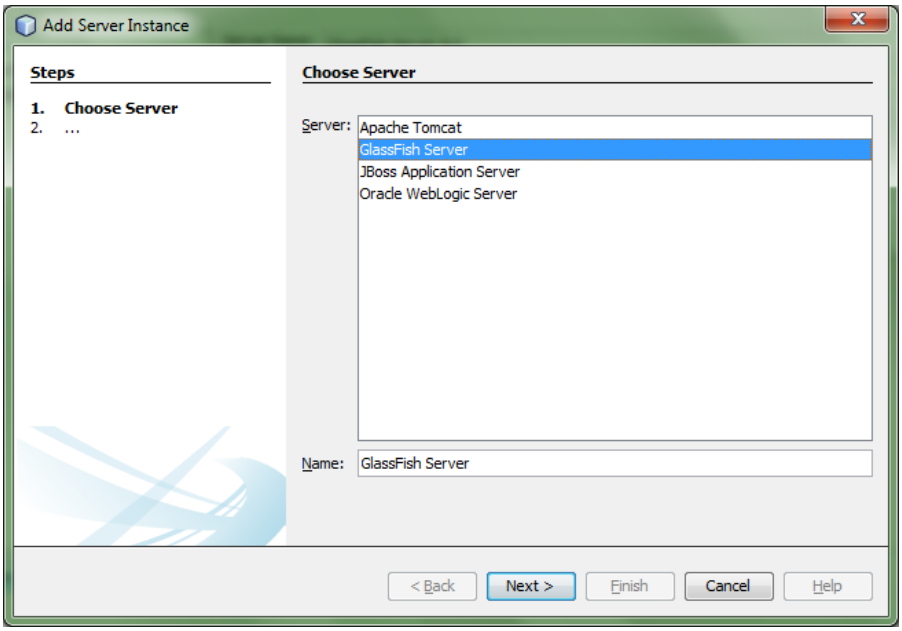


Figura 24. Paso 2 para añadir el GlassFish.

Como se puede ver en la figura en Netbeans basta con señalar que se desea integrar un servidor GlassFish y en la siguiente página pulsar sobre el botón 'Download Now' para que se descargue el servidor, lo instale y configure Netbeans para su integración.

A partir de ese momento al arrancar cualquier página XHTML de la aplicación arrancará el servidor GlassFish. Además Netbeans ofrece un elemento 'Server Resources' donde se pueden ver los ficheros de configuración de GlassFish.

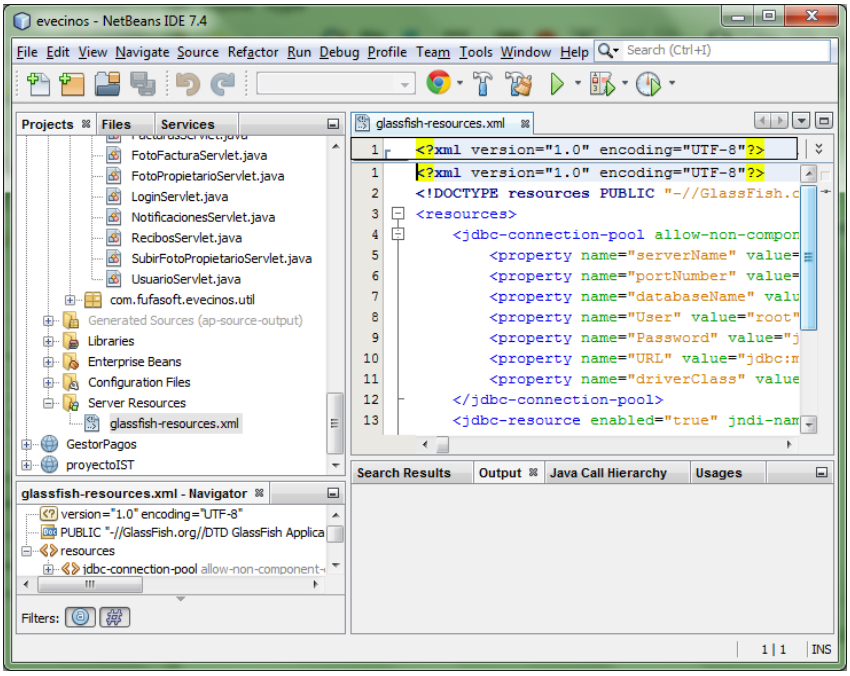


Figura 25. Fichero de configuración del GlassFish.

Concretamente se ha utilizado el fichero glassfish-resources.xml para definir el data-source del proyecto.

5.2.7 Resultado de la app web e-vecinos

Para acceder a la página de login que es la página de acceso a la aplicación web, se debe utilizar el prefijo faces accediendo a *http://localhost:8080/evecinos/faces/login.xhtml*, puesto que este ha sido un ejemplo hecho en local.

El aspecto de las páginas obtenido, usando las plantillas descritas anteriormente, es el que se muestra en las siguientes figuras:

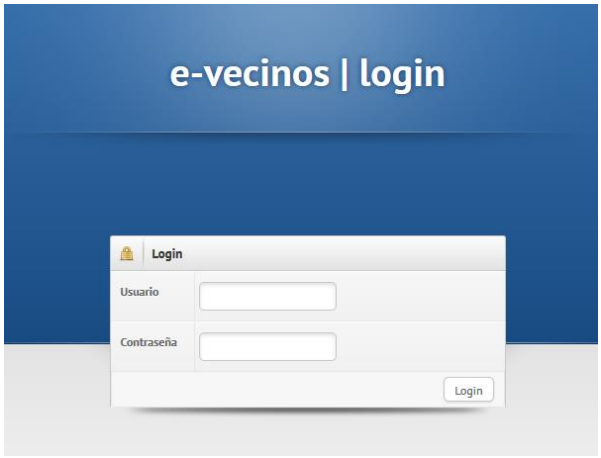


Figura 26. Página de login de la aplicación web e-vecinos.

Si se accede como propietario aparece la página de inicio definida para este usuario. Como se verá, se mantiene el diseño de la plantilla principal que hemos descrito antes, el cual se conforma de toda la parte de arriba, del menú para seleccionar las diferentes opciones que se tiene y del lateral de la izquierda con el perfil del usuario (nombre y foto).

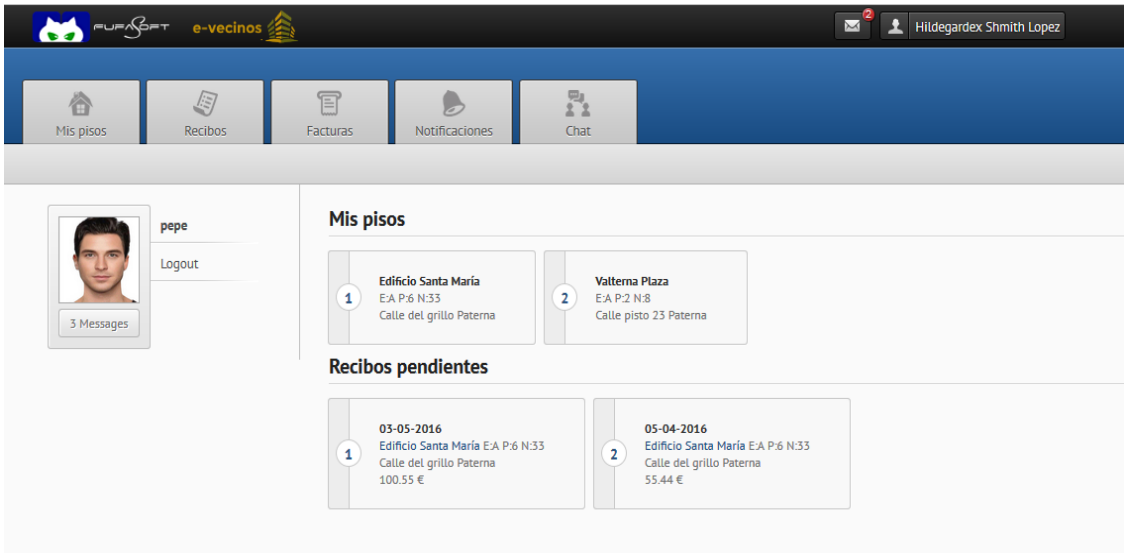


Figura 27. Página de inicio del propietario.

Al ir navegando entre las diferentes páginas que corresponden a cada pestaña, se verá que solo cambia el contenido de la parte central, manteniéndose activo la pestaña en la cual se encuentra.

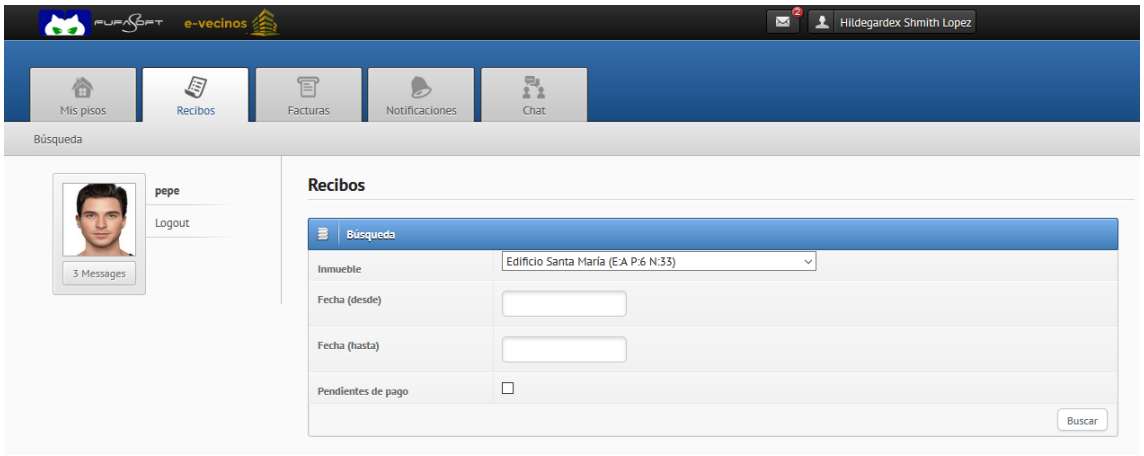


Figura 28. Página de Recibos usando la plantilla principal.

Un caso parecido ocurre si se accede como administrador, tras hacer el login aparece la página de inicio definida para este usuario.

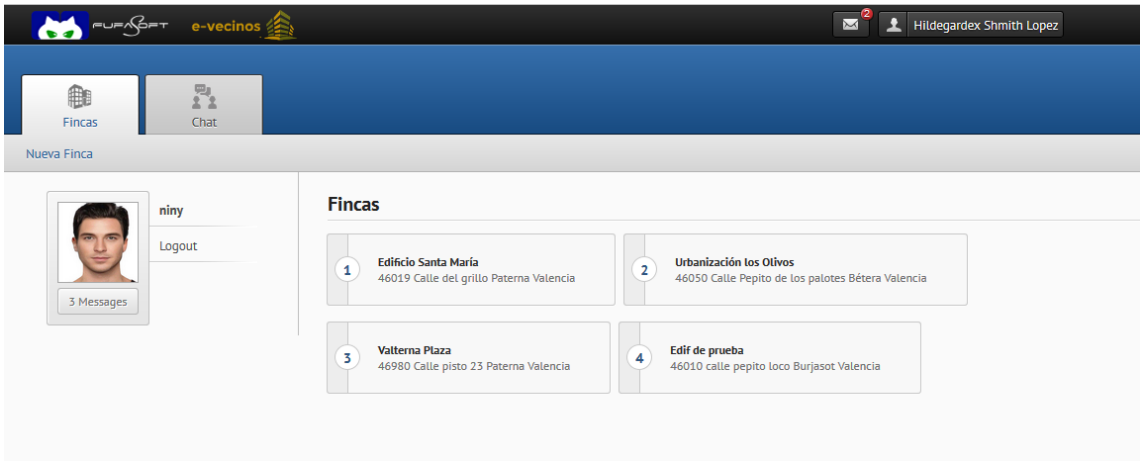


Figura 29. Página de inicio del administrador.

Cuando entra en una finca para gestionar, se mantiene también el diseño de la plantilla principal definida para este usuario con sus respectivas pestañas. Al ir navegando entre las diferentes páginas que corresponden a cada pestaña se verá que solo cambia el contenido de la parte central de la página como ocurre en el caso del propietario.

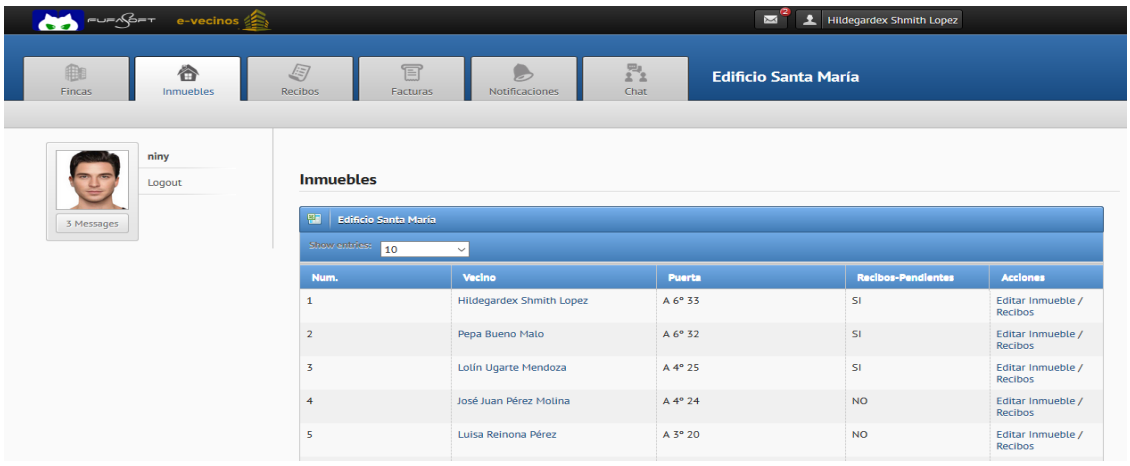


Figura 30. Página de inmuebles usando la plantilla principal.

5.3 Diseño web adaptativo

Responsive web design ó Diseño web adaptativo es una técnica de diseño y desarrollo que crea un sitio web y que se adapta al tamaño de la pantalla del usuario. Mejora la experiencia de navegación del usuario al ofrecer una página web adaptable y flexible, optimizada para el dispositivo que está utilizando.

5.4 Aplicación móvil multiplataforma

En esta segunda parte del proyecto se hablará del desarrollo y el resultado al realizar la aplicación móvil multiplataforma para e-vecinos.

5.4.1 Diferencias de las tecnologías nativas hibridas y web

Ahora vamos a centrarnos más en concreto en las aplicaciones que corren en cada uno de los dispositivos móviles. Hoy en día, en función de cómo se aborda su desarrollo, se suele hablar de tres tipos de aplicaciones móviles: Aplicaciones nativas, aplicaciones web y aplicaciones híbridas. Vamos a diferenciar cada una de ellas para entenderlas mejor.

1. *Aplicaciones Nativas*

   Este tipo de aplicaciones son aquellas que se desarrollan de forma específica para cada uno de los sistemas operativos existentes, permitiéndonos acceder a elementos del propio dispositivo, tales como la cámara, GPS, acelerómetro, actividad de la pantalla, entre otros. Esto hace que la experiencia del usuario sea mucho más positiva que con otro tipo de apps. Cada una de las plataformas existentes (Android, iOS o Windows Phone) tiene un sistema operativo diferente, con lo cual, si queremos que una aplicación sea utilizada en diferentes plataformas, habrá que desarrollarla para cada una de esas.

2. *Aplicaciones Web*

   Una aplicación web es una aplicación desarrollada con leguajes muy conocidos por los programadores, como HTML, CSS y JavaScript. Este tipo de aplicaciones no tiene la restricción de plataforma como ocurría en las aplicaciones nativas, ya que podemos programar independiente del sistema operativo en el que se usará la aplicación. De esta forma se pueden ejecutar en los diferentes dispositivos sin tener que crear varias aplicaciones. Este tipo de aplicaciones se ejecutan dentro del navegador web y el contenido se adapta al dispositivo simulando una aplicación cualquiera.

3. *Aplicaciones Híbridas*

   Las aplicaciones híbridas, son aplicaciones que contienen características de los dos tipos anteriores, se podría decir que recoge lo mejor de cada una de ellas. Las apps híbridas se desarrollan con lenguajes propios de una aplicación web, es decir, HTML, JavaScript y CSS por lo que permite su uso en diferentes plataformas, pero también dan la posibilidad de acceder a gran parte de las características del hardware del dispositivo. La principal ventaja es que a pesar de estar desarrollada con HTML, Java o CSS, es posible agrupar los códigos y distribuirla en app store.

Para hacer un pequeño análisis de la tendencia actual que se tiene sobre estas tres tecnologías, se hace una comparativa usando de nuevo la herramienta “Google Trends”.

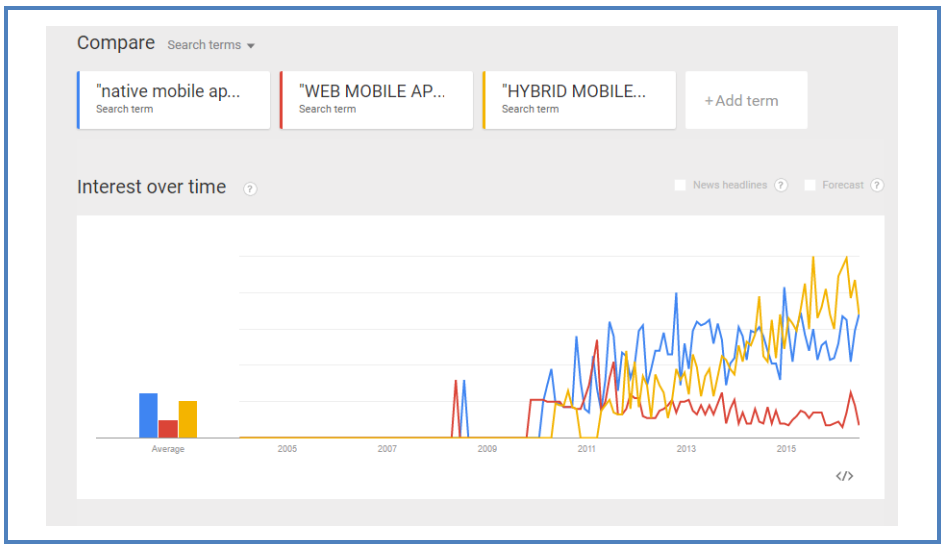


Figura 31. Tendencia del uso de las aplicaciones móviles.

El resultado de esta comparativa da una idea de cómo cada vez más se ha ido incrementando el uso de las aplicaciones hibridas en el mercado, siendo notorio el crecimiento en los últimos años. Se puede decir, que esto es debido al también crecimiento que se ha ido experimentado en el uso de los diferentes dispositivos móviles. Por lo que es más sencillo y rápido crear este tipo de aplicaciones con herramientas que te permiten acceder desde cualquier plataforma e incluso desde entornos de escritorio. Otro motivo por el cual estas aplicaciones resultan también más atractivas, es por el hecho de que al estar visibles en la app store, la promoción y la comercialización es inmediata y no debe realizarse de forma independiente como sería el caso de una app web.

Cada una de ellas tiene sus aspectos positivos y negativos que pueden y deben influir a la hora de hacer una elección para su desarrollo. La siguiente tabla pretende dar una idea general de las principales características para cada una de estas tecnologías.

Características	App Nativa	App Web	App Hibrida
Reutilización de código	No	Sí	Sí
Interfaz de usuario	Alta	Media	Baja
Coste desarrollo y mantenimiento	Elevado	Bajo	Bajo
Publicación en App Store	Sí	No permitido	Sí
Uso sin conexión de datos (sin conexión a Internet)	Sí	Parcial	Sí
Acceso al hardware del dispositivo	Completa	Parcial – alto	Muy limitado

Tabla 15. Comparativa entre las diferentes app móviles.

5.4.2 Tecnologías hibridas disponibles

El principal inconveniente de una aplicación híbrida es que su ejecución, desde la perspectiva del usuario, es más lenta y menos fluida que en el caso de una aplicación nativa. Además, para funcionalidades del sistema no utiliza las llamadas especificadas en la API de desarrollo de la plataforma sobre la que se ejecuta, porque ni siquiera utiliza el mismo lenguaje de programación. Por esto, en esta parte del desarrollo de la app móvil, para interactuar con el hardware del dispositivo se depende de que en el *framework* de programación en uso exista una librería o *plugin* que implemente la correspondiente llamada al sistema para la funcionalidad deseada. Por ejemplo, este proyecto requiere la cámara del propio dispositivo, la cual se obtendrá mediante una llamada a un *plugin*, que a su vez hará la llamada al sistema para realizar la acción de tomar una foto.

Existen en el mercado una gran diversidad de tecnologías que pueden ser útiles a la hora de elegir la que mejor se acople al desarrollo de esta aplicación, se estudiaran algunas de ellas para su posterior elección.

- **Córdova – PhoneGap:**

  Es un *framework* para desarrollo de aplicaciones móviles de tipo aplicaciones web. Ofrece una capa nativa, con plugins y librerías específicas que extienden las funcionalidades nativas del sistema, como gestión de la localización, estado de la batería, etc. Sobre ésta, se encapsula una instancia del motor de renderizado web. La lógica de negocio de la aplicación se programa como una aplicación web. Esto permite el desarrollo en cualquiera de las múltiples tecnologías de *front-end*. Los lenguajes más utilizados son HTML, CSS y JavaScript. Además se pueden utilizar otros lenguajes habituales en desarrollo de aplicaciones web. Córdova - PhoneGap es la versión *open source*, mantenida por Apache Foundation. PhoneGap es propiedad de Adobe y no se comercializa con licencia de software libre. Ofrece, no obstante, un set mayor de herramientas para facilitar la compilación, el testeo sobre dispositivo y la publicación de aplicaciones.

- **Xamarin:**

  Es una plataforma de desarrollo de aplicaciones móviles para plataformas Android, iOS y Windows Mobile. Xamarin utiliza el *framework* .NET de Microsoft. El lenguaje utilizado es C#. La ventaja de Xamarin es que permite compilar proyectos que trabajan de forma nativa sobre la plataforma escogida. Es, por tanto, una tecnología semi-híbrida.

  Su principal inconveniente es que se centra en un *framework* muy específico, exigiendo un conocimiento muy profundo del mismo para desarrollar una aplicación. Presenta, por tanto, una elevada deuda tecnológica a medio y largo plazo. No se puede afirmar que un desarrollo realizado en Xamarin sea rápido ni de bajo coste.

- **Enyo:**

  Al igual que pasaba con PhoneGap, Enyo cubre las necesidad básicas que requiere nuestra aplicación. Es de código libre, utiliza las mismas tecnologías para desarrollo que PhoneGap (HTML, CSS y JavaScript), y es soportado por las principales plataformas. También tiene en común con PhoneGap que este software ha sido diseñado para la creación de aplicaciones lo cual le da un valor añadido, aunque en este caso fue diseñado para aplicaciones web también permite aplicaciones de escritorio y dispositivos móviles.

  Una cosa muy valorable de Enyo es que soporta Web Services con JSON. Enyo como contrapartida, tiene algunas carencias. No soporta el sistema operativo Symbian, que aunque no sea de los más utilizados en la actualidad, fue si no el más utilizado cuando

salió al mercado. En cuanto al acceso al hardware de los dispositivos, solo permite acceder a la geolocalización.

- **Rhodes:**

  El *framework* Rhodes, cumple, en parte, todos los criterios que se podría requerir en este trabajo, aunque con algunas diferencias respecto a los otros. La licencia de Rhodes no es libre, pero esta desarrollado bajo la licencia MIT (Massachusetts Institute of Technology), esta licencia no tiene copyright, lo cual permite modificar su código, pero tiene ligeras diferencias con la licencia GNU GPL. En cuanto a tecnologías, permite el desarrollo de aplicaciones utilizando las tecnologías HTML y Ruby. Y su desarrollo es posible para cualquiera de las plataformas disponibles. Además de todas estas características, Rhodes tiene una ventaja sobre Phonegap y Enyo, y es que tiene un servicio en la nube, que permite al desarrollador crear aplicaciones online con el *framework* Rhodes, sin necesidad de tener actualizada la última versión del SDK (Software Development Kit) para cada plataforma.

5.4.3 Tecnología elegida – PhoneGap (Córdova)

Para elegir esta solución se han valorado muchos aspectos, sobre todo uno que juega un papel muy importante a la hora de ir avanzando en el desarrollo de este proyecto y es el coste mínimo de desarrollo. Por tanto, se considera que PhoneGap tiene una menor curva de aprendizaje puesto que su programación se basa en el uso de lenguajes como HTML, CSS y JavaScript. Estos lenguajes ya han sido estudiados previamente a la realización de este TFG, por lo que representará un menor tiempo de aprendizaje. Y si en un futuro se tuviese que incorporar a una persona al equipo de desarrollo, será fácil encontrar uno, a menor coste cuanto más fácil y rápido sea trabajar con esta tecnología, así como de lo amplia que sea la comunidad de desarrolladores de esta herramienta.

Otros aspectos a tener en cuenta han sido los ya marcados en los objetivos, como son el desarrollo de aplicaciones multiplataforma, con las menores modificaciones posibles de código y usar tecnología de licencia libre que no suponga ningún coste adicional.

5.4.3.1 *Introducción al PhoneGap*

PhoneGap es un *framework* implementado por Adobe, para el desarrollo de aplicaciones móviles multiplataforma haciendo uso de tecnologías web. Este *framework* ha tenido gran impacto en el mundo de los desarrolladores por su característica de desarrollo multiplataforma. Para los desarrolladores era un problema el tener que desarrollar la misma aplicación tantas veces como plataformas querían que su aplicación abarcara. Es decir, tenían que desarrollar la aplicación en lenguaje nativo de Android para poder exportarlo a esta plataforma, luego volver a realizar el desarrollo para la plataforma iOS si querían entrar en el mercado de Apple, y así para cada una de las plataformas existentes. Con PhoneGap, esto no ocurre, basta con desarrollar la aplicación una vez, con las tecnologías HTML, CSS y JavaScript, y una vez terminada, mediante el SDK correspondiente a cada plataforma, exportar la aplicación para cada una de ellas. Es por este motivo principal, por el que ha tenido tanta repercusión en el mundo del desarrollo de aplicaciones, y también en el mundo del desarrollo web [26].

5.4.3.2 *SDK soportados*

PhoneGap soporta sino, toda la mayoría de los sistemas operativos existentes, y por supuesto, todos los que tienen un nivel de utilización tal como para ser relevante para los desarrolladores.

A continuación nombramos la gran mayoría de ellos:

- Android.

- iOS.

- Windows Phone.

- BlackBerry OS.

- Web OS.

- Symbian.

Para poder desarrollar en cualquiera de las plataformas citadas, necesitamos el SDK (Software Developer Kit), lo que nos proporciona el kit de software de desarrollo correspondiente a la plataforma. Inicialmente se hará la implementación de la aplicación para Android, por motivos de popularidad, aunque no existen grandes diferencias si se desarrolla para otra plataforma, solamente las que se exija por el SDK correspondiente.

5.4.3.3 *Instalación*

En este apartado se explica cómo se debe proceder a la instalación del *framework* principal, para lo cual se accede a la página web oficial del mismo, y en el apartado "Install" se tiene una breve explicación de cómo hacerlo [26].

Primero, se indica la necesidad de instalar "Node.js" antes de instalar PhoneGap. Node.js es un intérprete JavaScript del lado del servidor cuyo objetivo es permitir al programador construir aplicaciones altamente escalables y cuyo código pueda manejar un alto número de conexiones. Una vez se tiene instalado Node.js, pasamos a instalar PhoneGap. Para ello, basta con utilizar la línea de comandos siguiendo los comandos que se indican, dependiendo del sistema operativo Linux o MacOS en el que se va a desarrollar. Si por el contrario se va a desarrollar desde un sistema Windows, se debe descargar el paquete desde la misma interfaz para instalarlo.

Ahora que ya tenemos instalado el framework en el ordenador, se tiene la funcionalidad para poder crear proyectos iniciales de aplicaciones. Se abre la ventana de comandos en este caso de Windows (CMD). El proyecto que se va a crear tiene como nombre "e-vecinos". El comando para crearlo es el siguiente: <*phonegap create e-vecinos*>. Con esto se creará una nueva carpeta con ese nombre a partir de la ruta en la que se encuentre.

Una vez completados estos pasos, ya se tiene un proyecto creado con el contenido necesario para desarrollar una aplicación para cualquiera de las plataformas citadas en la descripción del *framework*. Así que, se procede a instalar el SDK de Android. Android es una plataforma de software libre, por lo que cuenta con un SDK disponible para todo desarrollador que lo desee que incluye, entre otros elementos, el conjunto completo de API que este sistema soporta. Para descargarlo, basta con visitar la web de Android [27]. Una vez descargado el SDK, es necesario descomprimirlo. La ubicación de los ficheros resultantes no es relevante, pero conviene recordar la ruta para pasos posteriores.

Para terminar, se crea el APK de la aplicación con la instrucción de PhoneGap que se cita así: <*phonegap build Android*>.

5.4.3.4 *Plugin de la cámara*

Para conocer las funciones y partes del dispositivo que hay disponibles para su acceso, tenemos que adentrarnos en la documentación oficial del *framework phonegap*, que se encuentra en la página oficial del mismo [28]. En la documentación, se puede ver, por una parte, los *plugins* oficiales. Estos *plugins*, son una especie de librerías que ofrece el *framework* para acceder de manera nativa a las diferentes funciones del dispositivo para el que se desarrolle.

Para este proyecto se requiere hacer uso de la cámara del dispositivo para que el usuario que acceda a la app, pueda cambiar su foto de perfil. Por tanto, el *plugin* a utilizar es el *plugin* "camara".

Para integrar el *plugin* al proyecto se realiza la siguiente llamada:

phonegap plugin add cordova-plugin-camera[1]

A partir de ese momento se puede acceder a la cámara para tomar fotos programando una única línea JavaScript. Estos son los métodos utilizados en la app móvil de e-vecinos:

```
function tomaFoto() {
        navigator.camera.getPicture(onSuccess, onFail, { quality: 50, correctOrientation: true,
                destinationType: Camera.DestinationType.DATA_URL});
}
function onSuccess(imageData) { ... }
function onFail(message) { ... }
```

Para la llamada al plugin se pasan como parámetros el nombre del método que tratará con el resultado, en este caso el fichero con la foto en Base64, y el método a llamar en caso de que se produzca un error. Un último parámetro permite configurar la cámara del móvil para la foto que se ha de hacer. En el caso de este proyecto se determina la calidad de la foto, se indica que la orientación de la foto resultado ha de ser la misma que la elegida por el usuario (por defecto siempre aparece apaisada) y se le indica cómo debe ser el resultado.

5.4.3.5 *Funcionalidad y Resultado*

Para el aspecto de la aplicación móvil se ha usado una plantilla (*template*) de un diseño ya definido [29] e intentando mantener relación con la plantilla escogida para la parte de la aplicación web. Se ha incluido el CSS "stylemobile.css" en el proyecto e-vecinos dentro de Netbeans, para así hacer uso de este dentro de todas las páginas HTML del proyecto en PhoneGap.

Se han creado 5 páginas *.html que contendrán todo el código necesario para el desarrollo de la aplicación móvil y que están contendidas dentro de la carpeta "www" del proyecto "evecinos_phonegap" que se ha creado con antelación. Estas páginas son las siguientes:

- **index.html**: define un menú de inicio para recorrer las diferentes opciones que se tienen (inicio, facturas, recibos y notificaciones). También se define el código del plugin para acceder a la cámara del propio dispositivo y poder realizar la foto del usuario si este desea añadir o cambiar la que se tiene.
- **login.html:** permite hacer login para entrar a la aplicación, en esta entrega del proyecto solo hemos programado para que a la aplicación se pueda acceder solo como propietario, mostrando así sus pestañas correspondientes.
- **recibos.html:** muestra el listado de recibos relacionados con el inmueble del propietario en acción. Se muestra tanto los recibos que tiene pendientes de pago haciendo una distinción en el color y los que ya están al corriente.
- **facturas.html:** muestra el listado de facturas relacionadas con la finca o comunidad en la que se encuentra el inmueble del propietario en acción. Se muestran todas las facturas pudiendo seleccionar la que se desee y ver una imagen de la misma con todos sus detalles.
- **notificaciones.html:** muestra un listado de notificaciones nuevas, que informan sobre alguna reunión o sobre alguna información importante que considere oportuna enviar el administrador a los propietarios de una determinada finca.

1 En la documentación se indica que el plugin es org.apache.cordova.camera pero si se trata de instalar un mensaje indica que se le ha cambiado el nombre por cordova-plugin-camera.

Una vez hecho el login y validado el usuario, se presenta una página de inicio, donde aparece la foto del usuario y su nombre. Si se pincha sobre el icono de la foto, se puede acceder a la cámara del dispositivo y realizar una nueva foto que quedará inmediatamente almacenada en la BBDD. También aparecen tres enlaces (recibos, facturas y notificación). Cada uno de estos ítems, nos llevará a las funcionalidades definidas a en esta sección. Una vez dentro de cada opción elegida, aparece un submenú para poder movernos desde cualquier página a otra. El conjunto de enlaces está implementado con un código de HTML, CSS, JavaScript, JSON, Ajax.

Por último, el aspecto final de la aplicación móvil “e-vecinos” para Android queda de la siguiente manera:

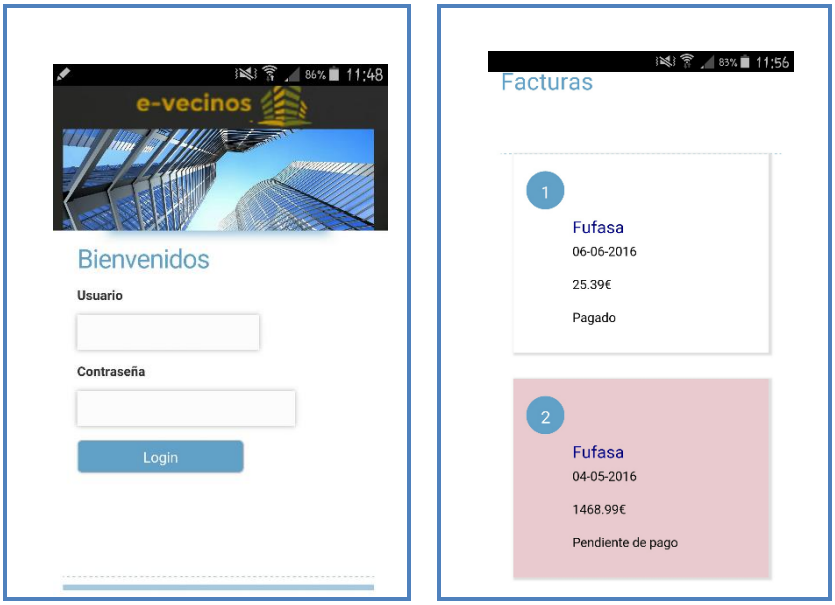


Figura 32. Ejemplo 1 - App móvil 'e-vecinos'.

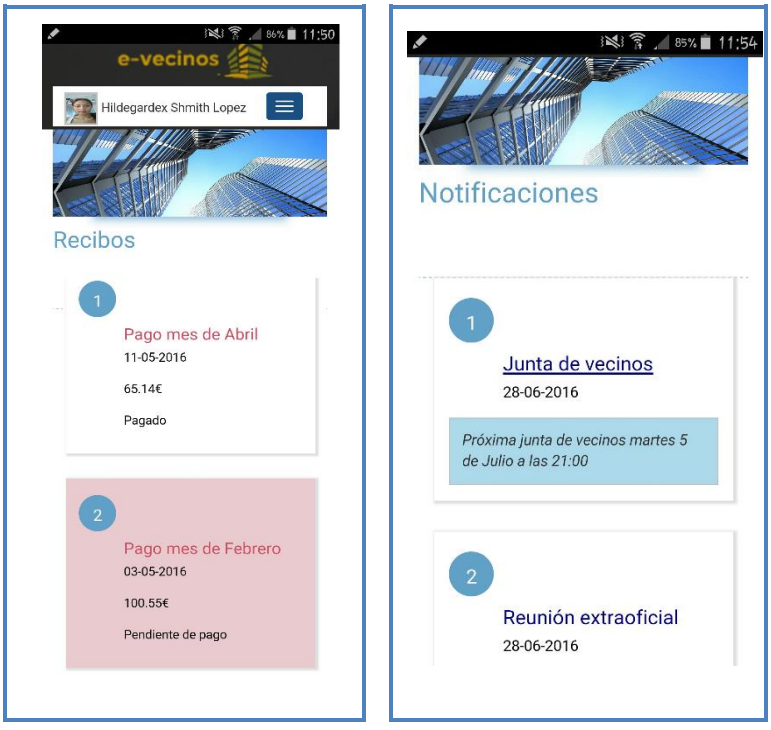


Figura 33. Ejemplo 2 - App móvil 'e-vecinos'.

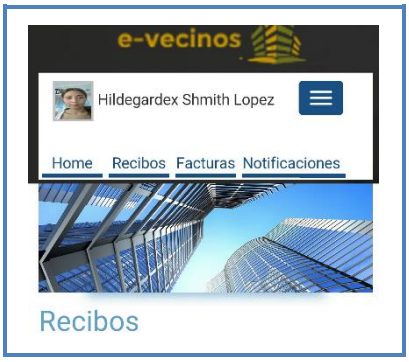


Figura 34. Menú de opciones de 'e-vecinos'.

5.4.4 Comunicación Ajax con JQuery y JSON

El hecho de que una app de PhoneGap se componga de páginas HTML (estático) hace que para que éstas puedan contener la información proveniente de un servidor tengan que generar el código HTML "al vuelo". Es decir, una vez cargadas deben llamar mediante JavaScript al servidor, obtener la información necesaria y rehacer partes del HTML para mostrar esa información. Esta técnica es conocida como Ajax y existen muchas librerías JavaScript que permiten su implementación [30]. Dado que en todo el proyecto se ha hecho uso de JQuery se ha usado su implementación de Ajax para cumplir este cometido.

Por ejemplo esta es la llamada que se realiza para hacer login:

```
$.ajax({
        dataType: 'json',
        url: "http://servidor:8080/evecinos/LoginServlet",
        data: {"name" : $("#name").val(), "password" : $("#password").val()}
 })
 .done(function( data, textStatus, jqXHR ) {
    document.location.href = "index.html";
  })
 .fail(function( jqXHR, textStatus, errorThrown ) {
    alert( "Su usuario o su contraseña no son correctos: " + textStatus);
 });;
```

Como se ve en la llamada se indica que la comunicación se establece mediante mensajes JSON, lo cuál es la URL a la que se ha de conectar (en este caso uno de los servlets realizados para la comunicación con la app móvil) y los parámetros de la llamada: el usuario y la contraseña introducidos por el propietario. En este caso la respuesta no tiene importancia y, si el servidor devuelve que todo va bien se muestra la página index.html.

Otro ejemplo donde se puede ver cómo se recarga el HTML es a la hora de mostrar los recibos:

```
$.ajax({
        dataType: 'json',
        url: "http://servidor:8080/evecinos/RecibosServlet",
        data: {}
 })
```

```
.done(function( data, textStatus, jqXHR ) {
        for(i=0;i<data.vector[0].recibo.length;i++) {
        º$("#recibosDiv").append('<div class="col-md-4 grid_1_of_3"><div class="ser-grids"><span class="dropcap">' + (i+1) + '</span><div class="extra-wrap"><h5><span style="color:#D44F66;">' + data.vector[0].recibo[i].concepto + '</span></h5><p>' + data.vector[0].recibo[i].fecha + ' </p><p>' + data.vector[0].recibo[i].importe + '&euro;</p></div><div class="clearfix"></div></div> </div>');
        }
})
fail(function( jqXHR, textStatus, errorThrown ) {
    alert( "La solicitud ha fallado: " +  textStatus);
});
```

Se puede ver como se utiliza el método “append” de JQuery para ir anexando la información de cada recibo.

5.4.5 Parte servidor desarrollada para el móvil

Para atender las llamadas que vienen de la app móvil se han desarrollado una serie de *servlets*. La funcionalidad de éstos, consiste en recoger los parámetros, llamar a la lógica, pasar los objetos Java de respuesta a JSON y finalmente devolver la respuesta.

En la comunicación entre la app de e-vecinos en un dispositivo móvil y el servidor se han definido mensajes con información codificada en JSON. Cuando el servidor devuelve información dispone de ella como objetos Java, por ejemplo una lista de objetos de tipo Recibo. XStream es la librería que permite convertir estos objetos Java en JSON para ser devueltos a la app móvil [31].

Éste sería el código utilizado para realizar la conversión:

```
XStream xstream = new XStream(new JettisonMappedXmlDriver());
xstream.setMode(XStream.NO_REFERENCES);
xstream.autodetectAnnotations(true);
xstream.registerConverter(new DateConverter("dd-MM-yyyy", new String[] {}));
String respuesta = xstream.toXML(lRecibos);
```

Lo más importante de este código es:

- En la primera línea se define el objeto XStream con el que se trabajará. Si no se le pasase ningún parámetro el resultado sería XML. Para obtener JSON es necesario incluir la librería ‘*XStream – jettison-1,2.jar’* y pasar el parámetro.
- En la cuarta línea se añade un conversor para los objetos de tipo fecha. Al pasar a JSON las fechas aparecerán con el formato dd-MM-yyyy.
- En la última línea es donde se produce la conversión de la lista de objetos Recibo.

En la tercera línea del código se indica que se deben aceptar anotaciones. Estas anotaciones se ubican en las clases que se van a convertir para indicar cómo debe realizarse la conversión.

Las anotaciones utilizadas son:

- *@XStreamAlias*: Se sitúa al principio de la clase e indica el nombre que se le dará a los objetos de esta clase en el fichero JSON.
- *@XStreamOmitField*: En algunas clases hay campos que no se desea que sean incluidos en el fichero JSON.

Capítulo 6. Conclusiones y propuesta de trabajo futuro

Detallaremos algunas conclusiones fruto del desarrollo técnico que ha acompañado a este trabajo, al proceso de investigación asociado. Además, se propondrán algunas de las líneas futuras que se pretende seguir.

6.1 Conclusiones

Como resultado de la realización de este trabajo de fin de grado, cabe destacar entre otras cosas, la experiencia y aprendizaje adquirido tanto en la investigación de nuevas tecnologías y herramientas, así como en el desarrollo de aplicaciones web y móviles multiplataforma. Además, se ha conocido y comprendido mejor las posibles necesidades técnicas que exige el desarrollo de estas aplicaciones ya sea web o móvil, en función de sus requerimientos. También las variables no técnicas que son determinantes en la elección de una tecnología frente a otra. Hoy en día existen multitud de tecnologías y lenguajes de programación que hacen casi lo mismo. Es necesario, por tanto, formarse un criterio para poder elegir la más adecuada para este proyecto. Teniendo en cuenta, sobre todo, las necesidades presentes y, lo más complicado de prever, también las futuras.

Ha sido un trabajo muy fructífero para aumentar los conocimientos en un campo que ha ido cogiendo mucha fuerza durante los últimos años, con el incremento de la cuota de mercado que ha ido experimentando los Smartphone y Tabletas. No obstante, ha requerido un esfuerzo muy grande y una dedicación especial para poder ir avanzando en las diferentes etapas del desarrollo de este TFG, teniendo que subsanar muchos errores, resolviendo inquietudes, pero sobre todo, manteniendo siempre claro el objetivo de la funcionalidad de esta aplicación de negocio.

Este proyecto se ha desarrollado en una fase experimental, donde se ha podido llegar a comprobar su verdadera utilidad y eficacia a la hora de gestionar información (de momento básica) relacionada con una comunidad de vecinos. La cual se ha aprendido a estructurar y manejar dentro de estas aplicaciones, para poder de esta manera presentarla de una forma atractiva al usuario final. En cuanto al alcance del trabajo, se dirá, que se ha conseguido prácticamente todos los objetivos planteados, ya que el proyecto realiza todas las acciones detalladas inicialmente y da una idea clara de lo que se pretende conseguir.

Ahora el proyecto se encuentra en un punto, que a pesar de tener una funcionalidad que se considera aceptable, tiene por delante multitud de funciones y posibilidades a explotar, tanto en la gestión de toda la información relevante que conlleva administrar una comunidad de vecinos, ya sea para el administrador, así como para el propietario. Se han estudiado muchas posibles soluciones para llevar a cabo la idea principal de este proyecto y se podría haber encontrado una solución técnica más avanzada. Sin embargo, la necesidad se centraba en el coste mínimo y en la medida posible hacer que la curva de aprendizaje de algunas de las tecnologías usadas fuese lo menos extensa.

Finalmente, durante este trabajo se han cubierto muchas expectativas personales y académicas. De esta manera, se concluye que se debe seguir formando en las nuevas tecnologías, ya que se vive en mundo en continuo cambio, con una gran tendencia a digitalizar todo tipo de información. Por tanto, es interesante estar siempre a la vanguardia de la Telemática, ya que esto puede abrir muchas más puertas en la rama profesional.

6.2 Propuesta de trabajo futuro

En una segunda iteración de este proyecto, se quieren completar casos de uso para:

- Ampliar las notificaciones para que los propietarios puedan realizar consultas, quejas y reclamaciones a los administradores, pasando a ser más bien un apartado de incidencias creados por la parte del propietario.

- Realizar un chat en el que tanto administradores como propietarios puedan estar en contacto. El modo de funcionamiento sería similar al de WhatsApp, ya que la mayoría de las personas están acostumbradas a su uso. Su implementación se realizaría mediante el protocolo XMPP (Jabber).

- Crear un apartado en el que los administradores puedan dar de alta servicios concertados para la comunidad: fontaneros, electricistas, etc.

- Ampliar el sistema de recibos de forma que cada inmueble pueda ser ponderado a la hora de repartir los gastos.

- Crear un apartado de juntas de vecinos que emita notificaciones de aviso a las nuevas juntas con el orden del día y donde se pueda consultar el acta de cada junta.

- Completar toda la información financiera de gastos e ingresos de forma que se pueda consultar en todo momento el estado de las cuentas de la comunidad y se puedan hacer previsiones a futuro de las mismas.

- Se debería realizar una segunda app móvil para los administradores de forma que todas las funcionalidades de la web se ofrezcan también para dispositivos móviles.

- Para este trabajo fin de carrera se ha realizado la app sólo para Android. Se querría obtener el mismo resultado al menos para iPhone e iPad. Para ello sólo habría que ejecutar el Phonegap en un sistema Mac OS.

- En relación a la seguridad, se debería requerir que las contraseñas tuviesen un nivel más alto de seguridad, pudiendo poner números, caracteres y que no fuesen tan cortas y sencillas. Por otro lado, abordar el tema de la comunicación entre el móvil y el servidor, donde no se deba pasar usuario y la contraseña en abierto, sino usando un túnel SSL.

Capítulo 7. Bibliografía

[1] Estadísticas acerca del Smartphone y el uso de Internet. http://www.fundaciontelefonica.com/arte_cultura/publicaciones-listado/pagina-item-publicaciones/itempubli/483/ [Online]

[2] Google Trends. https://www.google.com/trends/explore#q=mobile%20app. [Online]

[3] StarUML – software. http://staruml.io/ [Online]

[4] DBDesigner4 – software. http://fabforce.net/downloads.php [Online]

[5] DBDesigner4 – manual. http://downloads.mysql.com/DBDesigner4/DBDesigner4_manual_1.0.42.pdf. [Online]

[6] SQLyog – software. https://github.com/webyog/sqlyog-community/wiki/Downloads. [Online]

[7] Wikipedia, "SQLyog". https://en.wikipedia.org/wiki/SQLyog [Online]

[8] MySQL (sistema de gestión de bases de datos) – software. https://www.mysql.com/ [Online]

[9] Wikipedia, "NetBeans". https://es.wikipedia.org/wiki/NetBeans [Online]

[10] IDE Netbeans – software. https://netbeans.org/ [Online]

[11] Wikipedia, "HTML". https://es.wikipedia.org/wiki/HTML [Online]

[12] Wikipedia, "XHTML". https://es.wikipedia.org/wiki/XHTML [Online]

[13] w3c.es "CSS3". http://www.w3c.es/Divulgacion/GuiasBreves/HojasEstilo [Online]

[14] Wikipedia, "JavaScript". https://es.wikipedia.org/wiki/JavaScript [Online]

[15] Wikipedia, "Framework". https://es.wikipedia.org/wiki/Framework [Online]

[16] Wikipedia, "JavaServer Faces," https://es.wikipedia.org/wiki/JavaServer_Faces [Online]

[17] JavaServer Faces (framework de programación web). http://www.oracle.com/technetwork/java/javaee/javaserverfaces-139869.html [Online]

[18] JavaServer Faces – curso http://www.globalmentoring.com.mx/curso-jsf/leccion2/PDFs/Curso_JSF_Teoria%20-%20leccion2.pdf [Online]

[19] Managed Bean – Tutorial. http://www.tutorialspoint.com/jsf/jsf_managed_beans.htm. [Online]

[20] Managed Bean – Anotaciones. https://javaserverfaces.java.net/nonav/docs/2.1/managed-bean-javadocs/index.html [Online]

[21] Wikipedia, "Facelets". https://es.wikipedia.org/wiki/Facelets [Online]

[22] Plantilla HTML Peaches. http://themeforest.net/item/peach-clean-smooth-admin-template/780016. [Online]

[23] EJB (http://www.davidmarco.es/articulo/introduccion-a-ejb-3-1-i [Online]

[24] JPA http://www.lab.inf.uc3m.es/~a0080802/RAI/jpa.html [Online]

[25] GlassFish (servidor de aplicaciones). https://glassfish.java.net/ [Online]

[26] PhoneGap (Framework de programación para dispositivos móviles). http://phonegap.com/. [Online]

[27] Android SDK. http://android-sdk.uptodown.com/windows [Online]

[28] Plugin cámara – tutorial. https://github.com/apache/cordova-plugin-camera#module_camera.getPicture [Online]

[29] Plantilla HTML para la app móvil. https://w3layouts.com/ameliorat-a-medical-category-flat-bootstrap-responsive-web-template/. [Online]

[30] JQuery (biblioteca de JavaScript): https://jquery.com/ [Online]

[31] XStream (librería Java para pasar objetos Java a JSON – comunicación con la app móvil) http://x-stream.github.io/ [Online]

[32] Fallr (plugin Jquery para abrir ventanas). http://amatyran.com/codecanyon/fallr-js/ [Online]

Capítulo 8. ANEXOS

ANEXO A: JavaServer Faces API

Packages	
javax.faces	Top level classes for the JavaServer(tm) Faces API.
javax.faces.application	APIs that are used to link an application's business logic objects to JavaServer Faces, as well as convenient pluggable mechanisms to manage the execution of an application that is based on JavaServer Faces.
javax.faces.component	Fundamental APIs for user interface components.
javax.faces.component.behavior	APIs for attaching additional behavior to user interface components.
javax.faces.component.html	Specialized user interface component classes for HTML.
javax.faces.component.visit	APIs for traversing a user interface component view.
javax.faces.context	Classes and interfaces defining per-request state information.
javax.faces.convert	Contains classes and interfaces defining converters.
javax.faces.el	**DEPRECATED** Classes and interfaces for evaluating and processing reference expressions.
javax.faces.event	Interfaces describing events and event listeners, and concrete event implementation classes.
javax.faces.lifecycle	Classes and interfaces defining lifecycle management for the JavaServer Faces implementation.
javax.faces.model	Standard model data beans for JavaServer Faces.
javax.faces.render	Classes and interfaces defining the rendering model.
javax.faces.validator	Interface defining the validator model, and concrete validator implementation classes.
javax.faces.view	Classes for defining a View Declaration Language (VDL) for authoring JavaServer Faces user interfaces.
javax.faces.view.facelets	This package contains public classes for the Java code API of Facelets.
javax.faces.webapp	Classes required for integration of JavaServer Faces into web applications, including a standard servlet, base classes for JSP custom component tags, and concrete tag implementations for core tags.

ANEXO B: Etiquetas JSF

JavaServer Faces dispone de un conjunto básico de etiquetas que permiten crear fácilmente componentes dinámicos en las páginas web. Estas etiquetas son:

- h:commandButton. Un botón al que podemos asociar una acción.
- h:commandLink. Un enlace hipertexto al que podemos asociar una acción.
- h:dataTable. Crea una tabla de datos dinámica con los elementos de una propiedad de tipo Array o Map del bean.
- h:form. Define el formulario JSF en la página JSP-
- h:graphicImage. Muestra una imagen jpg o similar.
- h:inputHidden. Incluye un campo oculto del formulario.
- h:inputSecret . Incluye un campo editable de tipo contraseña (no muestra lo que se escribe)
- h:inputText. Incluye un campo de texto normal.
- h:inputTextarea. Incluye un campo de texto multilínea.
- h:message. Imprime un mensaje de error en la página (si se ha producido alguno).
- h:messages. Imprime varios mensajes de error en la página, si se han producido.
- h:outputFormat. Muestra texto parametrizado. Utiliza la clase java.text.MessageFormat de formateo.
- h:outputLabel. Muestra un texto fijo.
- h:outputLink. Crea un enlace hipertexto.
- h:outputText
- h:panelGrid. Crea una tabla con los componentes incluidos en el panelGrid.
- h:panelGroup. Agrupa varios componentes para que cierto componente los trate como un único componente (por ejemplo para meter varios componentes en una celda de un panelGrid.
- h:selectBooleanCheckbox. Crea una casilla con dos estados: activado y desactivado.
- h:selectManyCheckbox. Crea un conjunto de casillas activables.
- h:selectManyListbox. Crea una lista que permite seleccionar múltiples elementos.
- h:selectManyMenu. Crea una lista desplegable de selección múltiple.
- h:selectOneListbox. Crea una lista en la que se puede seleccionar un único elemento.
- h:selectOneMenu. Crea na lista desplegable de selección.
- h:selectOneRadio. Crea una lista de botones, redondos normalmente, excluyentes.

ANEXO C: JSON Tutorial

Due to XStream's flexible architecture, handling of JSON mappings is as easy as handling of XML documents. All you have to do is to initialize XStream object with an appropriate driver and you are ready to serialize your objects to (and from) JSON.

XStream currently delivers two drivers for JSON: The JsonHierarchicalStreamDriver and the JettisonMappedXmlDriver. The first one does not have an additional dependency, but can only be used to write XML, while the second one is based on Jettison and can also deserialize JSON to Java objects again. However, since the JettisonMappedXmlDriver transforms plain XML into JSON, you might get better JSON strings with the JsonHierarchicalStreamDriver, because this driver knows often about the type of the written data and can act properly.

One word of warning. JSON is made for an easy data transfer between systems and languages. It's syntax offers much less possibilities to express certain data structures. It supports name/value pairs of primitive data types, arrays and lists and allows to nest them - but that's it! No references, no possibility for meta data (attributes), no properties with same names (as generated for implicit collections), etc. Therefore do not expect wonders. XStream (and Jettison) try their best, but the procedure to convert any kind of object into JSON is a lossy transformation and especially deserialization will not be possible for any construct. See also FAQ for JSON limitations. Also note that any number value in JavaScript is a 64-bit double precision float value according IEEE 754 standard. In consequence it is not possible to represent all values of a Java long type without loss of precision. See again FAQ for JSON and JavaScript.

Since JSON has no possibility to express references, you should always set the NO_REFERENCES mode writing JSON.

- **Jettison driver**

Jettison driver uses Jettison StAX parser to read and write data in JSON format. It is available in XStream since version 1.2.2 and is implemented in:

`com.thoughtworks.xstream.io.json.JettisonMappedXmlDriver` *class.*

To successfully use this driver you need to have the Jettison project and StAX API in your classpath (see reference for optional dependencies). Alternatively you can download JARs manually.

The following example:

Write to JSON with the Jettison-based implementation:

```
package com.thoughtworks.xstream.json.test;

import com.thoughtworks.xstream.XStream;
import com.thoughtworks.xstream.io.json.JettisonMappedXmlDriver;

public class WriteTest {

        public static void main(String[] args) {

    Product product = new Product("Banana", "123", 23.00);
    XStream xstream = new XStream(new JettisonMappedXmlDriver());
    xstream.setMode(XStream.NO_REFERENCES);
    xstream.alias("product", Product.class);

    System.out.println(xstream.toXML(product));
```

```
        }

}
```

produces the following JSON document:

```
{"product":{"name":"Banana","id":123,"price":23.0}}
```

The following code:

Read from JSON

```
package com.thoughtworks.xstream.json.test;

import com.thoughtworks.xstream.XStream;
import com.thoughtworks.xstream.io.json.JettisonMappedXmlDriver;

public class ReadTest {

        public static void main(String[] args) {
                String json = "{\"product\":{\"name\":\"Banana\",\"id\":123"
                   + ",\"price\":23.0}}";

                XStream xstream = new XStream(new JettisonMappedXmlDriver());
                xstream.alias("product", Product.class);
                Product product = (Product)xstream.fromXML(json);
                System.out.println(product.getName());
        }

}
```

serializes JSON document created with preceding example back to Java object. It prints:

```
Banana
```

as a result.

< FALLR >

[GENERAL PATTERN] -> ONE & ONLY SIMPLE DEVINITIVE GLOBAL RULE TO MASTER :

```
$.fallr(method:String [,options:Object] [,callback:function]);
```

ANEXO D: Córdova-plugin-camera

This plugin defines a global navigator.camera object, which provides an API for taking pictures and for choosing images from the system's image library.

Although the object is attached to the global scoped navigator, it is not available until after the deviceready event.

```
document.addEventListener("deviceready", onDeviceReady, false);
function onDeviceReady() {
console.log(navigator.camera);
}
```

- **Installation**

Is possible to install via repo url directly (unstable)

cordova plugin add *https://github.com/apache/cordova-plugin-camera.git*

- **Camera**

camera.getPicture(successCallback, errorCallback, options)

Takes a photo using the camera, or retrieves a photo from the device's image gallery. The image is passed to the success callback as a Base64-encoded String, or as the URI for the image file.

The *camera.getPicture function* opens the device's default camera application that allows users to snap pictures by default this behavior occurs, when:

Camera.sourceType equals *Camera.PictureSourceType.CAMERA.*

Once the user snaps the photo, the camera application closes and the application is restored.

If Camera.sourceType is Camera.PictureSourceType.PHOTOLIBRARY or Camera.PictureSource Type.SAVEDPHOTOALBUM, then a dialog displays that allows users to select an existing image. The camera.getPicture function returns a CameraPopoverHandleobject, which can be used to reposition the image selection dialog, for example, when the device orientation changes.

The return value is sent to the cameraSuccess callback function, in one of the following formats, depending on the specifiedcameraOptions:

- ✓ A String containing the Base64-encoded photo image.
- ✓ A String representing the image file location on local storage (default).

You can do whatever you want with the encoded image or URI, for example:

- ✓ Render the image in an <img> tag, as in the example below
- ✓ Save the data locally (LocalStorage, Lawnchair, etc.)
- ✓ Post the data to a remote server.

NOTE: Photo resolution on newer devices is quite good. Photos selected from the device's gallery are not downscaled to a lower quality, even if a quality parameter is specified. To avoid common memory problems, set Camera.destinationType toFILE_URI rather than DATA_URL.

ANEXO E: Términos

- *Framework:* término con el que se define un amplio conjunto de elementos que permite desarrollar y organizar software utilizando un determinado lenguaje, sistema o tecnología. Habitualmente incluye bibliotecas, programas de desarrollo o manuales.
- *Dispositivo móvil:* aparato electrónico que es de reducido tamaño, cuenta con cierta capacidad tanto para la computación como para el almacenamiento de datos y cuenta con elementos de E/S básicos, por ejemplo pantalla y/o teclado.
- *Sistema operativo*: programa cuya finalidad principal es simplificar el manejo y explotación de un elemento con capacidad computacional, gestionando sus recursos, ofreciendo servicios a las demás aplicaciones y ejecutando mandatos del usuario.
- *Controlador:* programa que permite al sistema operativo interactuar con unos periféricos, abstrayéndolo y proporcionando una interfaz para usarlo. También conocido como driver.
- *Smartphone:* dispositivo móvil que representa una evolución de los teléfonos móviles, con la inclusión de pantalla táctil, teclado, conexión Wi-Fi, aplicaciones de usuario como navegador web o cliente de correo, entre otros.
- *GUI:* siglas de Graphical User Interface, en español Interfaz Gráfica de Usuario. Representa la parte del software que, mediante un contexto o lenguaje principalmente visual y simbólico, permite al usuario utilizar una aplicación.
- *API:* siglas de Application Programming Interface, en español Interfaz de Programación de Aplicaciones. Consiste en un conjunto de llamadas que ofrecen acceso a funciones y procedimientos, representando una capa de abstracción para el desarrollador.
- *SDK:* siglas de Software Development Kit, en español Kit de Desarrollo de Software. Constituye un conjunto de herramientas que permiten a un desarrollador crear aplicaciones para una determinada plataforma o lenguaje.
- *URI:* siglas de Uniform Resource Identifier, en español Identificador de Recursos Uniforme. Un URI es una cadena corta de caracteres que identifica inequívocamente un recurso (servicio, página, documento, dirección de correo electrónico, enciclopedia, etc). Normalmente estos recursos son accesibles en una red o sistema basado en el Protocolo IP.
- *Servlet:* elemento de la tecnología Java, que extiende la funcionalidad de un servidor Web, aceptando y procesando peticiones.
- *XMPP:* siglas de Extensible Messaging and Presence Protocol, en español Protocolo Extensible de Mensajería y Presencia. Es un protocolo basado en XML que se utiliza en servicios de mensajería instantánea.
- SSL: siglas de Secure Sockets Layer, en español Capa de Puertos Seguros. Es un protocolo criptográfico que proporciona autenticación y privacidad de la información entre extremos sobre Internet mediante el uso de criptografía, es decir, comunicaciones seguras por una red.