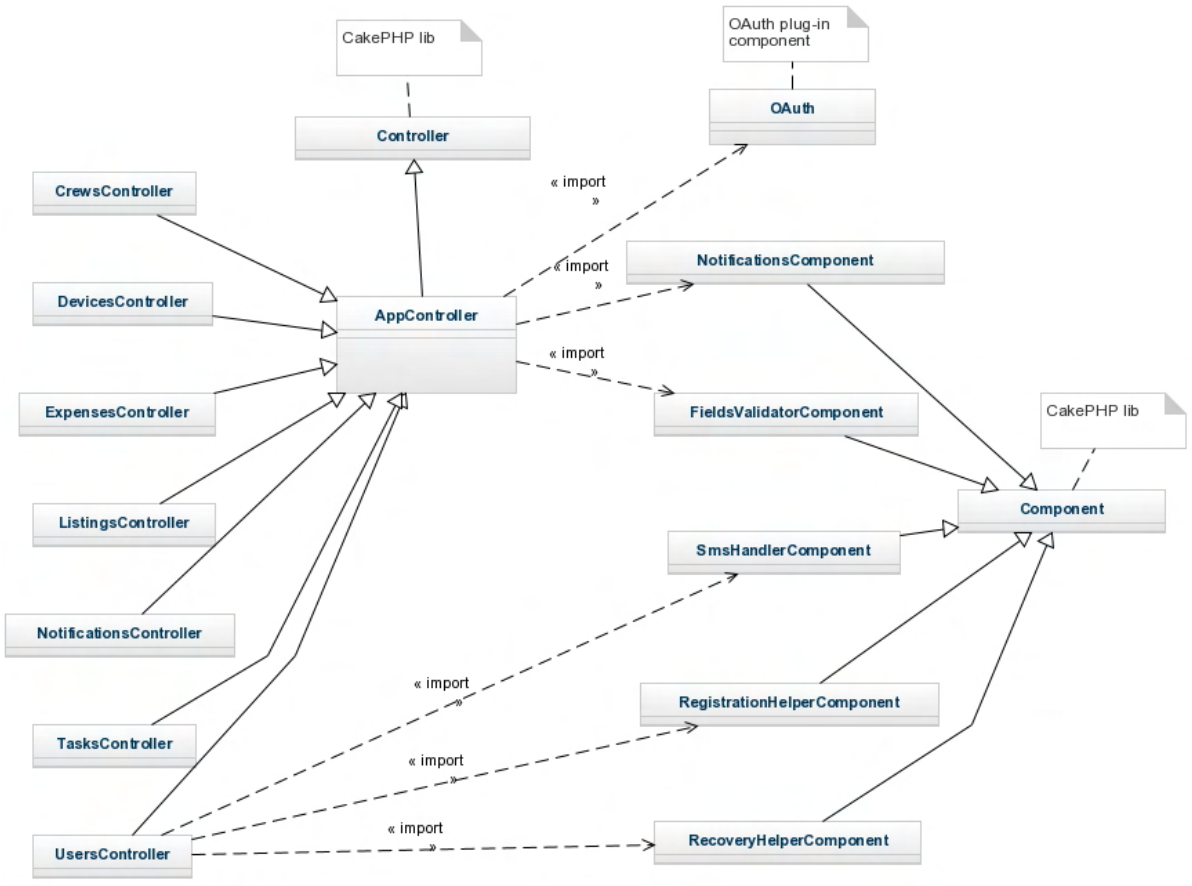

Figura 6.8: Diagrama general de jerarquía de clases

6.3. Validación de requisitos

En esta sección, una vez diseñado el sistema, se comprueba que los requisitos funcionales de la aplicación quedan satisfechos por los métodos y operaciones que el servicio *FlatCrew Server* ofrece. A continuación, en la tabla 6.1, se relacionan dichos métodos directamente con los casos de uso.

Tras observar la tabla queda comprobado que todos los casos de uso han sido cubiertos con los correspondientes métodos en la parte servidor para que el cliente sea capaz de acceder a los recursos a través de los verbos GET y POST, empleando la ruta de medios adecuada. Sin embargo, en la tabla existe un vacío para el caso de uso "Recargar monedero" esto se debe a que esa funcionalidad no es proporcionada por el servidor *FlatCrew Server*, han de ser los servicios propios de PangoPay los que suplan dicha necesidad.

Validación	
Caso de uso	**Método**
Crear grupo	CrewsController.add()
Abandonar grupo	CrewsController.leave()
Editar grupo	CrewsController.edit()
Crear tarea	TasksController.add()
Eliminar tarea	TasksController.delete()
Modificar tarea	TasksController.edit()
Crear gasto	ExpensesController.add()
Modificar gasto	ExpensesController.edit()
Eliminar gasto	ExpensesController.delete()
Alertar deudor	ExpensesController.alertDebt() ExpensesController.alertUser()
Crear lista	ListingsController.add()
Modificar lista	ListingsController.edit()
Abandonar lista	ListingsController.leave()
Cambiar líneas	ListingsController.toogleCheck() ListingsController.addLine() ListingsController.deleteLine()
Saldar deudas	ExpensesController.settle() ExpensesController.settleAll()
Recargar monedero	-
Invitar usuarios	UsersController.invite()
Gestionar cuentas	UsersController.edit() UsersController.delete()
Registrarse en FlatCrew	UsersController.register() UsersController.confirm()
Acceder a FlatCrew	OAuth 2.0 *plug-in*
Restablecer contraseña	UsersController.recoverPasswd() UsersController.resetPasswd()

Tabla 6.1: Validación de requisitos, casos de uso

7 Implementación y pruebas

7.1. Detalles de implementación

En esta sección se describirán los aspectos relevantes de la implementación del servicio *FlatCrew Server* tales como adaptaciones de especificaciones previas, consideraciones de funcionalidad y detalles de la arquitectura. Concretamente, se dividirá el contenido en tres subapartados: Transacciones, Componentes CakePHP e Implementación OAuth 2.0.

7.1.1. Transacciones

En CakePHP el modelo de datos se define mediante unas clases en la carpeta *Models* y el framework se encarga de vincular esa definición con la base de datos para insertar, borrar y actualizar las tablas. En este caso, detrás hay una base de datos MySQL cuya estructura de tablas y relaciones es creada automáticamente por CakePHP una vez se han definido correctamente las clases del modelo mencionadas anteriormente, las relaciones entre entidades se definen en estas clases. Por otra parte, la mayoría de llamadas a los métodos y su ejecución en los controladores de PHP, que responden a las peticiones HTTP, provocan una interacción con la base de datos. Estas interacciones, en ciertos casos, son de una cierta complejidad y requieren emplear transacciones para conseguir que todo el conjunto de operaciones se ejecute de forma atómica y no se hagan efectivas hasta comprobar que se han cumplido todas las condiciones necesarias derivadas de la lógica de la aplicación y que no se ha producido ningún otro error. Para dar soporte a este tipo de necesidades, CakePHP proporciona una clase en su biblioteca que ejerce de vínculo entre los modelos y la fuente de datos que los modelos representan. Esta clase es *DataSource* , ver documentación de CakePHP [3].

De esta forma, haciendo uso de *DataSource* podemos configurar una transacción dentro del controlador. A continuación se muestra un extracto de código para obtener y hacer uso del *DataSource* para implementar la transacción.

```
1
2 //Model es la clase que representa una entidad de la base de
     datos
3 //Sustituirlo por el nombre de la entidad
4
5 $dataSource = $this->Model->getDataSource();
6 $dataSource->begin();
7
```

```
//Operaciones que pueden afectar a la BDD

if(/*Todo correcto*/){
    $dataSource->commit();
}else{
    $dataSource->rollback();
}
```

Las transacciones se han empleado, por ejemplo, en el controlador *CrewsController* para evitar tener una estructura relacional inconsistente tras la petición de un usuario para abandonar un *Crew*. Podría ocurrir que, siendo el usuario el único miembro del *Crew*, se eliminara su relación con el grupo pero no se eliminara éste de la tabla *Crews* por algún problema o error en la base de datos.

7.1.2. CakePHP: Componentes

Es habitual encontrarse con fragmentos de código repetidos a lo largo del desarrollo de software. Además de la herencia propia de PHP, CakePHP proporciona unas clases designadas específicamente para extraer esa funcionalidad común y conseguir la reutilización de ese código entre los diversos controladores sin necesidad de tener fragmentos repetidos en cada uno de ellos. Esta clase se llama *Component* y, además, puede elegirse qué componentes se importan para cada controlador, según las necesidades de cada uno de ellos. Esto se realiza declarando en el controlador un atributo llamado "components" en el que se almacenará un *array* de cadenas de texto con los nombres de cada uno de los componentes que se desea importar, CakePHP se encargará de cargar esos componentes. A continuación puede verse el fragmento que declara y asigna el array al atributo "components".

```
class AppController extends Controller {

    public $components = [
        'RequestHandler',
        'OAuth.OAuth',
        'FieldsValidator',
        'Notifications'
    ];
```

Por otro lado, para que CakePHP sea capaz de encontrar los componentes que hemos indicado que debe cargar en cada controlador, deben cumplirse las siguientes condiciones:

- La clase que define el componente debe encontrarse en la carpeta "Components" de la carpeta "app" de la estructura del proyecto.
- El nombre incluido en la lista de componentes ha de ser exactamente el mismo que el de la clase.
- La clase debe extender a la clase "Component" proporcionada por el propio framework.

Una vez se han creado e incluido los componentes en el controlador, son accesibles desde cualquier método de éste a través de `$this`, por ejemplo, `$this->NombreDelComponente->metodo()`. En el desarrollo de *FlatCrew Server*, se han creado los siguientes componentes:

FieldsValidatorComponent
: Para comprobar cómodamente si los campos recibidos en la petición HTTP son los que se esperan para el método en concreto, pasándole un *array* con los campos que se espera encontrar.

NotificationsComponent
: Encapsula todas las operaciones relacionadas con las notificaciones: guardar notificaciones, enviar notificaciones *push* Android o enviar un correo electrónico.

RecoveryHelperComponent
: Proporciona los métodos necesarios para gestionar la recuperación y restablecimiento de contraseñas

RegistrationHelperComponent
: Contiene métodos para validar los datos del registro de usuario, proporciona el código de validación, confirma el registro guardando los datos de usuario en la base de datos, además de limpiar los datos temporales que pudieran haberse almacenado.

SmsHandlerComponent
: Ofrece una forma cómoda de enviar un SMS a un usuario dados el número de teléfono y el mensaje sin necesidad para el controlador de conocer los detalles del servicio externo contratado con el que se gestiona el envío.

Para más detalles acerca de la clase "Component", ver la documentación de CakePHP [2].

7.1.3. Implementación OAuth 2.0

Puesto que otros productos y servicios de PaynoPain emplean el estándar OAuth 2.0 para la autorización de aplicaciones, es natural haber adoptado también este protocolo para *FlatCrew* de forma que se consiga una plataforma homogénea. Debido a la gran cantidad de tareas que se presentan en este trabajo y la reducida cuota temporal, se decidió hacer uso de un *plug-in* para CakePHP que también ha sido utilizado previamente por la empresa para llevar a cabo la implementación de OAuth 2.0, ver respositorio de GitHub [4].

El complemento proporciona, además de un componente de CakePHP con las operaciones necesarias para el protocolo OAuth 2.0, las clases del modelo de datos que CakePHP emplea para crear las tablas de la base de datos donde se almacenaran los identificadores de cliente, los códigos de acceso, los *tokens*, etc.

Sin embargo, para hacer un uso correcto del *plug-in*, es necesario estudiar y conocer previamente el protocolo. A continuación se exponen algunos de los aspectos esenciales.

En primer lugar, el protocolo OAuth 2.0 define cuatro roles:

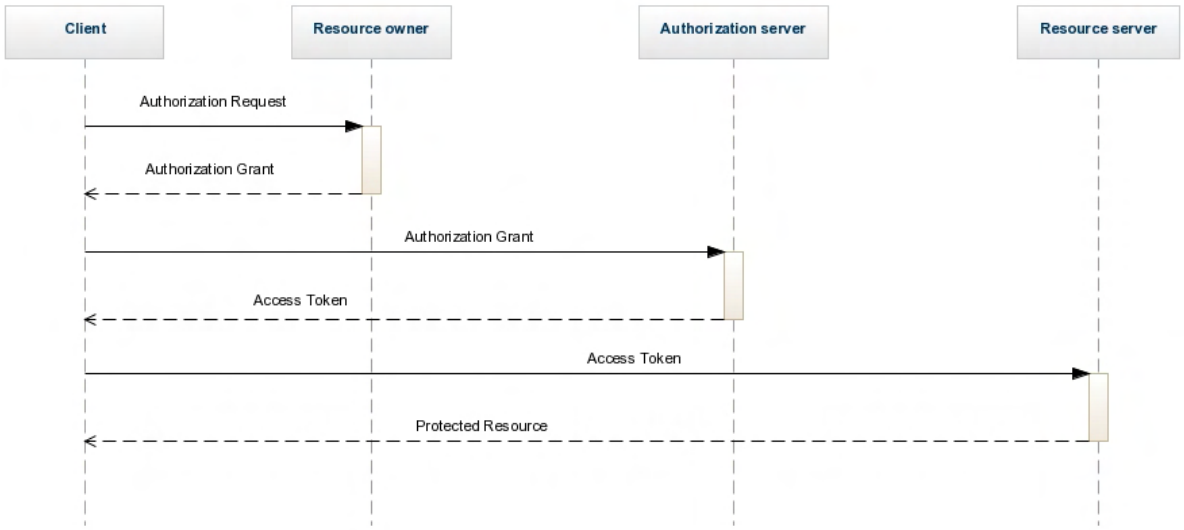

Figura 7.1: Diagrama de sequencia OAuth 2.0: estándar

Resource Owner
: Es el propietario de los recursos a los que se quiere acceder y debe garantizar el acceso en primer lugar.

Authorization Server
: Es el servidor de autenticación y se encarga de proveer los *tokens* de acceso después de haber obtenido autorización para acceder a los recursos.

Resource Server
: Es el servidor de recursos y es donde se encuentran los recursos protegidos. Capaz de aceptar y responder peticiones a estos recursos si el *token* de acceso es válido.

Client
: Es el cliente que quiere acceder a los recursos, no es determinante la implementación específica de esta aplicación.

La interacción entre estos roles puede observarse en la Figura 7.1.

Para esta implementación en concreto, se da la situación de tener tres de los cuatro roles comprendidos en uno mismo. El servicio que se ha desarrollado para *FlatCrew Server* lleva a cabo tanto el rol de *Resource Owner*, como el de *Authorization Server* y el de *Resource Server*.

Por otro lado, cabe prestar atención a los tipos de autorización y el funcionamiento de los

Figura 7.2: Diagrama de sequencia OAuth 2.0: autorización por credenciales

tokens. Existen varios tipos de autorización a emplear contra el servidor de autenticación pero en esta implementación en concreto sólo se van a emplear los credenciales de cada usuario o *refresh token* en caso de contar con uno. Así, pues, la secuencia de obtención de un recurso, será la que se muestra en la Figura 7.2.

En cuanto a la implementación de esta parte, sólo queda comentar que en la configuración por defecto del *plug-in* OAuth 2.0, ambos *access token* y *refresh token* tienen un periodo de validez y es necesario solicitarlos de nuevo una vez han expirado. Sin embargo, para este proyecto se toma la decisión de hacer que los *refresh tokens* no sean caducos de forma que una vez el usuario ha introducido sus credenciales, será la aplicación cliente la que se encargue de solicitar los nuevos *access token* cuando expiren, haciendo uso del *refresh token* que se ha obtenido en el momento de acceder a la aplicación. Este proceder imita al de algunas aplicaciones populares para dispositivos móviles como Twitter, puede verse en la Figura 7.3.

Finalmente, como consideraciones a tener en cuenta, ha existido cierta controversia en cuanto a la seguridad y a la utilidad como protocolo de OAuth 2.0. Eran Hammer, uno de los coordinadores y desarrolladores de OAuth 1.0 abandonó el proyecto en 2012 por discordancias con el grupo en cuanto a la evolución de OAuth, publicó un artículo exponiendo sus motivos, ver

Figura 7.3: Diagrama de sequencia OAuth 2.0: Obtención de un nuevo token de acceso

referencia [6]. Sin embargo, anteriormente también se había detectado un fallo de seguridad de OAuth 1.0 ante un ataque tipo *session fixation* (fijación de sesión).

Por otro lado, en Mayo de 2014, CNET publica un artículo acerca de la seguridad de OAuth y OpenID (ver referéncia [5]) viéndose ésta comprometida por el fallo bautizado "Covert Redirect". En cambio, en la defensa, se expone que esta vulnerabilidad es conocida y es responsabilidad de cada sitio web el implementar las medidas necesarias para evitar los posibles ataques debido a *open redirector*. En la sección 4.2.4 del RFC 6819, OAuth 2.0 Threat Model and Security Considerations [11] puede encontrarse información más detallada.

Tras este análisis, la empresa decide continuar eligiendo OAuth 2.0, conservando las consideraciones previamente mencionadas, siguiendo así el ejemplo de los grandes gigantes de Internet.

7.2. Variaciones en la planificación

Como suele ser habitual, a lo largo del desarrollo se producen variaciones en cuanto a la planificación inicial que se había realizado. En este apartado se exponen las dificultades encontradas que han retrasado la finalización de algunas tareas y el inicio de otras.

En primer lugar, la instalación y configuración de las herramientas de control de versiones, revisión de código e integración continua, no tomó únicamente un día como había sido planificado. Fueron necesarios 3 días para tener todo el entorno listo y un día más para determinar cuál era la mejor forma de utilizar estas herramientas adecuadamente. Aún así, durante el desarrollo de la tarea " Implementar servicio de gastos comunes", se enviaron varios cambios al respositorio de revisión de código sin haber pasado los test adecuadamente. Por desconocimiento de la herramienta, al intentar solucionar el problema y corregir el código para que los test fueran aceptados, se perdió uno de los cambios que habían sido enviados. Como el cambio más reciente dependía de este, los test continuaban sin ser aceptados. Finalmente, tomando un tiempo para investigar como solucionar el problema, fue posible corregir el error y continuar con el desarrollo. Sin embargo, esto ya había supuesto 3 días extra respecto a la planificación.

En segundo lugar, la integración del *plug-in* OAuth fue sencilla, las pruebas manuales eran satisfactorias pero los test de CakePHP escritos para ser ejecutados por la herramienta de integración continua, no se ejecutaban correctamente, no se conseguía la autorización a los recursos protegidos. Tras revisar exhaustivamente en varias ocasiones el código, se detectó un error en la petición que se llevaba a cabo en el test y que había pasado desapercibida en la mayoría de los test hasta el momento. Así pues, fue necesario reescribir algunas partes del código corrigiendo el fallo en la ejecución de la petición. Esto supuso 2 días más de lo previsto en la planificación.

En tercer lugar, la integración del servicio de SMS se retrasó ligeramente debido a un problema en la codificación del mensaje enviado al servicio externo. Como causa de este error, el servicio externo no hacía efectivo el envío del SMS al usuario. Fue necesario un día extra de pruebas para encontrar el fallo y poder solucionarlo.

Finalmente, debido a intereses empresariales, debió establecerse un nuevo hito para subir

la aplicación a Google Play Store antes de lo previsto. Por ello, fue necesario terminar las funcionalidades básicas de la aplicación y validarlas, dejando de lado la integración con el monedero de PangoPay puesto que era menos prioritario. Así pues, la integración con el cliente y las pruebas conjuntas tomaron gran parte del tiempo restante de la estancia por lo que la integración con PangoPay no fue posible. También tras la definición de este nuevo hito, se aplazó la implementación de la gestión de tareas y quedó prevista para la siguiente versión de la aplicación que se lanzará a Google Play Store.

7.3. Validación y pruebas

Para llevar a cabo los casos de prueba, CakePHP cuenta con soporte integrado para que algunas tareas sean más cómodas para el programador. Además, estas pruebas están integradas con PHPUnit. Puede configurarse el proyecto de CakePHP para hacer uso de una base de datos especial cuando se lleven a cabo las pruebas, es decir, soporta el uso de una base de datos de pruebas de forma automática. Las clases que implementan los casos de prueba, se deben colocar en la ruta "Test/Case"dentro del proyecto CakePHP. En este caso creamos una clase para cada uno de los controladores, de forma que ejecutarán pruebas sobre cada uno de los métodos que estos contienen, comprobando que las respuestas a las peticiones son consistentes y que el acceso a los datos sólo se permite a aquellos que están autorizados. Para facilitar más la tarea, CakePHP cuenta con unos ficheros especiales llamados "Fixtures" que permiten al programador escribir una serie de datos que se insertarán en la base de datos de prueba justo antes de ejecutar cada uno de los métodos del caso de prueba de cada controlador. De esta forma, es mucho más sencillo saber cuál sería la respuesta que deberíamos esperar para una petición en concreto al servicio. Como los test fueron escritos al mismo tiempo en que se implementaban las funciones del servicio, al finalizar cada tarea de implementación, toda la batería de pruebas estaba lista para ser lanzada y comprobar que todo sucedía según lo esperado.

Por último, en el momento de lanzar la aplicación a Google Play Store, se trabajó unos días de forma conjunta con el equipo Android para integrar el servicio con la aplicación cliente. A lo largo de este trabajo en tándem, se pudo comprobar también si el servicio proporcionaba las respuestas apropiadas al cliente, pudiendo llevar a cabo pequeñas modificaciones si se consideraba necesario. Una vez se subió la aplicación a Google Play Store , se dio acceso únicamente a un grupo de *beta testers* para que la utilizaran durante unos días e informaran de cualquier error o posible mejora antes de dar acceso a todo el público. Gracias a esta etapa de pruebas realizada por un grupo reducido de usuarios, se detectaron algunos errores en la parte servidor, principalmente en los métodos de login y gestión de cuentas. Además, también se identificaron mejoras potenciales en la interfaz y la forma en la que los usuarios prefieren consultar la información.

8 Conclusiones

A través del trabajo llevado a cabo en este proyecto dentro de la asignatura *EI1054 - Prácticas Externas y Proyecto Final de Grado*, se han ampliado los conocimientos adquiridos a lo largo de los cursos de la carrera, además de haber tenido la oportunidad de aplicar en un nuevo ámbito algunos de los conceptos y habilidades ya adquiridos por mí.

Por otra parte, la estancia en la empresa proporciona un nuevo punto de vista a los proyectos que se llevan a cabo, así como a las consideraciones que se han de tener en cuenta a la hora de planificarlos. La libertad que me ha prestado la empresa a la hora de investigar y estudiar nuevas tenologías, desconocidas por mí a priori, y para organizar las tareas segun mi criterio, me ha permitido probar mis capacidades en un entorno de producción real. La cooperación, las reuniones y la toma de decisiones conjuntamente con otros empleados de la empresa me ha ayudado a integrarme en el entorno empresarial y a adoptar unos hábitos de trabajo para desenvolverme de la mejor forma y conseguir la consecución de los objetivos. Como consecuencia de esto último, las capacidades de comunicación con equipos multidisciplinares también han mejorado.

Las claves de mi motivación durante el desarrollo de este proyecto han sido el conocer la aplicación de las TIC en un nuevo sector de negocio, las aplicaciones financieras, así como conocer la relevancia y repercusión que tienen ciertas decisiones tomadas en el desarrollo para la aceptación final del producto por parte del usuario.

Así pues, a modo de síntesis del todo el trabajo expuesto en este documento, se ha desarrollado un servicio basado en arquitectura REST partiendo de los *mockups* de FlatCrew, definiendo los requisitos y casos de uso, colaborando y aunando fuerzas con otros trabajadores de la empresa, aprendiendo a utilizar nuevas herramientas de soporte al desarrollo como Jenkins y Gerrit e incluyéndolas en la rutina de trabajo, además de haber aprendido a emplear y haber aplicado el framework de desarrollo CakePHP para crear servicios en PHP. En cuanto a la consecución de las tareas y los resultados en terminos de producción, al termino de la estancia se ha podido presentar la aplicación cliente para Android integrada exitosamente con la parte del servidor *FlatCrew Server*, la que corresponde al desarrollo realizado por mí y descrita en este documento. En cambio, algunas de las tareas propuestas en la planificación inicial no han podido completarse debido a dificultades inesperadas en otras tareas y cambios en los intereses empresariales.

Por último, el sistema desarrollado en conjunto, *FlatCrew Server* y *FlatCrew Client*, es la consecución de una idea latente en la dirección de la empresa y una nueva oportunidad para acceder a otro sector del mercado de las aplicaciones financieras. Sin embargo, quedan abiertas muchas posibilidades de trabajo futuro para convertir al producto actual en algo mucho más completo. Algunos aspectos interesantes podrían ser: completar la integración con PangoPay, crear un cliente Web o estudiar posibles extensiones de funcionalidades como la gestión de tareas empleando *tags* en el mundo real (NFC).

9 Agradecimientos

El progreso y desarrollo de este proyecto no hubiera sido posible sin la participación de mi familia y amigos que han sabido brindarme el apoyo e iniciativa necesarios para seguir trabajando con ilusión en los momentos más difíciles, ayudándome a crecer como persona y enseñándome a exprimir la alegría de cada instante.

En segundo lugar, quiero hacer llegar mi agradecimiento a los miembros del grupo de investigación UJI Geotec [15] porque han formado una parte fundamental en mi crecimiento y desarrollo profesional.

Por otra parte, agradezco a mi tutor de estancia en prácticas su comprensión y respaldo. A PaynoPain Solutions S.L, haberme adoptado como parte de la empresa a lo largo de la estancia en prácticas. Al supervisor y a los empleados, su colaboración. Concretamente a Juanjo Chust, Juan Ezquerro y Nelson Marco que consiguieron proporcionarme el soporte necesario para ampliar mis conocimientos y guiarme para llevar a cabo el proyecto de la mejor forma.

Finalmente quiero agradecer, especialmente, a Aaron Martinez, también en calidad de amigo, su interés e implicación en mi progreso tanto personal como profesional.

Bibliografía

[1] Bulk SMS - Internet based SMS messaging services. http://www.bulksms.com/. [Consulta: 16 de Septiembre de 2014].

[2] CakePHP Components. http://book.cakephp.org/2.0/en/controllers/components.html. [Consulta: 26 de Julio de 2014].

[3] CakePHP DataSources. http://book.cakephp.org/2.0/en/models/datasources.html. [Consulta: 26 de Julio de 2014].

[4] CakePHP OAuth2 Server Plugin - GitHub thomseddon. https://github.com/thomseddon/cakephp-oauth-server. [Consulta: 16 de Septiembre de 2014].

[5] CNET - Serious security flaw in OAuth, OpenID discovered. http://www.cnet.com/news/serious-security-flaw-in-oauth-and-openid-discovered/. [Consulta: 16 de Septiembre de 2014].

[6] Eran Hammer - OAuth 2.0 and the Road to Hell. http://hueniverse.com/2012/07/26/oauth-2-0-and-the-road-to-hell/. [Consulta: 16 de Septiembre de 2014].

[7] Fairshare App. http://www.getfairshare.com/. [Consulta: 5 de Agosto de 2014].

[8] FlatCrew. http://www.flatcrew.com/. [Consulta: 27 de Julio de 2014].

[9] Gerrit Code Review. https://code.google.com/p/gerrit/. [Consulta: 7 de Agosto de 2014].

[10] Jenkins continuous integration server. http://jenkins-ci.org/. [Consulta: 7 de Agosto de 2014].

[11] OAuth 2.0 Threat Model and Security Considerations. http://tools.ietf.org/html/rfc6819#section-4.2.4. [Consulta: 16 de Septiembre de 2014].

[12] PangoPay WebSite. http://www.pango-pay.com/. [Consulta: 26 de Julio de 2014].

[13] PaynoPain WebSite. https://www.paynopain.com/. [Consulta: 26 de Julio de 2014].

[14] Splitwise App. https://www.splitwise.com/. [Consulta: 5 de Agosto de 2014].

[15] UJI Geotec. http://www.geotec.uji.es. [Consulta: 6 de Octubre de 2014].

[16] Wee-Kheng Tan, Yi-Der Yeh, Shin-Jia Chen, Yu-Cheng Lin, y Chia-Yu Kuo. How Consumers Assess Product's Features?: A Case Study of Product Features of Smartphone. http://www.wseas.us/e-library/conferences/2012/Vouliagmeni/MMAS/MMAS-20.pdf. [Consulta: 2 de Agosto de 2014].

Índice de tablas