eman ta zabal zazu
Universidad del País Vasco
Euskal Herriko Unibertsitatea
INFORMATIKA FAKULTATEA
FACULTAD DE INFORMÁTICA

Grado en Ingeniería Informática

Ingeniería del Software

Trabajo de Fin de Grado

Aplicación para smartphones de DIPC

Autor

Urko Lekuona Rico

2019

eman ta zabal zazu
Universidad del País Vasco
Euskal Herriko Unibertsitatea
INFORMATIKA FAKULTATEA
FACULTAD DE INFORMÁTICA

Grado en Ingeniería Informática

Ingeniería del Software

Trabajo de Fin de Grado

Aplicación para smartphones de DIPC

Autor

Urko Lekuona Rico

Directora

Ana Sánchez Ortega

Resumen

Este Proyecto de Fin de Grado desarrollado se ha realizado en conjunto con el centro de investigación Donostia International Physics Center (DIPC), en concreto con el departamento Centro de Cálculo. El objetivo es facilitar a los investigadores del centro una manera de acceder a los servicios que hasta ahora solo tenían disponibles desde sus equipos de trabajo (incluso proveer alguno nuevo) y dar a los administradores una forma de gestionar el sistema y el trabajo de los investigadores, de nuevo sin necesitar estar delante de un equipo. En una época donde las tecnologías móviles están cada vez más a la alza, se ha decidido solucionar esta necesidad implementando y poniendo en producción una aplicación para *smartphones* propia del centro.

Para el desarrollo de la aplicación se ha escogido una arquitectura en cliente (*front-end*) - servidor (*back-end)*, donde la parte del cliente se ha implementado con Ionic, que a su vez hace uso de herramientas como Angular y Cordova, y la parte del servidor se ha compuesto de una API web utilizando el *framework* CodeIgniter y una base de datos MariaDB, además de los propios servidores que forman parte de la infraestructura del DIPC, que son claves para el funcionamiento de la aplicación.

Índice general

Índice de figuras

Índice de tablas

1. CAPÍTULO

Introducción

El DIPC (Donostia International Physics Center) es un centro de investigación cuyo objetivo es el de promocionar y catalizar el desarrollo al más alto nivel de la investigación básica y básica-orientada en ciencia de materiales. Es una institución abierta y ligada a la Universidad del País Vasco, que sirve como plataforma de internacionalización de la ciencia básica en el País Vasco en el campo de los materiales. Aglutina un alto número de investigadores, alrededor de 200, procedentes de diferentes países.

Muchos de estos investigadores necesitan realizar cálculos computacionales de gran calibre, por lo que no les es suficiente con un equipo de sobremesa. Para solventar esta necesidad, el centro alberga instalaciones de computación paralela y serial, y les ofrece a los investigadores una cuenta con la que acceder a los supercomputadores y trabajar con ellos.

Estos supercomputadores están alojados en el Centro de Procesamiento de Datos (CPD) del centro. Cada supercomputador está formado por una máquina de acceso, múltiples nodos de cálculo y un disco donde se almacena la información de sus trabajos e investigaciones, llamado *scratch* (nombre común de este tipo de discos). Además, estos supercomputadores tienen añadidos otros servicios extra, como el montaje de un disco que alberga los directorios *home* o el montaje del servidor LDAP que gestiona las cuentas.

Como los supercomputadores son formaciones de múltiples máquinas también suelen ser denominados *cluster*s. Cada *cluster* tiene un sistema de colas que es el encargado de administrar todos los trabajos que son ejecutados por los investigadores. Al ejecutar un trabajo, es necesario reservar un determinado espacio de memoria, un tiempo de ejecución

límite, y una cantidad de nodos de ejecución. En base a estos parámetros, se consumirán más o menos créditos del total asignado anualmente a cada investigador. Si se alcanza el máximo de créditos asignado, el investigador no será capaz de ejecutar nuevos trabajos hasta que se reinicien los créditos, por lo que es importante conocer bien los recursos consumidos hasta el momento, para poder estimar cuántos recursos asignar a los nuevos trabajos. Por ejemplo, muchos de los trabajos suelen tardar varios dás (algunos incluso semanas) en finalizar. Ser capaz de ajustar el tiempo previsto es crucial, ya que si nos quedamos cortos el trabajo se para al alcanzar el límite (puede suponer haber malgastado completamente todo el tiempo y los créditos utilizados, aunque dependerá del trabajo), pero si nos sobra mucho tiempo habremos desperdiciado créditos.

La idea del proyecto surge por la necesidad de agilizar el acceso a los datos de los clústeres del DIPC y hacerlo de manera más cómoda. Hasta la fecha, la única forma en la que un investigador podía acceder a los clústeres y consultar el estado de sus trabajos o de ponerse en contacto con el Centro de Cálculo era estando de forma física en su puesto de trabajo y accediendo a su equipo personal. Esto se debe a que tanto los equipos de trabajo de los investigadores como los servidores a los que acceden se encuentran dentro de la red interna de la Universidad Pública del País Vasco (en adelante UPV/EHU), que solo permite el acceso desde el exterior mediante algunas máquinas específicas, como los servidores VPN o los servidores web. Uno de estos servidores web, el del centro, se va a usar en este proyecto como método de conexión. El proyecto consiste en implementar una aplicación para *smartphones* que elimine esa restricción y permita a un investigador acceder a los servicios que les proporciona el DIPC desde cualquier parte (la aplicación se conectará al servidor web para extraer la información que necesite).

El proyecto se ha realizado en colaboración con el Centro de Cálculo del DIPC, departamento encargado de gestionar los clústeres de la empresa, administrar las cuentas de usuarios y sus equipos personales, además de controlar todo el apartado informático (como la página web o el control de la red).

El propio Centro de Cálculo también tiene interés en el desarrollo de la aplicación, pues se pretende crear un apartado exclusivo para los administradores que les permita beneficiarse de la aplicación. Así, está pensado que ciertas tareas que se realizan desde los equipos de la empresa se redirijan completamente a la aplicación, como el proceso de crear nuevas cuentas de usuarios o el ponerse en contacto con los usuarios (actualmente se hace mediante correo).

Este documento tiene como objetivo explicar el desarrollo del proyecto. La estructura del

mismo es la siguiente:

- Capítulo 2: Este capítulo explicará el proceso de planificación del proyecto, listando los objetivos, alcance y exclusiones, proporcionando una EDT y un análisis de riegos.
- Capítulo 3: En este capítulo se estudian las diferentes herramientas que se han considerado usar y las escogidas finalmente, comparando las diferentes opciones y argumentando las razones para escoger una de las candidatas.
- Capítulo 4: Capítulo donde se hablará sobre el Análisis del proyecto, listando los requisitos e identificando los casos de uso que se vayan a implementar más adelante.
- Capítulo 5: Capítulo centrado en el Diseño del proyecto. Se realizarán diagramas como el de casos de uso, un modelo relacional de la base de datos y diagramas de secuencia por cada caso de uso. Además, se diseñarán un esquema de la arquitectura del sistema y un esquema de la navegación de la aplicación.
- Capítulo 6: Aquí se explicará la implementación de cada uno de los casos de uso del capítulo 4, comentando también los diferentes problemas que se hayan encontrado y las soluciones que se han planteado.
- Capítulo 7: Este capítulo servirá para explicar las pruebas que se han realizado y los resultados obtenidos. También se hablará sobre el despliegue del proyecto.
- Capítulo 8: Se hablará sobre las gestión del proyecto que se haya realizado, comparando los valores estimados en el capítulo 2 con los reales.
- Capítulo 9: Capítulo que recoge las conclusiones del proyecto y las propuestas de mejora del mismo.
- Bibliografía: Listado de enlaces y referencias externas que se hayan mencionado en la memoria.
- Anexos: Apartado de la memoria donde se incluirán diferentes documentos de interés para el proyecto.

2. CAPÍTULO

Objetivos y Alcance

En este capítulo se van a explicar los objetivos del proyecto, así como su alcance y las exclusiones que se han creído oportunas. Estas decisiones se han tomado en conjunto con el DIPC, en una reunión donde se expusieron los objetivos de la aplicación y se acordó el alcance de las funcionalidades de la misma.

Además de lo comentado previamente, se realizará un análisis de riesgos que puedan alterar el alcance del proyecto, retrasar su avance o incluso cancelarlo. Junto a este análisis también se añadirá un apartado de soluciones para los riesgos, en caso de haberlas.

Por último, se va a agregar una estructura de descomposición del trabajo (EDT/WBS) del proyecto, una estimación de horas de dedicación y un diagrama de Gantt.

2.1. Objetivos

La aplicación del DIPC tiene como objetivo primario ser la aplicación oficial del Centro de Cálculo. Esto significa que todos sus participantes la usarán para diversas tareas. Esto incluye tanto a investigadores invitados como a la propia plantilla de la empresa.

Respecto a los investigadores, ya sean invitados o de la plantilla, se ha pensado ofrecerles acceso a información de los servidores de cálculo del centro. Esto podrá facilitarles el seguimiento de sus trabajos y sus cuotas en dichos servidores, ya que podrán consultarlo desde cualquier sitio, sin tener que estar sentados delante de su equipo de trabajo. Las funcionalidades que se han acordado implementar son: poder consultar el número de créditos utilizados y el espacio del disco ocupado en los clústeres, el estado de sus trabajos en la cola, el informe DCRAB[1] de los trabajos en ejecución y contactar con los empleados del Centro de Cálculo para notificar problemas o solicitar ayuda. Las funcionalidades sobre el estado de la cola y el informe DCRAB ya se implementaron durante un periodo de prácticas previo al inicio de este proyecto, pero debido a la nueva estructura de la aplicación (además de las nuevas herramientas que se van a utilizar, ver capítulo 3) habrá que reimplementarlas desde cero.

Para los empleados del Centro de Cálculo se ha decidido centralizar varias tareas en la aplicación. El objetivo es reducir el tiempo de realización de las mismas, hasta el punto de automatizar algunas de ellas, y poder hacerlo fuera de horas de trabajo. Las funcionalidades que se pretenden implementar son: validar cuentas de trabajo de investigadores (automatizaría el proceso de generación de la cuenta en los clústeres), leer y contestar consultas o avisos de los investigadores, mostrar el estado de la instalación (tanto de los clústeres como de la micro-informática) y ver estadísticas de uso.

2.2. Alcance

Se realizará una aplicación para smartphones capaz de cumplir los objetivos mencionados previamente, que implica también ser capaz de conectarse a los servidores de cálculo, servidor web y servidor de almacenamiento CEPH del DIPC desde cualquier lugar donde haya acceso a Internet, ya sea mediante Wi-Fi o datos móviles.

La idea original era la implementación de una aplicación para el sistema operativo An-

[1] Tecnología desarrollada por el DIPC para ver estadísticas sobre trabajos en ejecución.

droid, que se decidió durante el periodo de prácticas de forma arbitraria porque el autor tenía experiencia previa en el desarrollo de aplicaciones para este sistema. Sin embargo, dejar al sistema operativo iOS fuera del alcance supondría marginar a prácticamente la mitad de los usuarios. Por esta razón, se tendrá que considerar el desarrollo de una aplicación web o híbrida[2] (originalmente la aplicación era nativa) y una vez decidido el método de desarrollo, hacer un estudio de viabilidad para la versión de iOS.

Como metodología de trabajo se ha decidido utilizar el Proceso Unificado de Desarrollo de Software. Esta metodología está pensada para adaptarse al proyecto al que se aplique y provee al proyecto de ventajas considerables. En el capítulo 3 se realiza una comparativa en la que se exponen los puntos fuertes de este marco de trabajo.

Sobre el idioma de la aplicación, se ha decidido que la aplicación esté soportada únicamente en inglés. Esta decisión se ha tomado en la reunión que se ha tenido antes de empezar el proyecto. El DIPC considera que, como el resto de las herramientas con las que se trabaja en la empresa, sobre todo por parte de los investigadores, están en inglés, no es necesario traducirla a ningún otro idioma. Muchos investigadores son del extranjero y todos tienen un buen dominio del inglés.

2.3. Requisitos

Al tener que ser utilizada por diferentes tipos de usuarios, la aplicación debe ser fácilmente accesible y utilizable por cualquiera. Además, debe ser compatible con la mayoría de teléfonos que haya en el mercado, para excluir a la menor cantidad de usuarios posibles.

La aplicación ha de funcionar correctamente y de forma rápida. Una aplicación de uso diario que tiene como objetivo hacer más cómodas las tareas de sus usuarios no puede ser lenta. Por lo tanto, no se admitirán tiempos de carga desmesurados, problemas de conexión constantes, fallos frecuentes, o cualquier otra inconveniencia que pueda aumentar el tiempo de uso necesario para realizar una tarea.

Al tener que almacenar datos personales sobre los usuarios de la aplicación para poder implementar algunas funcionalidades (guardar los datos de un formulario de registro que contienen correo electrónico, número de teléfono y, a veces, dirección de un investigador, por ejemplo), se tendrá que hacer de acuerdo a lo establecido en la Ley Orgánica 3/2018,

[2] Véase el capítulo 3.

de 5 de diciembre, de Protección de Datos Personales y garantía de los derechos digitales[3] (en adelante LOPD)[4].

Por último, la apariencia de la aplicación ha de estar actualizada al estándar del resto de aplicaciones del mercado. Para poder hacer que esta aplicación acabe siendo utilizada por todos los participantes de la empresa, debe ser apetecible y con una curva de aprendizaje pequeña. Esto quiere decir que el estilo y la estructura deberá adecuarse al de aplicación similares o imitar a aplicaciones de mayor envergadura.

2.4. Exclusiones

Se excluye del alcance del proyecto el soporte en algún idioma adicional de la aplicación. Las razones se han comentado en el apartado sobre el alcance. Además, no se baraja la posibilidad de añadir idiomas a mitad del ciclo de desarrollo del proyecto, ya que a esas alturas un cambio así supondría no solo la generación de una traducción del texto de la aplicación hasta la fecha, sino también la reescritura de gran parte del código.

También se excluye del proyecto el soporte y mantenimiento de la aplicación una vez terminado el periodo del proyecto y realizada la defensa del mismo. En el ciclo de vida de un programa software, la fase de mantenimiento es la más duradera de todas. Mantener la aplicación al día en cuanto a seguridad, corregir errores o implementar nuevas funcionalidades son algunas de las características de esta fase, que podrían llegar a aumentar la vida del proyecto en gran medida.

2.5. EDT/WBS

La figura 2.1 representa la estructura de descomposición de trabajo del proyecto. La tarea 1.1.1 se repetirá tres veces, que es el número de iteraciones en el que se va a dividir el trabajo de desarrollo.

[3] https://www.boe.es/boe/dias/2018/12/06/pdfs/BOE-A-2018-16673.pdf
[4] https://www.boe.es/boe/dias/2018/12/06/pdfs/BOE-A-2018-16673.pdf

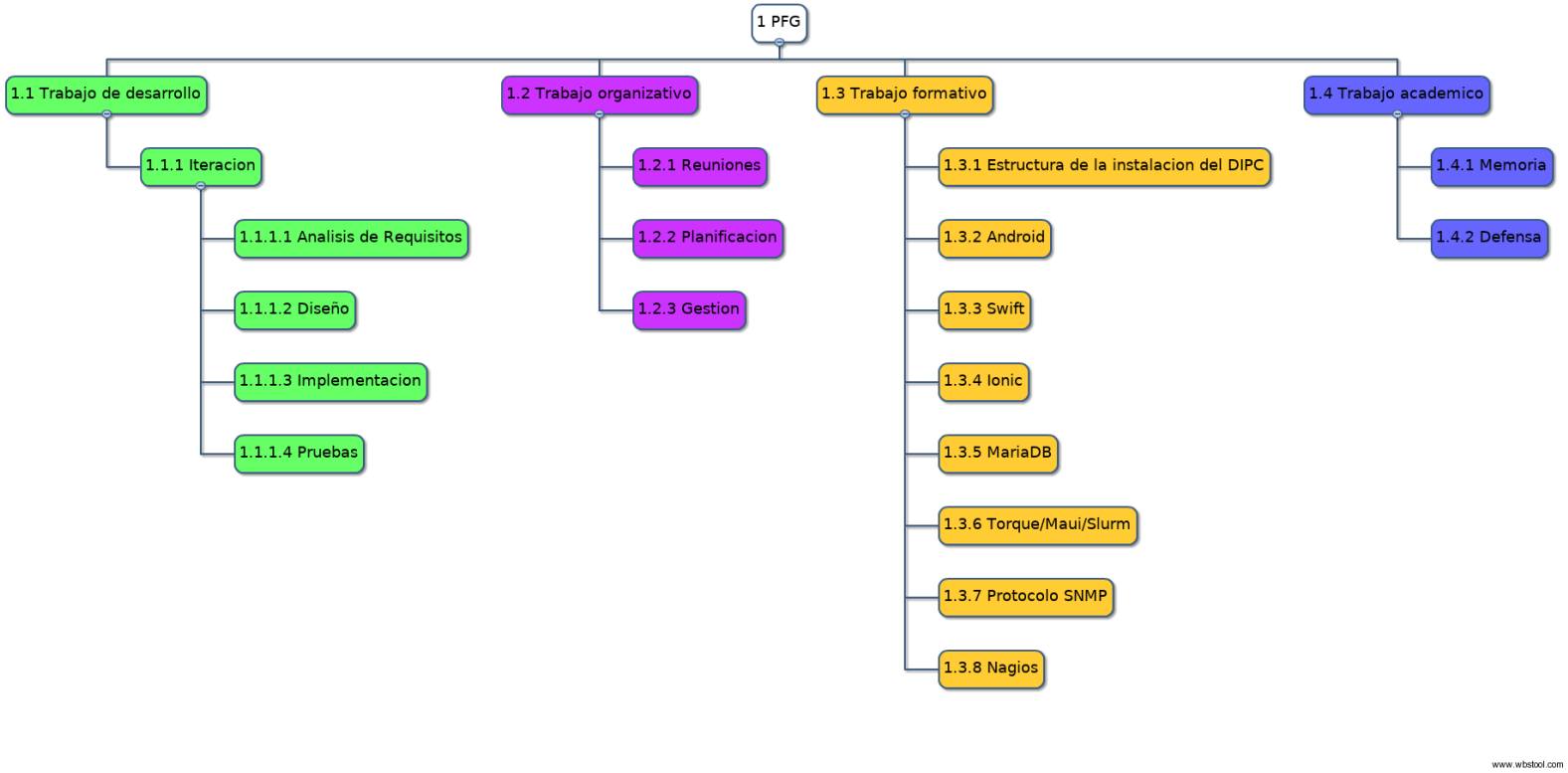

Figura 2.1: Estructura de Descomposición del Trabajo.

2.6. Previsión de horas

La realización de este proyecto supone 12 créditos del plan de estudios del autor. Según el Sistema Europeo de Transferencia y Acumulación de Créditos, cada crédito se puede traducir en entre 25 y 30 horas. Con estos datos se puede calcular que la realización de un Trabajo de Fin de Grado debería suponer entre 300 y 360 horas de trabajo.

Para que este proyecto sea viable, la previsión de horas de dedicación necesarias al mismo deberá estar cerca de las mencionadas con anterioridad. Para hacer el cálculo, se separarán las tres tareas principales del EDT y se hará una previsión sobre cada una de sus subtareas.

2.6.1. Trabajo de desarrollo

El trabajo de desarrollo, en el EDT, se estructura en tres iteraciones. En cada una de estas iteraciones se trabajará en diferentes casos de uso, pero todas ellas tienen la misma estructura. En la siguiente tabla se hace referencia al tiempo de trabajo total entre las tres iteraciones.

Fase	**Horas**
Análisis de Requisitos	15
Diseño y Arquitectura	15
Implementación	170[5]
Pruebas	10
Mantenimiento[6]	0
Total	**210**

Tabla 2.1: Estimación del tiempo necesario para el desarrollo.

2.6.2. Trabajo organizativo

El trabajo organizativo agrupa todo el trabajo que se necesite realizar para la gestión del proyecto.

[5] Dependiendo de la herramienta seleccionada, podrá variar.
[6] En la sección de exclusiones se explica que no se realizará ningún trabajo de mantenimiento.

Fase	Horas
Reuniones	5
Planificación	15
Gestión	15
Total	**35**

Tabla 2.2: Estimación del tiempo necesario para la organización.

2.6.3. Trabajo formativo

Este tipo de trabajo incluye el tiempo aprendizaje de las diferentes herramientas y tecnologías que se van a utilizar, así como el tiempo necesario en estudiar alternativas y hacer estudios de viabilidad.

Fase	Horas
Estructura de la instalación del DIPC	3
Android	10
Swift	10
Ionic	20
MariaDB	5
Protocolo SNMP	5
Torque/Maui/Slurm	2
Nagios	5
Total	**60**

Tabla 2.3: Estimación del tiempo necesario para la formación.

2.6.4. Trabajo académico

Todo el trabajo que esté orientado a la generación de documentación necesaria para la entrega del proyecto se considera trabajo académico. La tarea de *Memoria* incluye también la documentación de la sección *Trabajo de desarrollo*, ya que parte de la realización de la memoria será la documentación del código implementado[7].

[7] Véase capítulo 6.

Fase	Horas
Memoria	40
Defensa	10
Total	**50**

Tabla 2.4: Estimación del tiempo necesario para labores académicas.

2.6.5. Total

El total supone la suma de las estimaciones realizadas y sirve para calcular el tiempo previsto necesario para cumplir con los objetivos del proyecto.

Trabajo	Horas
Trabajo de desarrollo	210
Trabajo organizativo	35
Trabajo formativo	60
Trabajo académico	50
Total	**355**

Tabla 2.5: Estimaciones de cada trabajo y su suma.

Después de las estimaciones realizadas, se comprueba que el total está dentro o se acerca al rango de horas necesario para la realización de un TFG. Por lo tanto, se puede asumir que es viable en cuanto a carga de trabajo.

2.7. Diagrama de Gantt

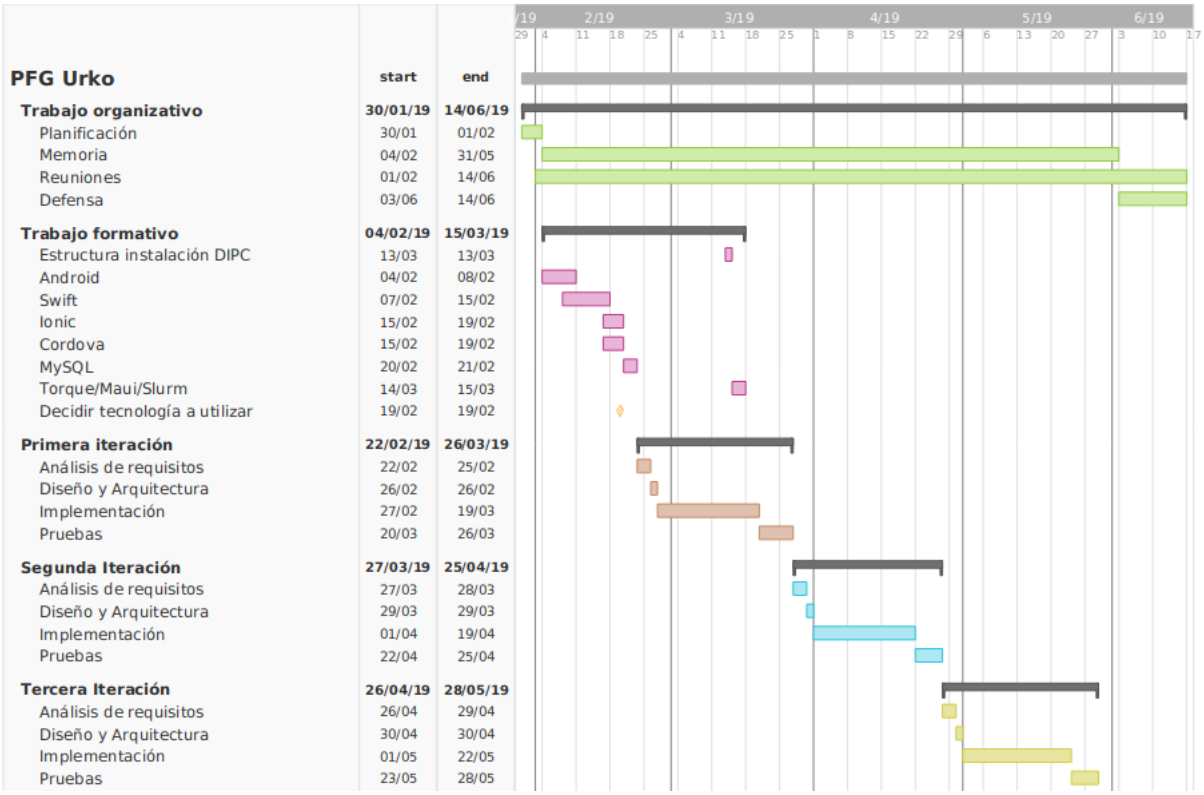

Figura 2.2: Diagrama de Gantt.

2.8. Análisis de riesgos

Respecto a los riesgos del proyecto, hay varios factores que pueden hacer que la entrega del proyecto prevista se retrase, haya que modificar el alcance o incluso haya que cancelar el proyecto. En esta sección, se elaborará una lista de estos riesgos y se creará un plan de contingencia para intentar evitar que ocurran y disminuir su impacto en caso de no haber podido evitarlos.

Los factores de riesgo identificados son:

- Una organización inadecuada o una mala planificación del proyecto puede hacer que las fechas de los hitos no estén bien posicionadas o que la calidad del proyecto se vea disminuida y haya que emplear un trabajo extra en mejorarlo. Este riesgo

existe por la falta de experiencia del estudiante realizando proyectos de amplitud similar a este.

- Perder información relacionada con el proyecto. Esto incluye la memoria, notas personales del estudiante, correos con el tutor o la empresa y ficheros de código de la aplicación. Dependiendo de la importancia del material extraviado y de la altura en la que se esté dentro del proyecto, este riesgo puede suponer una pequeña demora en el progreso del proyecto o puede suponer tener que cancelarlo.

- Al tener que trabajar con nuevas tecnologías de forma independiente, en un entorno de trabajo nuevo y en una organización jerárquica diferente (tutor <->estudiante <->empresa), existen riesgos como el de necesitar emplear demasiado tiempo en el aprendizaje o el de caer en malas costumbres de comunicación con alguna de las partes.

- Para ser capaces de cumplir con todos los objetivos que se han propuesto, será necesario que el centro ponga a disposición del proyecto ciertos recursos. Estos incluyen, pero no están limitados a, una cuenta de cálculo para tener acceso a la cola de trabajos y otros servicios, un equipo con acceso a la red del centro, máquinas donde poder montar los servidores, incluso acceso al servidor donde se alberga la página web del centro (es donde se albergará también nuestra API). Algunos de estos recursos puede que tengan más sentido después de leer los capítulos 3, 4 y 5. Hay que mencionar que la mayoría de estos recursos es poco probable que no se faciliten (crear una cuenta de cálculo suele tardar alrededor de 10 minutos). El más peligroso de ellos es el acceso al servidor web donde se planea albergar la API de la aplicación, ya que está siendo desarrollado por una empresa subcontratada y que se complete a tiempo no está en manos del DIPC.

- La salud y disponibilidad tanto del estudiante como del director y del representante de la empresa también habrá que tenerlas en cuenta, al ser un proyecto de una duración elevada. Esto comprende desde enfermedades a problemas de disponibilidad entre las partes, ya sea por vacaciones o por cuestiones laborales.

2.8.1. Plan de contingencia

En respuesta a los riesgos listados anteriormente, se han establecidos las siguientes pautas para cada uno de ellos. Así, se pretende establecer un protocolo de actuación para el

proyecto, con el objetivo de evitar y mitigar el daño que puedan ocasionar los riesgos. Las medidas establecidas son las siguientes:

- Para solucionar el problema de la mala organización y/o planificación, se ha decidido realizarla bajo constante supervisión de la tutora, que es una persona más experimentada y que puede guiar al alumno en este tipo de cuestiones. Además, establecer fechas para los hitos con un margen de precaución puede ayudar a cumplirlos con mayor facilidad.

- Respecto a la posible pérdida de información, la forma más sencilla de evitarlo es realizando copias de seguridad con frecuencia. Se ha decidido que estas copias estarán almacenadas en varios lugares, para evitar problemas como la pérdida de las copias de seguridad. Una copia estará en el equipo de trabajo habitual, otra en una plataforma de almacenamiento en la nube (como Google Drive, Dropbox o OneDrive) y otra en un dispositivo de almacenamiento externo. Estas copias se realizarán con diferentes frecuencias. La copia del equipo de trabajo se realizará a diario, al finalizar cada sesión de trabajo o antes de implementar un cambio grande en algún elemento del proyecto. Las copias que vayan a la plataforma online se realizarán cada vez que se haya producido un cambio relevante y al final de cada semana. Las copias que se hagan en la memoria externa serán semanales.

- Prever cuánto tiempo va a ser necesario para el aprendizaje de las nuevas tecnologías es complicado. Aun así, se ha decidido establecer una rutina de trabajo durante las fases de aprendizaje que garanticen su avance de manera adecuada. Esto incluye horas extras de trabajo si se estiman necesarias y cursos online intensivos.

- Dependiendo del recurso que no se provea, las repercusiones en el proyecto serán diferentes. Por ejemplo, si no se facilita una cuenta de cálculo, será prácticamente imposible alcanzar los objetivos propuestos. En cuanto al plan de contingencia, no se puede hacer gran cosa, ya que dependen únicamente del DIPC, aunque hay que añadir que la mayoría de recursos no son indispensables y sin ellos el proyecto sigue teniendo envergadura suficiente.

- Añadiendo márgenes a las fechas de los hitos se ha resuelto en cierta medida también el problema de la disponibilidad y la salud de los participantes del proyecto. Además, al planificar los hitos se intentarán recoger las fechas que se sepa que alguno de los participantes no vaya a estar disponible y se modificarán los hitos en consecuencia. Como medida extra, para favorecer la comunicación entre las partes,

se mantendrán reuniones periódicas para hablar sobre el progreso del proyecto y tomar las medidas que sean necesarias.

3. CAPÍTULO

Metodología de trabajo y herramientas utilizadas

En este capítulo se hablará sobre las diferentes tecnologías que se hayan usado o estudiado para generar el producto y para realizar la documentación necesaria. Elegir las herramientas adecuadas para realizar el proyecto es un factor clave para que la aplicación acabe siendo un producto satisfactorio.

Primero se hablará sobre la metodología de trabajo utilizada durante el ciclo de vida del proyecto. Se compararán las diferentes alternativas que se han valorado y se explicará en detalle la forma en la que funciona la seleccionada.

A continuación se hará un estudio para comparar entre el desarrollo de la aplicación de forma nativa o como una aplicación híbrida. Es importante tomar esta decisión antes de continuar con la elección de tecnologías porque, en consecuencia del resultado del estudio, cambiarán algunas de las herramientas de desarrollo utilizadas.

Al finalizar el estudio, se hará un análisis en detalle de las herramientas y tecnologías que se hayan decidido utilizar como conclusión del mismo, así como del resto de herramientas y tecnologías que jueguen un rol crucial en el resultado del producto.

Por último, se hablará sobre algunas herramientas que se han tenido que utilizar o estudiar pero no han tenido un papel muy relevante o no han requerido mucho trabajo para utilizarlas. Sus descripciones serán concisas, mencionando solo los puntos que se consideran relevantes para comprender correctamente su función dentro del proyecto.

3.1. Metodología de trabajo

La metodología de trabajo es un factor con gran relevancia dentro del desarrollo de un proyecto. Puede determinar la calidad del producto final, el tiempo necesario de desarrollo e incluso el coste. Dentro de la Ingeniería del Software hay varios tipos diferentes que son utilizados.

Uno de estos tipos son las llamadas ”metodologías ágiles”, que están diseñadas teniendo siempre en mente al cliente. Entre ellas, una de las más conocidas es SCRUM. Esta metodología se centra en ser capaz de adaptarse a los cambios que el cliente imponga sobre el producto y en la comunicación entre los diferentes miembros del equipo. En la asignatura Ingeniería del Software del grado ya trabajamos con esta tecnología y, por lo tanto, me sentiría cómodo utilizándola en caso de ser elegida.

La alternativa a las metodologías ágiles sería el Proceso Unificado de Desarrollo Software o simplemente Proceso Unificado. Este marco de trabajo se centra en dividir el ciclo de vida del proyecto en diferentes fases, cada una con sus características diferentes y que son comunes entre diferentes proyectos de desarrollo de software. Estas fases son:

- **Concepción:** Fase inicial del proyecto donde se analizará el coste del mismo y se realizará la planificación.
- **Elaboración:** Generar una arquitectura del sistema y diseñar los componentes del mismo.
- **Construcción:** Implementar el producto, ajustándose en la arquitectura de la fase de Elaboración, e intentar hacerlo de la manera más práctica, rápida y barata posible. En esta fase también habrá que garantizar la calidad del producto, mediantes pruebas o versiones *alfa* o *beta*.
- **Transición:** Esta fase se centra en el despliegue del producto y todas las labores que ello conlleva, como formar a los usuarios o elaborar la documentación necesaria.

3.1.1. Comparativa

Al haberse presentado ya ambas tecnologías, la comparativa se centrará en la identificación de ventajas y desventajas de cada una de ellas, valorándolas siempre dentro del

marco del proyecto. Una vez se hayan listado estas características, se procederá a tomar la decisión final.

SCRUM

Como ya se ha comentado, SCRUM es una metodología ágil que se centra en ser capaz de adaptarse al cliente. Esta característica le permite hacer cambios de cierta relevancia en el producto sin verse el coste del proyecto muy alterado, ya que la previsión desde un principio es que el cliente cambie de opinión respecto al producto. En este proyecto, al tener que satisfacer las necesidades del DIPC y trabajar de manera conjunta con el centro, es muy posible que haya constantes alteraciones en diferentes aspectos del proyecto. Sin embargo, como el proyecto se va a realizar desde un equipo de trabajo alojado en el propio centro, la comunicación entre ambas partes será aún mayor de lo habitual en un proyecto que utilice SCRUM, independientemente del tipo de tecnología que se termine utilizando.

Otra característica importante de SCRUM es su gran énfasis en hacer retrospectiva. Por ejemplo, a diario, nada más empezar la jornada de trabajo, se realiza una reunión grupal llamada *daily scrum* en la que se listan las tareas realizadas el día anterior, se fijan objetivos para el día actual y se analizan posibles mejoras en el método de trabajo de cara al futuro. También, al final de cada iteración llamada *sprint*, se realiza una reunión en la que, además de hablar sobre el próximo *sprint*, se lista el trabajo del anterior que no se haya podido completar, se identifican los problemas que haya habido y se establecen pautas para evitarlos. Esta costumbre de mirar atrás y aprender de los errores me parece una muy buena práctica, sobre todo en un proyecto con una duración estimada de varios meses, como este.

El problema de elegir SCRUM es que está pensado para equipos de entre 3 y 9 personas. Esta metodología también se caracteriza por la comunicación entre las diferentes partes del equipo y los diferentes roles que toman parte. El *daily scrum* también tiene como objetivo poner al día al resto de integrantes del equipo del estado del trabajo. Todo esto no tiene mucho sentido realizarlo si el equipo de desarrollo va a constar solo de una persona. Se han encontrado algunas variantes[1] de SCRUM, diseñadas específicamente para trabajos individuales, que son capaces de solventar este problema con algunos cambios en la metodología, pero no deja de ser parte de la naturaleza de SCRUM trabajar en equipo.

[1] https://www.raywenderlich.com/585-scrum-of-one-how-to-bring-scrum-into-your-one-person-operation

Proceso Unificado

El Proceso Unificado está caracterizado por los siguientes puntos:

- Dirigido por casos de uso: "*Un Caso de Uso es una secuencia de interacciones entre un sistema y alguien o algo que usa alguno de sus servicios*". Los usuarios pueden hacer referencia no solo a personas, sino también a otros sistemas o máquinas que hagan uso de nuestra aplicación. Se suelen utilizar para definir los requisitos funcionales. En este proyecto, enfocarse en los casos de uso va a permitir focalizar el trabajo en los más relevantes y facilitar el análisis del total completado del proyecto.

- Iterativo e incremental: El ciclo de vida del proyecto se separa en iteraciones. Cada una de estas iteraciones aporta algo nuevo al producto. Normalmente, se separa el conjunto de casos de uso y se divide entre las diferentes iteraciones. El proyecto está dividido en cuatro fases llamadas *Concepción, Elaboración, Construcción y Transición*, y cada iteración en *Análisis de requisitos, Diseño, Implementación y Prueba*.

- Centrado en la arquitectura: La arquitectura está formada por los elementos más significativos para la construcción del modelo y necesario que tanto el equipo de desarrollo como los usuarios estén de acuerdo. En el proceso unificado de desarrollo de software no existe un solo modelo que defina las características del producto, por lo que se plantean diferentes vistas que se representan mediante los diagramas UML. Estos diagramas son necesarios para que poder comprender correctamente el funcionamiento del producto y hacer el análisis y mantenimiento del mismo más sencillo. No es necesario utilizar el Proceso Unificado para generar los diagramas UML, pero es la metodología original para la que está pensada y tienen mucho que ver al estar ambos tan ligados a los casos de uso.

Decisión

Teniendo en cuenta lo que se ha comentado, se ha decidido trabajar en el proyecto utilizando el Proceso Unificado de Desarrollo Software. Aunque no se hayan establecido unas pautas concretas para valorar los aspectos de ambas partes y la decisión final se haya tomado de forma arbitraria, se considera que con los puntos listados está justificada la elección.

Aún así, el concepto de hacer retrospectiva sobre el trabajo realizado que aporta SCRUM parece tan útil que también se aplicará al marco de trabajo. Para esto, es importante mantener constancia de los problemas encontrados en cada fase del proyecto y hacer un informe con las diferentes mejoras que se puedan aplicar.

3.2. Tipo de aplicación

En cuanto a aplicaciones para smartphone se refiere, se pueden distinguir tres tipos: aplicaciones nativas, aplicaciones web y aplicaciones híbridas. Dependiendo del tipo que se escoja para desarrollar nuestro producto, el resultado final tendrá diferentes características. Pero, más importante aún, el tiempo, coste y dificultad de desarrollo variará mucho. Las características de cada uno de los tipos son las siguientes:

- **Nativas:** Este tipo de aplicaciones son las primeras en las que se piensa y han sido las más comunes durante mucho tiempo. Son aplicaciones desarrolladas específicamente para un sistema operativo y que se publican en las tiendas de aplicaciones, como la Play Store de Android. Son capaces de utilizar el hardware del teléfono en su totalidad (micrófono, sensores, cámara, altavoces, ...) y también consiguen el rendimiento más alto de entre los tres tipos.

  Sin embargo, aunque funcionalmente sean las más completas, también tienen ciertas desventajas notables. El tiempo y coste de desarrollo de este tipo de aplicaciones suele ser el más alto. Para desarrollar de esta manera, cada sistema operativo acepta ciertos lenguajes de programación (Java o Kotlin para Android, Objective C o Swift para iOS) y unos entornos de desarrollo concretos. El mantenimiento de las aplicaciones nativas es mayor que el del resto y, si el desarrollo es para varias plataformas diferentes, habrá que generar una aplicación diferente para cada plataforma, cada una en su lenguaje y su entorno de desarrollo pero todas con las mismas funcionalidades y apariencia similar.

- **Web:** Las aplicaciones web se pueden ver como páginas web corrientes que están pensadas para ser utilizadas en un smartphone. Son aplicaciones a las que habrá que acceder desde el propio navegador web del teléfono, aunque una vez dentro se intente imitar el comportamiento y la apariencia de una aplicación nativa. Son convenientes porque su funcionalidad no depende del sistema operativo desde el que se esté utilizando, sino del navegador web que, exceptuando las características

más novedosas, siempre son las mismas. Para desarrollarlas funcionan igual que una página web: se tendrá que utilizar un servidor web y el método de programación es el universal (HTML, CSS, JavaScript, ...).

Al contrario que las aplicaciones nativas, estas son más lentas, menos intuitivas y tienen acceso limitado o complejo al hardware del teléfono. Además, son inaccesibles desde las tiendas de las plataformas y no son instalables. Para usarlas, habrá que acceder desde el navegador web del terminal. Esto implica ciertos riesgos de seguridad que los otros tipos no tienen, como el acceso a rutas no permitida. Tampoco tienen acceso a características como las *notificaciones push*[2] o acceso a la lista de contactos. Además, la comunicación con el servidor se hace de manera más constante y esto puede hacer que el servidor se sobrecargue o tenga unos tiempos de respuesta mayores.

- **Híbridas:** El último tipo de aplicaciones se puede considerar una mezcla entre los dos anteriores. Las aplicaciones híbridas son programas desarrollados utilizando tecnologías web (HTML, CSS, JavaScript, ...) que, mediante *frameworks* independientes al sistema operativo, se transforman en aplicaciones nativas de manera totalmente transparente para el desarrollador. El método de transformación puede variar pero el más común (Cordova[3]) es encapsular la ”página web” que se haya desarrollado en un componente[4] diseñado para interpretar páginas web, e introducir este componente como único elemento de una aplicación nativa. Así, y limitando algunas funcionalidades de la aplicación nativa generada, es posible generar aplicaciones nativas para cualquier plataforma sin trabajar directamente con las herramientas de ellas. La figura 3.1 describe de manera visual el resultado final de Cordova.

[2]Mensajes enviados de forma directa desde el servidor a un dispositivo en concreto. Sirven para avisar a los usuarios y mantenerles informados. Ejemplo: nuevo mensaje de WhatsApp.

[3]https://cordova.apache.org/

[4]En las aplicaciones nativas, estos componentes se suelen utilizar para poder navegar por la Web sin tener que salir de la aplicación

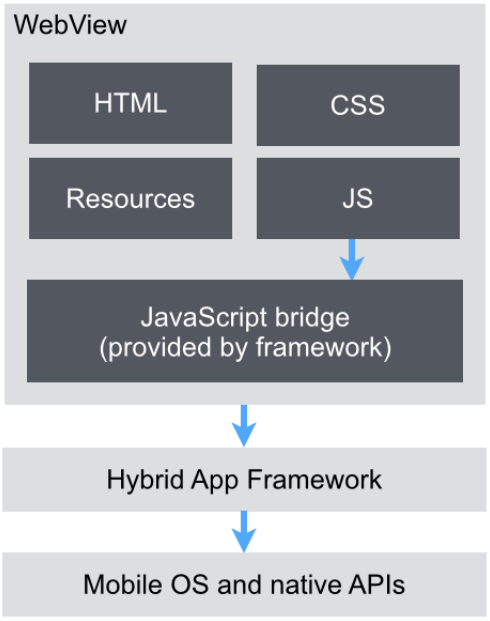

Figura 3.1: Método de encapsulado de Cordova[5].

El desarrollo de aplicaciones multiplataforma es tan sencillo como en el caso de las aplicaciones web, pero también se conservan las ventajas de las aplicaciones nativas: la aplicación es capaz de utilizar las herramientas hardware del dispositivo (gracias a *frameworks* como Cordova), se pueden publicar en las tiendas de las diferentes plataformas y los usuarios se la instalarán como cualquier otra aplicación, en la mayoría de casos el rendimiento es tan alto como el de las aplicaciones nativas, etc.

Pese a todo, también tienen alguna pega. La manera de comunicarse con el exterior se hace igual que en una aplicación web: mediante un servidor que esté esperando nuestras peticiones HTTP. Por último, el acceso al hardware es total pero puede ser más complejo que en el caso de las nativas, ya que en muchos casos habrá que realizarlo de diferente manera dependiendo del sistema operativo.

En la siguiente tabla se muestran las características listadas de una forma más visible y ordenada. El color de cada campo indica su resultado, siendo verde un buen resultado, naranja uno intermedio y rojo uno malo.

[5]Fuente: https://blog.codecentric.de/en/2014/11/ionic-angularjs-framework-on-the-rise/

	Tipo de aplicación		
	Nativa	**Web**	**Híbrida**
Rendimiento	Alto	Medio/bajo*	Alto/medio*
Publicables en la tienda	Sí	No	Sí
Uso de hardware del teléfono	Sí	No	Sí
Tiempo de desarrollo	Alto	Medio	Medio
Número de apps a desarrollar	2	1	1
Lenguajes soportados	Java/Swift	HTML, CSS, JavaScript	HTML, CSS, JavaScript
Necesidad de un servidor web	No	Sí	Sí
Comunicación con el servidor	Ninguna**	Alta	Media
Notificaciones push	Sí	No	Sí

* Dependerá del tipo de aplicación, en nuestro caso Medio para Web, Alto para Híbrida
** Tendrá que haberla, pero podrá ser con cada servidor específico en vez de con uno centralizado que haga de intermediario

Figura 3.2: Comparativa entre los tipos de aplicación.

A continuación se explicará la importancia de cada una de estas características en nuestra aplicación. Esto servirá para justificar los colores de cada uno de los campos de la figura anterior.

- **Rendimiento:** El comportamiento general de la aplicación es importante. Es necesario que sea apetecible de usar y que no haya retardos entre las acciones del usuario y las respuestas de la aplicación. Sin embargo, este campo se refiere únicamente al rendimiento de la aplicación en sí, y no al tiempo de respuesta del servidor, que es donde se prevé que vaya a haber más problemas. Esto es porque las tareas que tiene que realizar la aplicación solo será navegar entre páginas y mostrar información obtenida del servidor, no tendrá que realizar tareas como renderización de imágenes o trabajar con archivos de gran tamaño. Por todo esto, no se espera una diferencia de rendimiento notable entre los diferentes tipos, sobre todo entre el nativo y el híbrido.

- **Publicables en la tienda:** Esta característica es importante únicamente para que los usuarios sean capaz de encontrarla.

- **Uso de hardware del teléfono:** Aunque en un principio no está pensado hacer uso de ninguna de las funcionalidades del teléfono, puede ser que más adelante en el proceso de implementación sí que haya que usar alguna (como guardar información en ficheros de configuración). Es por esto que tener la opción puede ser ventajoso.

- **Tiempo de desarrollo:** Un apartado muy importante, ya que puede cambiar las horas de dedicación estimadas e incluso la viabilidad del proyecto. Con los recursos limitados que se dispone, cuanto más bajo sea el tiempo de desarrollo requerido, mejor.

- **Número de apps a desarrollar:** Está ligado con el tiempo de desarrollo pero también tiene en cuenta aspectos como los conocimientos requeridos o el mantenimiento después del despliegue. Tiene un gran peso en la decisión final.

- **Lenguajes soportados:** Aunque no se hayan valorado los diferentes lenguajes que toman parte (comparar las ventajas y desventajas de cada uno supondría un coste muy elevado para, probablemente, no obtener un resultado claro), sí que se tiene en cuenta la experiencia que se posee y la comodidad del autor trabajando con ellos. Por esto, las aplicaciones nativas se quedan un poco atrás por no conocer el lenguaje Swift o Objective C.

- **Necesidad de un servidor web:** Este apartado es importante, ya que el servidor web deberá ser público y capaz de conectarse a los diferentes servidores del DIPC. Este tipo de servidores suelen ser objetivo de ataques y requieren un mantenimiento en seguridad.

- **Comunicación con el servidor:** Al igual que el rendimiento, es importante para calcular el comportamiento general del sistema. No se espera una gran cantidad de usuarios de forma concurrente (existen alrededor de 200 cuentas activas), por lo que es probable que no tenga mucho impacto en el rendimiento final. De todas maneras, se intentará que sea lo menor posible.

- **Notificaciones push:** Es la única característica que se planea utilizar de entre todas a las que las aplicaciones web no tienen acceso. No tendrá mucho peso en la decisión final.

Decisión final

Por la gran cantidad de trabajo que supondría implementar las aplicaciones para Android como para iOS, se ha descartado la opción nativa. Entre las restantes opciones, las aplicaciones híbridas parecen ofrecer ventajas claras sobre las aplicaciones web, que se quedan atrás en casi todos los aspectos.

Por todo esto, se ha tomado la decisión de desarrollar la aplicación de forma híbrida. Se espera obtener resultados similares a los que se hubiese obtenido desarrollando una aplicación nativa pero en menor cantidad de tiempo. Para conseguirlo, haremos uso de diferentes herramientas que se comentarán a continuación.

3.3. Front-end

El *front-end* de un proyecto de software se refiere a la parte de la aplicación con la que el usuario interactúa. En nuestro caso, el *front-end* está formado por la aplicación que va a estar instalada en los dispositivos de los usuarios.

3.3.1. Ionic

Ionic[6] es un *framework* basado en Angular[7] pensado para la construcción de aplicaciones para dispositivos móviles tanto en Android como en iOS. Ionic se centra principalmente en el diseño y la estética, generando mediante HTML5, CSS y JavaScript aplicaciones que parecen nativas.

Aunque Ionic ofrezca herramientas para generar aplicaciones por sí misma, es recomendable usarlo de manera conjunta con Angular. Sin entrar en detalle, Angular provee un conjunto de componentes muy potentes, capaces de asemejar el comportamiento de nuestra aplicación aún más al de una nativa, incluso añadir algunas funcionalidades no presentes.

3.3.2. Angular

Angular es un *framework* estructural de desarrollo de aplicaciones web basado HTML. Es un proyecto *open source* y está mantenido tanto por Google como por una extensa comunidad de colaboradores. Su arquitectura se denomina *Component-based Architecture*, que es un enfoque que cogieron de React al crear la versión Angular2. La figura 3.3 ayuda a entender esta arquitectura haciendo una comparación con una arquitectura Modelo Vista Controlador, pero hay que entender que, aunque tengan similitudes, son diferentes. Se va

[6] https://ionicframework.com/
[7] https://angular.io/

a hablar más a fondo sobre ella en el capítulo 5, en la sección sobre 'Arquitectura de la aplicación'.

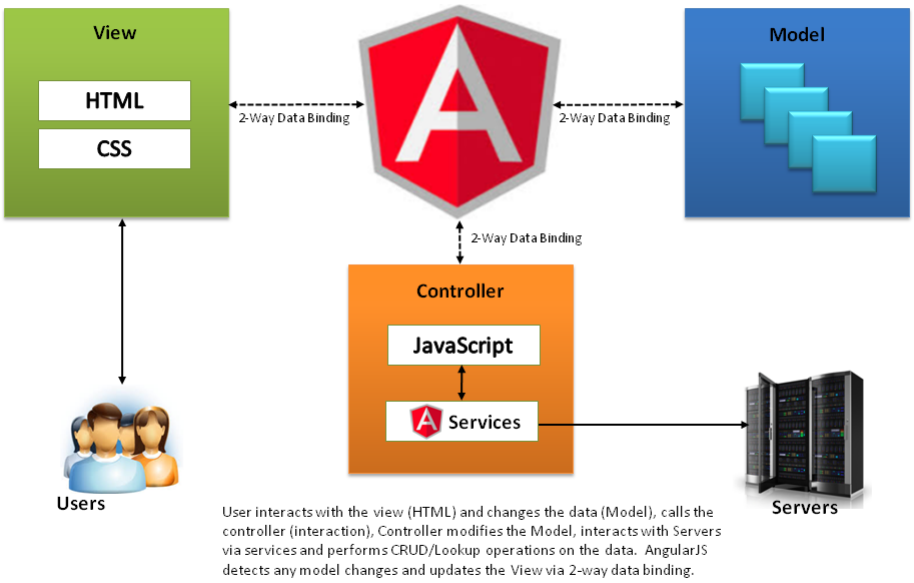

Figura 3.3: Arquitectura Angular[8].

Algunas de las características más importantes que proporciona Angular a nuestro proyecto son:

- **Two-Way Data Binding:** Como ya se ha comentado, es la sincronización automática de datos entre las 3 capas de la aplicación. De la manera en que está implementado, se puede entender como que el modelo es la fuente original de los datos, y el resto de componentes una representación actualizada de dichos datos. Así, cuando el modelo cambia, también cambian sus representaciones, y viceversa. Una de las muchas ventajas de esta característica es la facilidad que otorga a la hora de hacer tests, ya que se puede probar cada elemento individual por separado.
- **Servicios:** Los servicios se pueden ver como unidades de código que se pueden insertar en otras unidades. Como su nombre indica, esto sirve para dar un servicio común a diferentes páginas. Estos servicios solo se cargan cuando son necesitados

[8]Fuente: https://cellcode.us/quotes/diagram-how-internet-work-does.html

y cada página en la que se insertan guardan una referencia a la instancia del servicio de toda la aplicación (*singleton*).

- **Router:** El router de Angular es un módulo que permite navegar entre diferentes páginas de la aplicación y generar una jerarquía entre los componentes. De esta manera, se pueden cargar de forma dinámica (*lazy-loading*: se refiere a la carga por demanda de módulos si el router ve que la vista lo necesita y no ha sido cargado) los diferentes elementos de una vista, dependiendo de a dónde se quiera acceder. Para el desarrollador, la implementación es muy sencilla y el potencial muy grande.

- **Otras:** También se incluyen en Angular la capacidad de crear componentes y directivas (atributos HTML para proporcionar nuevas funcionalidades a los elementos), formularios *reactive*, simplicidad en la validación de formularios, un cliente HTTP integrado, animaciones, y muchas otras características más.

El uso de Angular en nuestra aplicación es constante y las ventajas que provee a la hora de programar el *front-end* del proyecto son muchas. Cada vez que cambiamos de página, utilizamos un elemento, o rellenamos un campo de un formulario, se estará usando Angular.

3.3.3. Cordova

En el entorno de nuestra aplicación, Cordova lo utilizaremos para encapsular los componentes generados en componentes WebView, capaces de conectarse a las APIs nativas del dispositivo. Cordova dispone de varios *plugins* para dar acceso a estas funcionalidades. Algunos de los más importantes son: *Network, Keyboard, Storage, Device,* etc, que dan acceso a funcionalidades de la red, uso del teclado del teléfono o de aplicaciones externas, almacenamiento interno y externo del dispositivo, y a características propias de dispositivo, respectivamente.

En la figura 3.4 se puede entender mejor la arquitectura de Cordova y el papel que juega en nuestra aplicación. La parte WebApp se refiere al código de nuestra aplicación que será generado con Ionic y Angular.

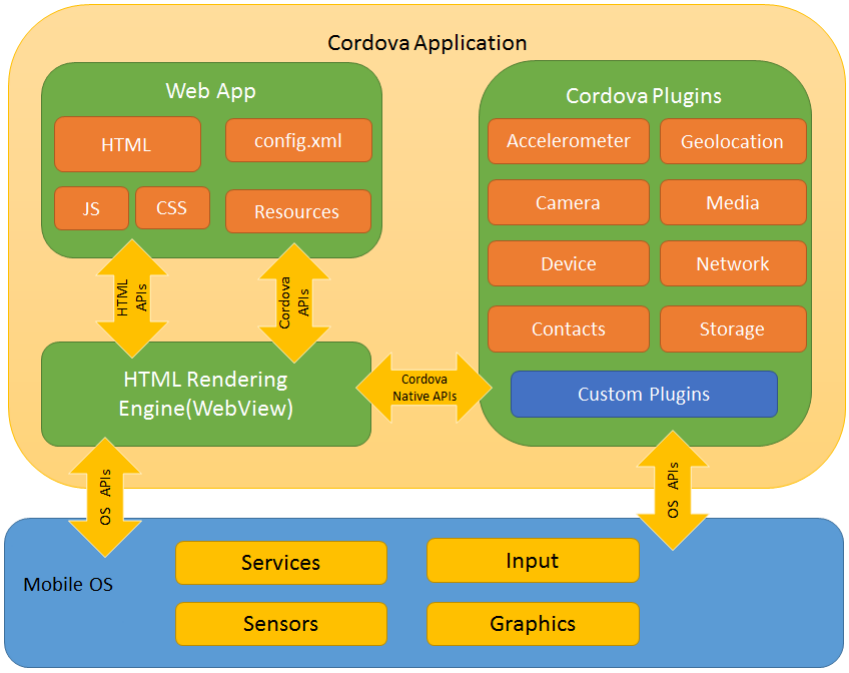

Figura 3.4: Arquitectura de una aplicación Cordova[9].

Y, ¿cómo usamos todas estas herramientas en conjunto? Ionic nos ofrece variedad de herramientas para trabajar con su software. Desde *low-code IDE*s[10] (**Ionic Creator**), pasando por herramientas de compilación y empaquetado en la nube (**Ionic Package**), hasta aplicaciones para smartphones que permiten probar y compartir el producto sin necesitar de instalarlo ni desplegarlo (**Ionic View**). De entre estas herramientas y más que están disponibles, las que se han decidido utilizar son:

- **Ionic CLI:** Probablemente la herramienta más potente de todo el *framework*, Ionic CLI es la forma por defecto de trabajar con Ionic. CLI significa *command line interface* y, como su nombre indica, se refiere a un programa que aporta varias funcionalidades desde el terminal. Para el desarrollador, esta es la forma de trabajar con Cordova: instalación de *plugins* de Cordova, compilación de la aplicación, despliegue multiplataforma, y emulación son algunas de las características más importantes.

  Pero no solo se queda ahí. Con un sencillo comando como *ionic start*, CLI nos generará desde cero una aplicación utilizando Angular, además de darnos la opción

[9] Fuente: https://cordova.apache.org/docs/en/latest/guide/overview/index.html
[10] Entornos de desarrollo integrados sin tener que programar, normalmente con tecnologías *drag&drop* para incluir elementos y funcionalidades al producto.

de usar una de las plantillas que ofrece. Otros comandos muy potentes son *ionic generate*, que genera diversos elementos de la aplicación (como páginas, servicios o proveedores) y los añade automáticamente a la estructura del programa, o *ionic serve*, que permite probar la versión actual de nuestra aplicación en el navegador web del ordenador desde el que se está desarrollando. *ionic serve* lo despliga al puerto 8100 de la IP privada y pública (si está asignada) de nuestra red, por lo que es posible conectarse a la aplicación desplegada desde un teléfono en la misma red. Una opción muy útil es *livereload*. *Livereload* permite visualizar los cambios que se hagan en la aplicación en vivo, sin tener que parar la ejecución. Los cambios se producen cada vez que se guarde uno de los ficheros del código de la aplicación y solo se compila el módulo al que pertenece, por lo que se evitan pasos innecesarios y agiliza el proceso de implementación y pruebas.

- **DevApp:** Una aplicación para smartphones, permite probar la aplicación servida desde CLI en un teléfono en vez de un navegador. Así, se consigue una experiencia más parecida a lo que sería la aplicación instalada en el dispositivo. Aún así, varias funcionalidades de Cordova no están disponibles usándola, por lo que siempre se recomienda hacer pruebas compilando e instalando la aplicación en un dispositivo físico o emulador. Esta herramienta se usará principalmente en los primeros meses de implementación para *debuggear* la aplicación en iOS, hasta que se disponga de un iMac[11] desde donde lanzarla.

- **Ionic Labs:** Lo que antes era una aplicación independiente, ahora es parte del comando *ionic serve*. *Labs* permite que, al lanzar la aplicación con *ionic serve*, se pueda hacer en dos plataformas a la vez. Son dos aplicaciones independientes por lo que se podrán utilizar de forma concurrente, además de permitir buscar fallos o diferencias entre plataformas sin tener que parar la ejecución y volver a lanzarla. La figura 3.5 muestra perfectamente lo que es.

[11] para instalar la aplicación en un dispositivo iOS es necesario hacerlo desde el IDE XCode, solo disponible para iMac

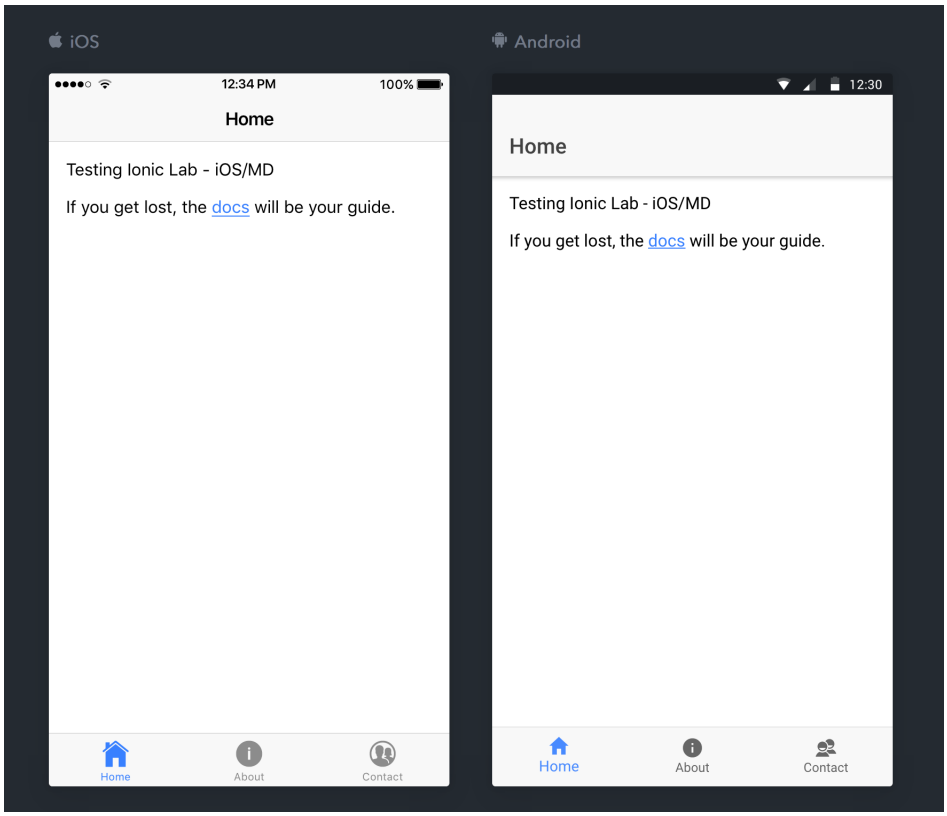

Figura 3.5: Ejemplo de Ionic Lab[12]

En cuanto a las versiones de estas herramientas, como todas ellas están aún siendo mantenidas, han ido variando durante el desarrollo del proyecto. Por esto, se mencionarán las versiones utilizadas pero se marcarán con una "x" las *minor*, que suelen ser para corrección de fallos.

Herramienta	**Versión**
Ionic (ionic CLI)	4.x.x
Ionic Framework	4.0.x
Angular	7.2.x
Cordova	8.1.x
AndroidSDK	26.1.x
iOS	12.1.x
XCode	9.4.1

Tabla 3.1: Versiones herramientas *front-end*

[12]Fuente: https://github.com/ionic-team/ionic-cli/issues/3687

3.4. Back-end

Hasta ahora se ha explicado todo lo que tiene que ver con la organización del proyecto y la parte cliente del sistema. Como ya se ha dicho con anterioridad, esta aplicación tendrá que comunicarse de alguna manera con ciertos servidores de la empresa. Para esto, habrá que poner en marcha al menos un servidor web que trate con las peticiones HTTP, pero también habrá que conectar una base de datos que se encargue de almacenar cierta información de los usuarios y la información necesaria para generar estadísticas.

3.4.1. Servidor web

Desplegar un servidor web es un mundo. Hay que elegir si se hace en una máquina física o virtual, el sistema operativo que vaya a tener instalada la máquina, el software que vaya a gestionar el servidor web, el lenguaje en el que se vaya a implementar, las diferentes medidas de seguridad que se vayan a adoptar, etc. Además, habrá que adaptar las necesidades de la aplicación a los recursos de los que nos provea el DIPC.

Para facilitar bastante el proceso de toma de decisiones, se ha trabajado de forma conjunta con la empresa y así intentar aprovecharnos de ciertas decisiones que ya se hayan tomado para cuestiones similares con anterioridad. La página web oficial de la empresa está albergada en un servidor web del propio DIPC, pero además hay contratado un servicio externo donde están publicadas páginas web antiguas que no merece la pena mantener. Esto es así porque, como este servidor no está correctamente protegido, en caso de un ataque externo solo se pararía el servicio de las páginas poco relevantes. Además de esto, hay en proceso una actualización de la página web del DIPC. En cuanto la actualización se realice, el servidor que albergue la página oficial será otra máquina (aunque seguirá siendo una máquina del DIPC), y estará mantenido por la empresa que se encarga de realizar el nuevo sitio web. A esto hay que añadir el requisito de que, para ser capaces de trabajar con los servidores necesarios (nodos de cálculo, servidores de administración, ...), es imprescindible que el servidor web donde se vaya a implementar el *back-end* de la aplicación tenga acceso a la red interna de la UPV/EHU[13].

Teniendo en cuenta estos puntos, se ha decidido que para la fase de producción, se pondrá

[13]El DIPC pertenece a esta red, lo que implica que para acceder a sus servidores solo se podrá hacer desde dentro de la propia red o con tecnologías como VPN. En nuestro caso, un servidor web con IP pública ubicado dentro de la red tendría acceso a los servidores necesarios y sería accesible desde cualquier parte.

a nuestra disposición una máquina dentro de la red del DIPC. Esta máquina tendrá CentOS[14] 7, que es la versión que se usa en el servidor web actual, además de ser un sistema operativo que se caracteriza por su estabilidad y por no requerir mucho mantenimiento. En esta máquina habrá que instalar un software que gestione sitios web e implementar la página web que se encargará de tratar las peticiones de la aplicación. Una vez el sitio web oficial sea migrado, se migrará también la página de la aplicación a la misma máquina. La máquina de producción ha sido pensada para ser idéntica a la máquina de despliegue, por lo que no se prevé que haya ningún problema serio. La única diferencia es que en el caso del despliegue, habrá dos sitios web alojados en el mismo servidor.

Para gestionar estos sitios web, el software que se va a utilizar es Nginx[15]. Al tener que utilizar la misma máquina que la página web oficial, el software que se vaya a usar también tendrá que ser el mismo. Por esta razón, este proyecto se ha adaptado a las especificaciones técnicas del proyecto encargado de la construcción de la página web, que se comenzó antes y tiene mayor envergadura que este. De todas maneras, Nginx hubiese sido uno de los candidatos si se hubiese que tenido que tomar la decisión en este proyecto, siendo su principal rival Apache[16]. La forma de configurarlos es parecida y Nginx hace mejor uso de recursos, razón por la que está aumentando su participación en el mercado[17].

Una vez esté el servidor configurado para tratar peticiones HTTP, habrá que implementar la página que se encargue de ello. Para el tipo de funcionalidades que se necesitan, lo más apropiado será crear una API. Aunque se podría considerar una API a cualquier servicio que trate con peticiones y responda a las peticiones recibidas, en este caso en concreto nos referimos a una API sin interfaz de usuario ni parte visual, que se encargue de enviar datos en vez de servir páginas.

En cuanto al tipo de API, los dos más comunes serían REST[18] y SOAP[19]. Se podría hacer una comparativa seguida de una toma de decisión, pero en este caso no se considera necesario. Para el tipo de servicio que necesitamos, la opción correcta es sin duda REST. Mientras que SOAP es más comúnmente utilizado para aplicaciones financieras o de pago, donde es necesario realizar múltiples transacciones de manera continuada y donde la autorización es parte del protocolo, REST es más popular con aplicaciones móviles y de

[14] https://www.centos.org/
[15] https://www.nginx.com/
[16] https://httpd.apache.org/
[17] Fuente: https://news.netcraft.com/archives/2019/02/28/february-2019-web-server-survey.html
[18] Representational State Transfer
[19] Simple Object Access Protocol

redes sociales, donde se distingue claramente la parte servidora del cliente, las peticiones a la API son independientes unas de las otras y la gestión de la autorización se puede gestionar desde el cliente. Otras ventajas de REST son su bajo consumo de datos (muy importante para aplicaciones móviles), la sencillez para implementar y mantenerlo, y los diferentes formatos que soporta para realizar la comunicación, como XML o JSON.

En concreto, el formato que se ha elegido para el intercambio de información es JSON. Las razones para haber tomado estado decisión frente a otros formatos (como XML) son varias: reducido tamaño, diseñado específicamente para ser usado como formato de comunicación, fácil conversión array-JSON y objeto-JSON, soporte en múltiples lenguajes sin inclusión de librerías externas, etc.

Lenguaje del servidor

El próximo paso sería elegir el lenguaje en el que se va a programar la API. Básicamente, cualquier lenguaje que permita tratar con peticiones HTTP, trabajar con JSON y realizar conexiones mediante SSH sería candidato. Para reducir la lista de opciones, se van a seleccionar solo los *frameworks* recomendados por la comunidad de Ionic para implementar un *back-end*: ExpressJS[20], CodeIgniter[21] y Django[22]. Como las tres opciones son viables y no va a haber diferencias notables en cuanto a rendimiento, se van a listar las características de cada opción y se elegirá una en base al trabajo que requiera utilizarlo y la comodidad que se tenga utilizándolo.

ExpressJS es un *framework* construido sobre NodeJS[23] para implementar aplicaciones web. Con la cantidad de utilidades HTTP y de *middleware* que ponen a disposición del desarrollador, se jacta de ser capaces de construir APIs robustas de forma rápida y sencilla. Un punto fuerte de esta opción es que, como su nombre indica, se desarrolla usando JavaScript, que es un lenguaje parecido a TypeScript[24] (ambos lenguajes están basados en ECMAScript). Desarrollar las dos partes del proyecto en lenguajes similares ayudará a agilizar el tiempo de aprendizaje, además de ser más cómodo para el desarrollador. Sin embargo, elegir ExpressJS implicaría tener que instalar NodeJS en la máquina y configurar el sitio web de Nginx para comportarse como un *proxy* entre la aplicación y la API. La

[20] https://expressjs.com/
[21] https://www.codeigniter.com/
[22] https://www.djangoproject.com/
[23] https://nodejs.org/
[24] Lenguaje utilizado en Angular

complejidad de este proceso es alta, además de elevar los costes de mantenimiento (dos servidores en vez de uno) y dificultar la migración entre máquinas.

Al igual que ExpressJS, CodeIgniter es un *framework* de PHP orientado al diseño de aplicaciones web. Es muy ligero y tiene variedad de funcionalidades integradas para que el desarrollador se pueda centrar en la aplicación y dejar de lado cuestiones como la seguridad, el rendimiento y la configuración. Al estar basado en un lenguaje tan potente y con tantos años como PHP, la cantidad de documentación es mayor que la del resto. Por último, la instalación no puede ser más sencilla: copiar los archivos del *framework* a la raíz del sitio web y está hecho. Esto también facilita mucho el proceso de migración, evitando posibles errores de dependencias y de diferencias en archivos de configuración.

Por último, Django es el equivalente de ExpressJS y CodeIgniter, pero en Python. Se centran en ser capaces de construir las aplicaciones en tiempo récord. Su arquitectura *Model-View-Controller* es muy característica y el acceso a bases de datos relacionales se hace mediante ORM[25], olvidándonos así de sentencias SQL. El problema es el mismo que en ExpressJS: la instalación sobre un servidor Nginx es más difícil, por lo que la migración también lo será. Además, la implementación de una API requeriría de un *framework* extra, como por ejemplo Django REST[26].

Teniendo en cuenta los aspectos que se han explicado de cada alternativa y poniendo especial importancia en la sencillez de la instalación (con el fin de facilitar lo máximo posible el proceso de despliegue), se ha decidido utilizar CodeIgniter. Como se ha dicho al principio, cualquier opción era potencialmente válida, lo que se ha intentado es minimizar el coste que va a suponer el desarrollo del servidor.

CodeIgniter está estructurado de la siguiente manera: cuando se recibe una petición HTTP, llega al fichero *index.php* ubicado en la raíz del servidor, que se encargará de inicializar todos los recursos necesarios para ejecutar CodeIgniter, y redirige la petición al *router*. El *router* tendrá que decidir qué se va hacer con dicha petición, si está en caché se servirá la vista asociada, y si no, se mandará al módulo de seguridad, siguiendo el flujo normal de la aplicación. El módulo de seguridad se encargará de analizar la petición y la información recibida, y de filtrarla si fuese necesario. El resultado del filtrado se mandará al controlador, que se encargará de cargar el modelo, las librerías necesarias y otro tipo de recursos, y genera la vista correspondiente. Está vista se mandará al cliente y se almacenará en cache para futuras peticiones. La figura 3.6 muestra de forma gráfica los pasos que se acaban de explicar.

[25] *Object-Relational Mapping*: https://es.wikipedia.org/wiki/Mapeo_objeto-relacional
[26] https://www.django-rest-framework.org/

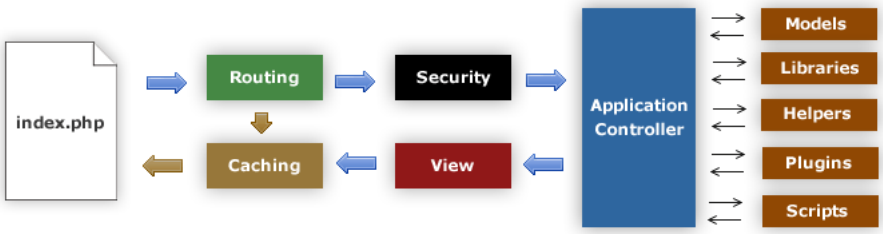

Figura 3.6: Flujo de ejecución de CodeIgniter[27].

La estructura Vista-Modelo-Controlador que se utiliza es bastante libre. La parte del Modelo es totalmente opcional, y si se cree que el trabajo de mantener los modelos va a ser mayor que el beneficio que aportan, se pueden dejar de lado y trabajar solo con vistas y controladores. En nuestro caso, como la comunicación se hace en formato de objetos JSON, las vistas no serán más que estos objetos, por lo que se pueden generar desde el mismo controlador. Por lo tanto, nuestra API en concreto podrá centrarse en el desarrollo de controladores.

Algo que sí que se va a necesitar son los *helpers* y librerías de CodeIgniter. Los helpers son agrupaciones de funciones que se pueden invocar desde cualquier vista o controlador. Su propósito principal es fomentar el código reutilizable, aunque también sean un método sencillo de importar funciones de otros autores. Las librerías son clases, de estructura más compleja que los *helpers*, que son exportadas y se pueden cargar cuando sean necesarias. Normalmente, cuando nos referimos a librerías, estamos hablando de las librerías propias de PHP o CodeIgniter, como la librería que implementa las sesiones o la librería de criptografía.

Otra característica importante de CodeIgniter es la seguridad que ofrece a los sitios web que aloja. CodeIgniter filtra las URI que son permitidas en nuestro sitio web para prevenir ataques que intenten introducir información maliciosa en nuestro sistema. También tiene un filtro XSS[28] que evita la ejecución de código JavaScript en nuestro servidor de manera remota. Por último, también está implementada una funcionalidad que protege de CSRF[29] (aunque no esté habilitada por defecto), ataque que engaña al cliente haciéndoles enviar peticiones sin que él lo sepa.

[27] Fuente: https://www.codeigniter.com/user_guide/overview/appflow.html
[28] *Cross-Site Scripting Filter*
[29] *Cross-Site Request Forgery*

JWT

Para el tratamiento de sesiones en la aplicación y evitar acceso a ciertas funciones de la API a usuarios que no hayan iniciado sesión o no tengan permisos suficientes, se utilizará un estándar llamado JWT. Esta tecnología sustituye a las clásicas *session* de PHP.

El funcionamiento es completamente diferente pero logran objetivos similares. JWT consiste en enviar una *cookie* en formato JSON que esté firmada con la clave privada del servidor. El cliente almacenará esta *cookie* y la enviará en futuras peticiones a la API para identificarse. A todas las peticiones que se hagan a ciertas funciones de la API sin un JWT o con uno no válido, se les denegará el acceso.

El JWT está compuesto por tres campos: *header, payload* y *signature*. Cada campo está separado del resto por un punto '.' y a simple vista, parece que no contienen ninguna información coherente. En la figura 3.7 se puede ver el formato, cada campo separa por un punto y de un color diferente.

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJz
dWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gR
G9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.rtPTS_xDiBg
Eb5IuelrKPRf8wMqdhJ1z5GFZgUS-iZg
```

Figura 3.7: Ejemplo de un posible JWT.

1. ***Header:*** Es la cabecera del JWT y contiene meta-información sobre el objeto. Normalmente consta de dos campos: el tipo de objeto que es (JWT) y el algoritmo de cifrado que se usa en el campo *signature*. Está codificado en base 64.

2. ***Payload:*** El cuerpo del JWT, contiene toda la información que se quiera intercambiar entre emisor y receptor. Sigue una estructura JSON convencional y se recomienda que contenga ciertas declaraciones (*claims*). Estas declaraciones son de tres tipos: *registered, public* y *private claims*. *Registered claims* son un conjunto de declaraciones establecidas en el estándar[30] que están pensadas como punto de inicio y para definir ciertas declaraciones de uso común. *Public claims* son declaraciones definidas por los usuarios pero que, para prevenir colisiones, deberán estar registradas. Por último, *private claims* son las declaraciones que, sin ser *registered* o *public*, se utilizan entre emisor y receptor bajo previo acuerdo. Todo el campo *payload* estará también codificado en base 64.

[30] Estándar RFC 7519: https://tools.ietf.org/html/rfc7519

3. ***Signature***: Último campo del JWT, está creada a partir del campo *header* seguido de un punto, seguido del campo *payload*. Todo esto se cifra con el algoritmo especificado en el campo *header* y una clave (será pública o privada dependiendo del tipo de algoritmo que se utilice). Así, este campo verifica que el JWT no ha sido modificado por el camino y, si se ha usado una clave privada como secreto, también servirá para validar que el emisor es realmente quién dice que es.

Si decodificásemos el JWT de la figura anterior obtendríamos un JWT legible y con los campos de los que se acaba de hablar. El resultado se puede ver en la figura 3.8:

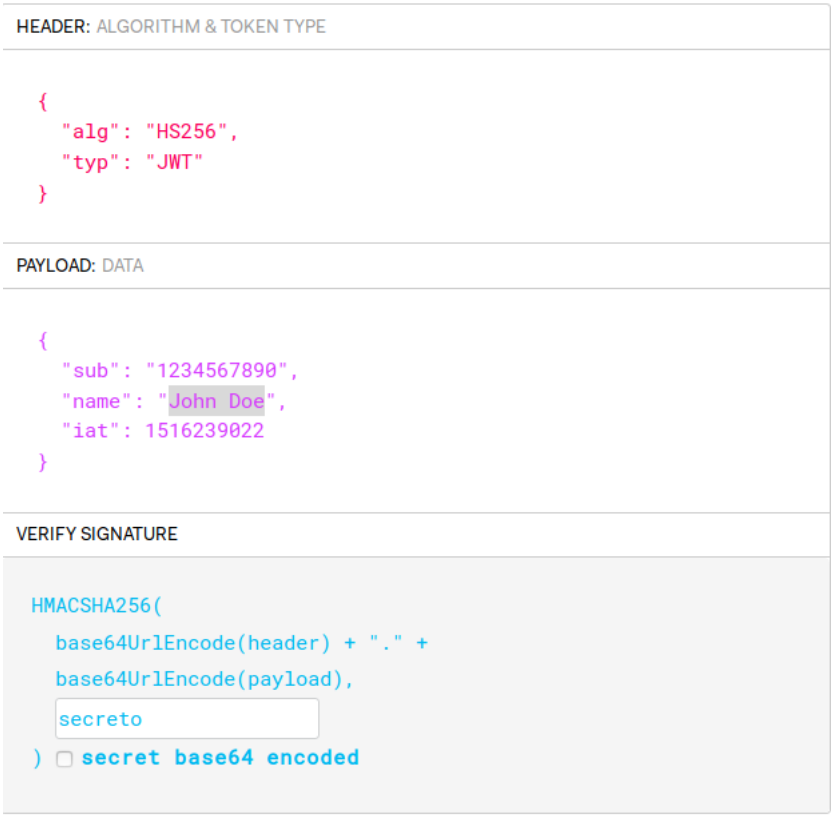

Figura 3.8: JWT decodificado.

Sabiendo cómo funciona esta tecnología, podremos utilizarla para bloquear todas las peticiones a la API que no lleven un token válido (exceptuando las de algunas funciones en concreto como la de iniciar sesión). Con un token válido nos referimos a un token que haya sido generado por el servidor. Además, esta implementación nos permitirá en un futuro añadir más campos al token, como un tiempo de expiración. Es importante aclarar que no

se puede añadir información privada en el token, ya que el contenido no está cifrado, solo codificado.

3.4.2. Base de datos

Un elemento imprescindible para que nuestro sistema funcione correctamente será una base de datos. Aquí habrá que almacenar cierta información relacionada con la aplicación que no esté en ninguna otra ubicación del sistema, por ejemplo los avisos que envía un usuario a los administradores.

Hasta el momento, con el conjunto de tecnologías que se ha listado se puede ver la similitud con la arquitectura de servidores web LEMP. LEMP es una variante de LAMP[31] que sustituye Apache por Nginx, al estar creciendo en popularidad y tener un impacto en los recursos del sistema mucho menor[32].

De los dos sistemas de base de datos que propone LEMP, el que se ha decidido utilizar es MariaDB. Estructuralmente son los dos iguales, ya que MariaDB es una derivación de MySQL después de que esta última tecnología fuese comprada por Sun Microsystems en 2008. Sin embargo, MariaDB está en continuo desarrollo por el equipo de MySQL original, además de tener una comunidad de usuarios más amplia.

Como ya se ha dicho, las similitudes entre MariaDB y MySQL son muchas. La sintaxis de las sentencias es la misma, la arquitectura de tablas es similar y la mayoría de *plugins* externos funcionan para ambos SGBDs. De todas formas, MariaDB proporciona mejoras en cuanto a rendimiento, sobre todo con los nuevos motores de base de datos que ofrece, siendo algunos de ellos versiones mejoradas de los viejos motores (como lo es XtraBD de InnoDB).

Para nuestro *back-end*, ha habido que tomar una decisión importante: ¿queremos tener la base de datos en la misma máquina que el servidor web o queremos que estén en dos diferentes? Ambas opciones tienen sus ventajas y sus desventajas:

[31] *Linux Apache MySQL/MariaDB PHP*
[32] La 'A' se sustituye por 'E' porque Nginx se pronuncia '*engine-x*'

	Ventaja	Desventaja
Misma máquina	Menos uso de red	Migración más compleja y más costosa
Base de datos remota	Proporciona un extra de seguridad	Acceso a los datos más lentos

Tabla 3.2: Ubicación base de datos

Como se puede ver en la tabla 3.2, alojar la base de datos en la misma máquina que el servidor web usará menos ancho de banda y tendrá acceso más rápido. Aunque el acceso vaya a ser constante, la cantidad de datos con la que se va a trabajar no va a ser demasiado grande y la latencia entre las dos máquinas no será muy alta (ambas máquinas estarían en la misma subred). Es por esto que se ha considerado que la mejor opción es separar los dos servicios. Así, a la hora de migrar el servidor web lo único que habrá que cambiar para que el servicio siga funcionando será permitir el acceso de forma remota a la dirección del nuevo servidor web (en vez de tener que reinstalar el SGBD, además de evitar problemas de compatibilidades con la base de datos del sitio web oficial). Además, separar los servicios implica que si un atacante consigue acceso al servidor web, seguirá sin tener acceso a la base de datos. Es un caso extremo y poco probable, pero cualquier medida de seguridad extra es bienvenida.

En conclusión, se instalará la base de datos en un servidor remoto que esté dentro de la red del DIPC. Este servidor se configurará para solo ser accesible mediante la IP del servidor web y nos conectaremos a través del controlador de bases de datos de CodeIgniter.

3.4.3. Nagios

Nagios es un software diseñado para monitorizar sistemas, redes e infraestructuras. La estructura de Nagios se puede dividir en tres partes: el propio **servidor**, que se encarga de monitorizar y de almacenar la información; los **objetos**, que son los diferentes elementos monitorizados por el servidor y se contacta con ellos mediante protocolos como TCP/UDP o SNMP; y el **cliente**, que se refiere a todas las formas de acceso a la información del servidor que existan, normalmente un navegador web.

El DIPC tiene varios servidores Nagios instalados, que utilizan para monitorizar diferentes grupos de objetos. El que interesa en concreto para este proyecto es el encargado de monitorizar la micro-informática de la empresa. Los equipos que se monitorizan son prin-

cipalmente los ordenadores de los empleados e investigadores del centro y las impresoras que hay en los edificios. De esta manera, es posible saber qué impresoras están activas, a cuáles les falta tinta o papel y qué ordenadores no tienen el antivirus instalado sin tener que revisarlos personalmente uno por uno, entre otras muchas cosas.

Uno de los objetivos de esta aplicación es ser capaz de mostrar la información obtenida del servidor Nagios en los *smartphone* de los administradores, para que puedan ser capaces de monitorizar los equipos sin tener que revisar el servidor periódicamente. Para poder realizar esta tarea, se hará uso de los ficheros de *host*s y de servicios que utiliza Nagios. Estos ficheros se encuentran en el directorio */usr/local/nagios/etc/objects* por defecto y se llaman *hosts.cfg* y *services.cfg*. En el fichero *hosts.cfg* se define toda la lista de máquinas clientes que se quieran monitorizar y en el fichero *services.cfg* se define de qué forma se va a monitorizar cada cliente (se incluye hasta una línea con el comando específico que habrá que ejecutar). De esta manera, se podrían leer estos dos ficheros y ejecutar los comandos de cada servicio para obtener el estado de las máquinas clientes de forma manual. También merece la pena mencionar que Nagios dispone de *plugin*s o *script*s que serán los que se ejecuten con cada servicio y que se encuentran por defecto en el directorio */usr/local/nagios/libexec*.

3.4.4. SNMP

SNMP[33] es un protocolo de red que sirve para obtener información y modificar parámetros de dispositivos en cierta IP. El protocolo funciona con tres elementos: **los dispositivos gestionados**, que puede ser prácticamente cualquier tipo de dispositivo que tenga una tarjeta de red compatible con el protocolo pero suelen ser aparatos como *router*s, estaciones de trabajo, etc; **agentes**, que es el software que se está ejecutando en el dispositivo gestionado y se encarga de manejarlo (el agente suele venir instalado por el fabricante, como en nuestro caso); y **el sistema de administración de red**, que se encarga de comunicarse con los agentes que estén asociados.

Existen tres versiones del protocolo: v1, v2c y v3. Cada una es una mejora de la anterior pero, mientras que la versión 2c es retro-compatible con la versión 1, la versión 3 no lo es con ninguna de las otras dos, ya que implementa cambios drásticos en la forma de obtener información de los agentes (son necesarios unos credenciales, mientras que en las otras versiones solo se utiliza la IP de la máquina gestionada y un nombre de una comunidad[34]).

[33] *Simple Network Management Protocol*
[34] Palabras clave utilizadas para acceder a los agentes.

En nuestro caso, la tarjeta de red solo está configurada para la versión 1 y 2c, por lo que se usará la 2c.

Dependiendo del agente y de la máquina que se esté gestionando, el número de parámetros que se pueden controlar por SNMP varía. Estos parámetros tienen un identificador llamado OID[35] que los identifica inequívocamente dentro del árbol de objetos. Muchos de estos OIDs son comunes porque forman parte del estándar, pero dependiendo del fabricante y el tipo de dispositivo, el resto de campos suelen ser propios de cada máquina. En el capítulo 6 se verá exactamente qué objetos son los que nos interesan y cómo los obtenemos. De momento, un ejemplo para obtener el valor de un objeto en concreto se puede ver en la figura 3.9.

```
snmpget -v<<version>> -Ov -c <<comunidad>> <<IP-maquina>> <<OID>>
```

- **-v**: Corresponde al protocolo que se va a utilizar. En nuestro caso 2c.
- **-Ov**: Indica que el único *output* que queremos es el valor del objeto.
- **-c**: Nombre de la comunidad a la que se va a acceder. En nuestras máquinas se utiliza la comunidad *public* para acceder a objetos con permiso de lectura.

```
> snmpget -v2c -Ov -c public [redacted] .1.3.6.1.4.1.318.1.1.27.1.4.1.2.1.3.1.9
INTEGER: 271
```

Figura 3.9: Ejemplo para extraer el valor de un objeto por SNMP.

El valor que conseguimos en la figura 3.9 se puede dividir en dos partes: el tipo de objeto y el valor del objeto. Este valor se corresponde con la temperatura en Celsius del aire de entrada de una de las máquinas de frío del CPD[36] del DIPC, que tendremos que usarlo más adelante. El último dígito del valor son las décimas de grado (el valor sería 27.1 °C).

3.4.5. Torque/Maui/Slurm

Para administrar las colas de trabajos que los investigadores ejecutan en los supercomputadores, el DIPC utiliza actualmente dos *software*, llamados Torque[37] y Maui[38], que son

[35] *Object IDentifier*
[36] Centro de Procesamiento de Datos
[37] http://www.adaptivecomputing.com/products/torque/
[38] http://www.adaptivecomputing.com/products/maui/

los dos propiedad de la misma empresa. Torque es una extensión del original PBS y su función es administrar la cola de trabajos. Está diseñado para ser utilizado junto a Maui, que es un programador de trabajos y prioridades de la cola.

Al no estar Torque y Maui actualmente mantenidos por quién los diseñó, está en proceso un cambio de sistema de gestión de colas, modificando el gestor actual por Slurm[39]. Slurm es un gestor de colas diseñado para sistemas Linux que es más popular en grandes supercomputadores y está siendo activamente mantenido.

Para este proyecto no es necesario tener una comprensión profunda de estas herramientas. Habrá que acceder a la cola de trabajos y extraer información en algunos casos de uso, por lo que es necesario mencionar cómo funciona el sistema y cómo se extraerá esa información.

[39] https://slurm.schedmd.com/overview.html

4. CAPÍTULO

Análisis

Dentro del ciclo de vida de la Ingeniería del Software, el Análisis es la fase donde se analizan los requisitos técnicos del proyecto, se identifican los actores que van a tener un papel en el sistema y se describen los casos de uso que van a formar parte del producto.

En lo que a este proyecto se refiere, la información necesaria para poder analizar el proyecto se ha extraído del acuerdo que se realizó con el cliente[1]. Además ha habido que concretar con el cliente algunas características que la aplicación deberá tener, por lo que ciertos requisitos se han visto ligeramente modificados o ha habido que añadir alguno nuevo, siempre con el objetivo de hacer más completo el producto final y estableciéndose a lo acordado previamente.

[1] Véase: Anexo A

4.1. Requisitos

4.1.1. Requisitos técnicos

Smartphone

Para poder usar la aplicación, será necesario disponer de un *smartphone* con sistema operativo Android o iOS. Para el caso de Android, será necesario una versión 7.0 o superior. Esto es debido a que el *WebView* que Cordova utiliza solo puede ser utilizado a partir de la versión 7.0[2]. Para iOS, la versión mínima necesaria será iOS 11[3], por la misma razón que en Android.

La aplicación también podrá ser usada mediante emuladores siempre que la versión sea válida. La API que se va a generar no tendrá esta limitación y podrá ser usada por otros servicios si en un futuro se considera necesario.

Conexión a Internet

La mayor parte de funcionalidades de la aplicación se realizarán intercambiando información con la API. Para que este intercambio sea efectúe correctamente será necesario tener una conexión a Internet, lo más estable y rápida posible (algunas funcionalidades harán un uso constante de datos recibidos mediante el servidor web o descargarán ficheros de gran tamaño).

Antes del despliegue del producto, será necesario que la conexión se haga desde la red del DIPC o mediante VPN, ya que el servidor web aún no estará disponible desde ningún otro sitio. Para la fecha de finalización debería estar listo el servidor web final, para poder migrar la API y hacer uso de la aplicación con cualquier conexión a Internet.

Cuenta de cálculo

Cualquier persona que quiera acceder a los servicios que ofrece el Centro de Cálculo es necesario disponer de un usuario en los clústeres del DIPC y que esté activo. La aplicación hará uso de los credenciales de la cuenta del usuario para acceder a elementos como sus

[2]Distribución de versiones de Android se puede consultar aquí: https://developer.android.com/about/dashboards

[3]Distribución de versiones de iOS: https://developer.apple.com/support/app-store/

trabajos encolados o sus ficheros, entre otros, por lo que deberán proporcionarse durante el inicio de sesión. Desde la aplicación podrán realizarse peticiones para crear nuevas cuentas si no se dispone de una, pero entonces será necesario esperar a que un administrador dé el visto bueno y la active. Para poder crear estas peticiones será necesario enviar ciertos datos personales que identifiquen al solicitante.

Estas cuentas se pueden crear de dos formas: utilizando el formulario de registro de la aplicación o rellenando un formulario de papel y entregándoselo al administrador, que es el método que se ha estado utilizando hasta el momento. Las cuentas que estén creadas mediante el formulario a papel también tendrán que tener acceso a la aplicación y a todas sus funcionalidades, es decir, independientemente del método de registro se tendrá que generar el mismo tipo de cuenta. Para hacer esto posible, este proyecto se adaptará a la forma de crear cuentas actual del DIPC, utilizando un *script* creado por el Centro de Cálculo para ayudarles en este proceso. Este *script* será adaptado por el Centro de Cálculo para que se pueda utilizar desde el servidor web y, en consecuencia, por la aplicación.

4.1.2. Requisitos funcionales

Actores

En esta aplicación tomarán parte tres actores, que dispondrán cada uno de su propio conjunto de casos de uso. Los actores son:

- **Investigador**: Un investigador es un usuario con una cuenta de cálculo activa. Tendrá acceso al área de funcionalidades de un usuario corriente.
- **Administrador**: Un administrador es un investigador que posee privilegios extra. Además de poder acceder al área de un usuario normal, tendrá acceso también a la de administrador, con funcionalidades nuevas.
- **Otro**: Este usuario se refiere a aquellos que aún no hayan iniciado sesión, por lo que no se es capaz de asignarles un rol específico. Después de iniciar sesión, este usuario pasará a ser un Investigador o un Administrador.

Casos de uso

La aplicación deberá contener los casos de uso establecidos en el acuerdo, que son:

- **Solicitud de cuentas de cálculo**: Un usuario de tipo Otro deberá ser capaz de rellenar un formulario de registro que solicite los campos necesarios para generar la cuenta.
- **Inicio de sesión**: Un usuario de tipo Otro deberá ser capaz de iniciar sesión, proveyendo los credenciales de su cuenta de cálculo. Si dichos credenciales están asociados a una cuenta de Investigador, se redirigirá al área de usuarios. Si en cambio están asociados a una cuenta de Administrador, se le dará la opción de acceder al área que desee.
- **Consulta de los trabajos de cálculo**: Un Investigador deberá ser capaz de consultar la cola de trabajos del supercomputador ATLAS y comprobar el estado de sus trabajos.
- **Consulta de los directorios de trabajo**: Un Investigador deberá ser capaz de navegar por los directorios de ATLAS a los que tenga acceso y acceder al contenido de ciertos ficheros. Principalmente, los ficheros a los que deberá tener acceso serán los ficheros de *output* y de errores de los trabajos. También se le permitirá acceder a cualquier otro fichero regular que no esté en formato binario.
- **Estadísticas personales**: Con cierta frecuencia se generan ficheros que contienen información sobre el uso del clúster. Un Investigador deberá ser capaz de acceder a esta información y ver **sus** estadísticas de uso.
- **Contactar con un Administrador**: Un Investigador deberá ser capaz de enviar mensajes directos a un Administrador en concreto para notificar problemas o solicitar ayuda. Estos mensajes funcionarán de forma similar a un foro, donde cada hilo contiene varios mensajes. También deberá ser capaz de consultar los hilos que ya hayan sido cerrados.
- **Crear cuentas de cálculo**: Un Administrador deberá ser capaz de revisar las solicitudes de cuentas de cálculo creadas por un usuario Otro. Si el Administrador decide que la solicitud es válida, rellenará ciertos campos adicionales y la aplicación creará la cuenta automáticamente. Después se le mostrará una serie de datos que demuestren que la cuenta ha sido generada correctamente. Si el Administrador lo decide, deberá ser capaz también de rechazar solicitudes y de modificar cualquier campo de la solicitud, exceptuando el campo de correo electrónico.
- **Comprobación de temperaturas**: Un Administrador deberá ser capaz de ver la temperatura ambiente de la sala donde se encuentra la fuente de alimentación de

los superordenadores así como las temperaturas de entrada y salida del aire de las máquinas de frío.

- **Estado de ATLAS**: Un Administrador deberá ser capaz de visualizar el uso que se está haciendo del supercomputador ATLAS. Más específicamente, la información que se desea extraer es el número de nodos que están activos del total y el número de núcleos que estén siendo utilizados.

- **Estado de la micro-informática**: Un Administrador deberá ser capaz de comprobar el estado de los todos los aparatos informáticos que estén siendo monitorizados en el servidor de Nagios. En concreto, qué máquinas se encuentran activas/inactivas y los errores o avisos que se reciban.

- **Estadísticas propias de la aplicación**: Un Administrador deberá ser capaz de visualizar ciertas estadísticas de uso de la aplicación. Algunas de estas estadísticas serán: número de inicios de sesión por mes, número de formularios de registro, número de nuevas cuentas de cálculo, etc.

- **Responder a un Investigador**: Un Administrador deberá ser capaz de ver las consultas generadas por un Investigador desde el caso de uso *Contactar con un Administrador* y responder a estos mensajes. Además, ambos usuarios podrán cerrar el hilo una vez decidan que ya se ha solucionado el problema. También deberá ser capaz de consultar los hilos que ya hayan sido cerrados.

Con esta información, se ha generado el diagrama de casos de uso de la figura 4.1.

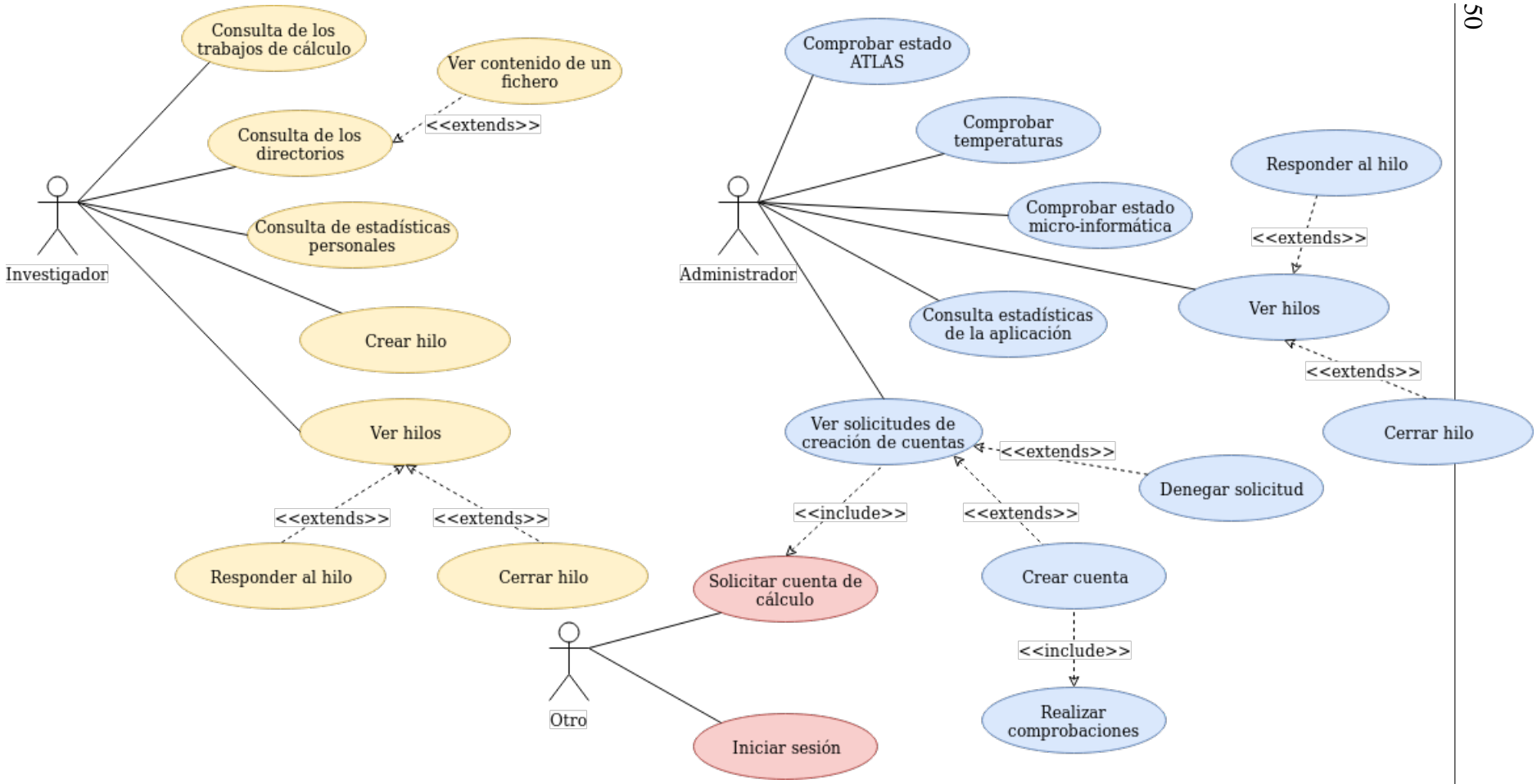

Figura 4.1: Diagrama de casos de uso.

5. CAPÍTULO

Diseño

El Diseño es la fase del ciclo de vida de un proyecto de Software donde se propone una solución a los requisitos planteados en la fase de Análisis y se específica el comportamiento de cada caso de uso. En esta etapa se generan diagramas que permiten comprender el funcionamiento del producto final a alto nivel, así como de cada uno de sus módulos. Además, sirve para tener una referencia a la que ajustarse a la hora de programar en el periodo de implementación.

En este capítulo se expondrá un diagrama de la arquitectura de la aplicación, para poder comprender mejor el papel que juega cada elemento, así como un diagrama de clases del proyecto, que sirva para entender el modelo de datos que se utilizará. Después, se planteará un diagrama de secuencia para cada caso de uso enumerado en el punto 4.1.2.

5.1. Arquitectura de la aplicación

Uno de los aspectos más particulares del proyecto que se va a implementar es la arquitectura del sistema. Si se pretende comprender los diagramas de secuencia de este mismo capítulo y, en general, el comportamiento del producto, será necesario entender esta arquitectura debidamente. Nuestra arquitectura, como muchas otras, se puede dividir en dos partes claramente distinguidas: cliente o *front-end* y servidor o *back-end*. La parte **cliente** se refiere a la aplicación que estará instalada en los *smartphone*. Este elemento sigue la estructura de Angular, donde cada página tiene un módulo, una vista, un controlador y un fichero de estilo. La parte **servidora** está compuesta de varios elementos diferentes,

como se puede ver en la figura 5.1, y cada uno de estos elementos se va a explicar a continuación.

Todo el *back-end* está ubicado dentro de la red interna del DIPC (que a su vez pertenece a la red de la UPV/EHU). A esta red solo tienen acceso las máquinas que estén ubicadas dentro de la misma red (incluso hay algunas zonas restringidas dentro, pero no es relevante para este caso), por lo que el servidor web estará instalado en un servidor de la red. Este servidor habrá que configurarlo para que sea accesible desde el exterior. De esta manera, los usuarios serán capaces de enviar peticiones a la API del servidor desde la aplicación instalada en su dispositivo, desde cualquier parte del mundo con conexión a Internet.

Entre el cliente y el servidor la comunicación se realiza mediante HTTP, como en la mayoría de navegadores. Angular nos ofrece una API llamada *HTTPClient* que se encarga de todo y es muy sencilla de integrar en cualquier proyecto. Esta API se basa en la interfaz *XMLHttpRequest*, que es la misma interfaz que utiliza la tecnología AJAX[1] para cargar objetos de un servidor sin tener que recargar la página entera, por lo que no se interrumpe la interacción con el usuario. Entre el servidor web y el resto de elementos la forma de comunicarse varía, aunque se intentará utilizar el protocolo SSH en la medida de lo posible, para hacer que la aplicación trabaje de la misma manera en que lo haría un usuario desde su equipo de trabajo, y que de cara al administrador sea transparente.

[1] https://en.wikipedia.org/wiki/Ajax_(programming)

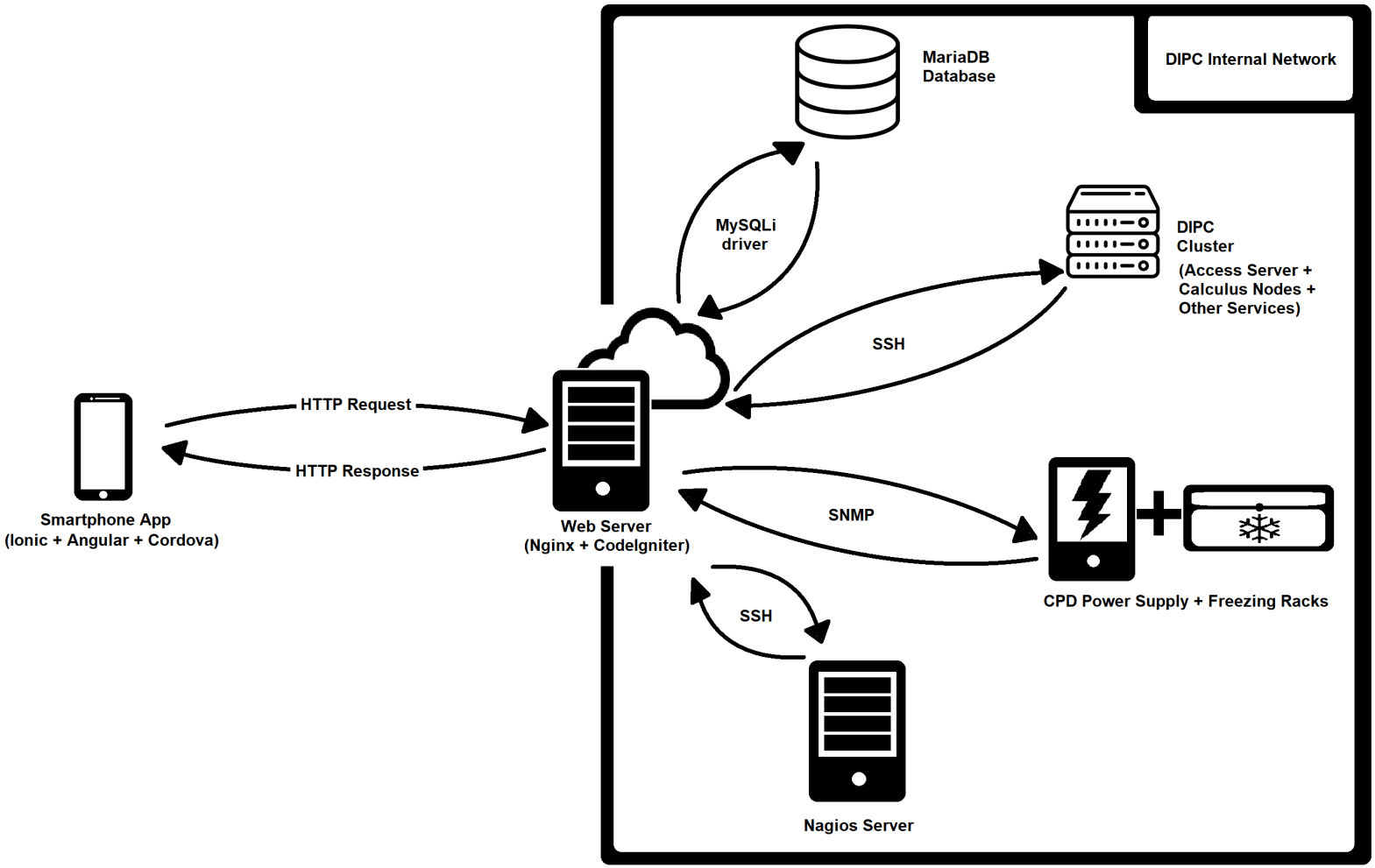

Figura 5.1: Esquema de la arquitectura de la aplicación.

Respecto al *front-end*, hay que comentar que la aplicación, al utilizar Angular, tiene una arquitectura llamada *Component-based architecture*. Bajo esta arquitectura no existen los Controladores ni ViewModels de la versión anterior, sino que está compuesta por unos elementos llamados Componentes. Los Componentes son cualquier clase de la aplicación que contenga la directiva (*decorator*) *@Component* y están compuestos por tres elementos: la **plantilla** (*template*), que se puede comparar con una vista de la arquitectura MVC; la **clase** (*class*), que contiene toda la lógica del componente; y los **metadatos**, que se refiere al conjunto de directivas que pueden encontrarse dentro de un Componente. Los Componentes más comunes dentro de nuestra aplicación son las **páginas**, que son conjuntos de elementos visuales que se presentan en una pantalla y tienen asociada una lógica por detrás. Estas páginas utilizan **servicios**, que proveen funcionalidades que no estén estrictamente relacionadas con las páginas. Estos servicios se pueden inyectar en las páginas mediante dependencias (los servicios contienen el decorador *@Injectable*) y uno de los ejemplos más claros de servicio en nuestra aplicación es el *NetworkCheck*, que se encargará de la API *HTTPClient* mencionada anteriormente.

Los cuatro sistemas que forman parte del *back-end* (excluyendo el servidor web) son:

Cada **clúster del centro** es el que da acceso a la información sobre la cola de trabajos de ese clúster. Para acceder a esta información existen multitud de comandos que habrá que ejecutar desde estas máquinas. Además de esto, el DIPC proporciona un disco llamado *scratch* para que los investigadores almacenen ahí cualquier tipo de información que necesiten. Uno de los usos más comunes es almacenar el código de los programas que ejecutan, los *inputs* de datos que utilizan, o los resultados de la ejecución de los trabajos. Este disco está montado en los clústeres de cálculo del centro, y cada clúster tiene el suyo propio.

Dentro de la red del DIPC también se encuentran las **máquinas encargadas de mantener en correcto estado el CPD** del centro. De todas ellas, algunas de las más importantes son la protección del suministro eléctrico y las máquinas de frío. No es importante entender las funciones de estas máquinas, pero sí es importante saber que, entre otras cosas, monitorizan las temperaturas del CPD. Estas temperaturas son información importante para el centro de cálculo, que hasta el momento no tiene otra forma de consultarlas más que accediendo a la interfaz web de cada una de ellas. Como estás máquinas no disponen de un sistema operativo personalizable, la única alternativa que se nos proporciona es extraer la información desde el protocolo SNMP. Como ya se ha dicho, el protocolo que soportan es el v2c, y la única comunidad que está activada es la *public*, que solo otorga permisos de lectura. La información que provee esta comunidad no necesita ser secreta. Por ejemplo,

para el centro no es un problema que cualquiera dentro de la red sepa las temperaturas del CPD o la tensión máxima soportada por la protección del suministro eléctrico.

Además de todo lo mencionado anteriormente, el Centro de Cálculo es el encargado del mantenimiento de todos los aparatos electrónicos del centro, como son los ordenadores, impresoras, pantallas, *routers*, etc. A todas estas máquinas las denominamos 'micro-informática'. Existe un **servidor Nagios** pensado para monitorizar la micro-informática y poder encontrar problemas en cualquier máquina que esté configurada. Las conexiones a este servidor se suelen hacer mediante HTTP con una interfaz web, pero este método de acceso no sería posible (o sería demasiado complejo) para nuestro sistema. La solución que se ha pensado es acceder mediante SSH a la máquina que aloja el servidor y ejecutar 'manualmente' (desde el servidor web) los *scripts* que ejecuta Nagios para obtener la información de los equipos (los que se han explicado en el apartado sobre Nagios del capítulo 3) y mostrando una versión personalizada de su *output*. Para esto, habrá que recorrer toda la lista de máquinas monitorizadas y ejecutar para cada una su *script* correspondiente.

Por último hay que hablar sobre la **base de datos** que se va a instalar en este proyecto. Esta base de datos contendrá información que no esté almacenada en ningún otro sitio de toda la arquitectura, por lo que almacenará datos que solo incumban a la aplicación. Esta decisión se ha tomado para evitar la duplicación de datos (como almacenar información sobre las cuentas tanto en la base de datos como en los servidores NIS/LDAP del centro), ya que si se duplicase requeriría un trabajo extra para mantener todos los datos sincronizados y actualizados en ambos servidores. Toda la información que se va a almacenar se explica de forma detallada en la sección 5.2.

En el DIPC, la información de un usuario no se almacena en un solo lugar, sino que dependiendo del tipo de información que sea, se guarda en diferentes sitios. Por ejemplo, para el caso de un nuevo investigador:

- La información que proporciona al rellenar el formulario de registro se almacena en el servidor LDAP junto con información extra que añade el administrador, como los grupos de UNIX a los que pertenece (esto luego le dará acceso a diferentes directorios), o el servidor de acceso que debe usar para conectarse.
- La información correspondiente con la persona en sí se almacena en una base de datos llamada *admin-01*, que contiene datos como el número de teléfono, edificio y despacho que tiene asignados, si está alojado en el centro o no, y las fechas de estancia, entre otros.

- El servidor NIS actualmente está en proceso de ser sustituido por LDAP, pero sigue siendo necesario insertar la misma información para que ciertas funciones se comporten correctamente.

- Además de todo esto, hay cierta información relacionada con el centro de cálculo que se guarda en ciertos ficheros creados específicamente para esta función. Son datos como: la lista de clústeres de cálculo a la que tiene acceso el investigador, el número de créditos que se le va a asignar, si el investigador tiene permiso para calcular o no, el espacio en disco máximo que puede utilizar, etc.

Para que el proceso de crear nuevas cuentas sea exitoso, todas estas conexiones a diferentes servidores se hacen desde un *script* que está creado para gestionar todo lo relacionado con las cuentas. Este *script* también comprueba que la inserción del usuario ha sido exitosa en cada uno de los pasos listados.

5.2. Esquema de la base de datos

Para poder implementar ciertas funcionalidades de la aplicación, será necesario almacenar información en la base de datos y trabajar con ella. Para esto, es imprescindible definir un buen esquema que permita su fácil manejo y no dificulte tareas como el mantenimiento o la ampliación de la base de datos, es decir, que sea escalable. Sin embargo, este esquema tendrá que tener sentido dentro del sistema que se ha explicado en la sección sobre la arquitectura. Es decir, con la base de datos se pretende dar acceso a nueva información o facilitar el acceso a información que no era sencilla de conseguir antes. También se pretende hacer que la base de datos requiera el mínimo mantenimiento posible, permitiendo a los administradores acceder a su información desde la aplicación y evitando el acceso directo excepto en casos en los que no quede otra alternativa.

Para poder realizar el registro de nuevas cuentas de la manera en la que se pretende, utilizando dos actores que completen parcialmente los datos necesarios para la inserción, será necesario hacer uso de la base de datos. Habrá que almacenar toda la información ingresada por el actor Otro al completar el formulario de registro hasta que un Administrador decida lo que hacer con dicho formulario. Si se decide rechazar el formulario, habrá que eliminar esta información de la base de datos. Si, en cambio, se decide aceptar el formulario, el Administrador deberá completar la información restante del formulario y crear el nuevo usuario. Los nuevos campos rellenados por el Administrador no se almacenarán,

aunque si que se actualizarán los campos que introduzca el actor Otro y el Administrador haya modificado. Los campos que se deberán almacenar son:

- **Nombre de usuario**: Se refiere al nombre de la cuenta final, necesario para iniciar sesión e identificar al usuario, entre otras muchas cosas. Deberá ser un nombre de usuario que no esté registrado con anterioridad, aunque cabe mencionar que no tiene por qué ser el nombre que se le vaya a dar finalmente a la cuenta, ya que un Administrador podría modificarlo (en caso de no ser apropiado, por ejemplo).

- **Contraseña**: Este campo se refiere a la contraseña que el usuario introduzca y que utilizará para iniciar sesión (por lo menos la primera vez, hasta que la cambie). El valor de este campo se eliminará en cuanto el Administrador acepte o rechace la cuenta, por cuestiones de seguridad. Además, la contraseña no se almacenará en texto plano, sino que se cifrará con el algoritmo AES-128[2] en modo CBC (el que utiliza CodeIgniter por defecto). Esta forma de almacenamiento de contraseñas es subóptima, puesto que es posible para un atacante obtener las contraseñas originales si consigue la clave utilizada para cifrarlas. Sería más conveniente utilizar una función criptográfica específica para almacenamiento de contraseñas (como PBKDF2[3] o bcrypt[4]), pero no es posible ya que es necesario obtener la contraseña original a la hora de introducirla en el servidor LDAP. Sin embargo, este método de almacenamiento se considera suficiente ya que estas contraseñas no pertenecen a usuarios que ya han sido registrados (porque las contraseñas de las cuentas registradas se eliminan), por lo que si alguien obtiene acceso a ellas seguirá siendo incapaz de acceder a los sistemas del centro.

- **Apellidos**: Se refiere a los apellidos de la persona solicitando una cuenta.

- **Nombre**: Se refiere al nombre de la persona solicitando una cuenta.

- **Posición**: Se refiere al cargo o título académico que la persona solicitando una cuenta ostenta. Por defecto, se ofrecen valores como PhD, Post-doc, Senior o Visitor, aunque el usuario podrá introducir cualquier otra posición.

- **Edificio**: Se refiere al edificio donde se encuentra el despacho del usuario solicitando una cuenta. Este campo es opcional, ya que puede darse el caso de que no se le haya asignado aún o de que no vaya a trabajar desde el centro.

[2] https://en.wikipedia.org/wiki/Advanced_Encryption_Standard
[3] https://en.wikipedia.org/wiki/PBKDF2
[4] https://en.wikipedia.org/wiki/Bcrypt

- **Despacho**: Se refiere al número del despacho del usuario solicitando una cuenta. Campo opcional, por la misma razón que el anterior.
- **Teléfono**: Se refiere al número de teléfono que el usuario solicitando una cuenta quiere que se utilice para contactar con él. Es opcional, ya que por defecto se utiliza un correo electrónico como método de contacto, y normalmente se refiere al número de teléfono del despacho donde se encuentra el usuario, aunque puede no serlo.
- **Correo electrónico**: Se refiere al correo electrónico del usuario solicitando una cuenta. Necesario para contactar con él.
- **Segundo correo electrónico**: Se refiere a un correo electrónico secundario del usuario solicitando una cuenta. Campo opcional, se utilizará solo en caso de que el principal no funcione.
- **Supervisor**: Se refiere al supervisor del usuario solicitando una cuenta, en caso de que lo tuviese.
- **Fecha de inicio**: Se refiere a la fecha desde la que se espera tener la cuenta activa.
- **Fecha final**: Se refiere a la fecha hasta la que se espera tener la cuenta activa.
- **Financiación**: Se refiere al presupuesto con el que cuenta la investigación del usuario solicitando una cuenta. Es un campo opcional.

Además de estos campos, que son los que tiene que completar el actor Otro, deberán existir otros dos campos que serán necesarios para la gestión de la base de datos:

- **Aceptado**: Se pretende mantener almacenada la información de los formularios que ya han sido aceptados, aunque de momento no haya una función específica para ellos. Por esta razón, será necesario distinguir entre los formularios que se hayan registrado y los que aún queden pendientes (los rechazados se eliminará, como ya se ha explicado). Con un campo *boolean* será suficiente para hacer la distinción.
- **Política de privacidad**: Para cumplir con el artículo 7 del Reglamento general de protección de datos[5] (en adelante RGPD) es necesario almacenar una prueba que indique que el usuario ha aceptado la política de privacidad del centro. Para esto será suficiente con un campo *boolean* que indique si ha sido aceptada o no. En este

[5] https://www.boe.es/doue/2016/119/L00001-00088.pdf

campo debería estar siempre con valor *true*, porque si no se ha aceptado la política de privacidad el usuario no debería poder enviar el formulario ni el centro podría almacenar información sobre él.

Para poder mostrar las estadísticas de uso de la aplicación, como no hay ninguna otra ubicación actualmente para hacerlo, se almacenarán también en la base de datos. Estas estadísticas tendrán tres campos y ninguno de ellos se utilizará como clave primaria, ya que no es necesario identificar ninguna estadística de forma única. Los campos serán: **nombre de usuario** que está realizando la acción (si lo hubiese), **acción** que se está realizando, y **fecha** en la que se realiza la acción. Estos dos últimos campos servirán para filtrarlas.

Por último, se necesitará almacenar también toda la información referente a los hilos que se creen entre los Investigadores y los Administradores, así como sus mensajes, ya que es una funcionalidad que no existía hasta el momento y es el sitio idóneo para recoger estos datos. Este sistema estará compuesto por dos tablas, una para los hilos y otra para los mensajes de los hilos.

La tabla referente a los hilos tendrá los campos:

- **Identificador**: Clave primaria, para poder diferenciar los hilos entre sí. Será auto-incremental.
- **Autor**: Usuario que ha creado el hilo.
- **Administrador**: Administrador al que va dirigido el hilo.
- **Título**: Título del hilo. Podrá haber títulos de hilo repetidos incluso con el mismo autor y administrador.
- **Cerrado**: *Boolean* que indica si el hilo está cerrado o no.

La tabla referente a los mensajes de lo hilos tendrá una relación N:1 con la tabla de hilos, es decir, cada hilo podrá tener múltiples mensajes pero cada mensaje solo pertenecerá a un hilo. Los campos que forman esta tabla son:

- **Identificador**: Parte de la clave primaria, junto con el siguiente campo. Identificará al mensaje y será auto-incremental.
- **Identificador del hilo**: Parte de la clave primaria y clave extranjera que hace referencia al identificador de la tabla hilo.

- **Remitente**: Nombre de usuario que ha enviado el mensaje.
- **Contenido**: Contenido del mensaje.
- **Fecha**: Fecha en la que se ha enviado el mensaje.

Todas estas tablas irán unidas a una última tabla que sirve como enlace. Esta tabla recogerá el conjunto de usuarios que hayan iniciado sesión o se hayan registrado con la aplicación. Solo tendrá un campo, que será su clave primaria, y contendrá el nombre de usuario al que hace referencia.

Con esta estructura se ha generado la figura 5.2.

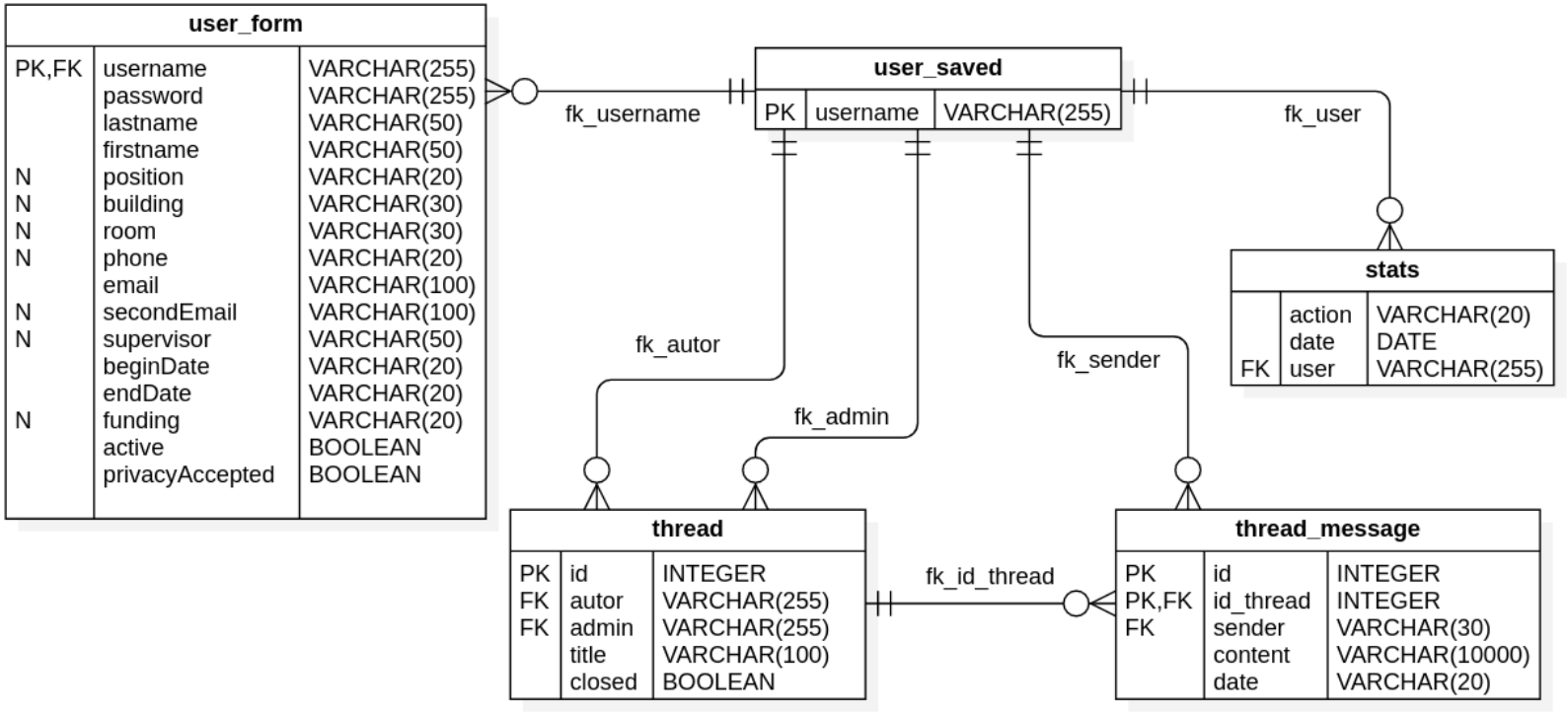

Figura 5.2: Esquema de la base de datos.

5.3. Diagramas de secuencia

Para cada caso de uso identificado en el diagrama de casos de uso de la figura 4.1 se ha realizado un diagrama de secuencia que ejemplifica el flujo de eventos del caso de uso al que pertenece. El objetivo de estos diagramas es mostrar de forma clara cómo funciona cada caso de uso y los elementos que están presentes en él. Con el fin de simplificar el diagrama y de mantener la legibilidad, se han omitido las secuencias de eventos en las que ocurren errores.

Como se puede apreciar en los diagramas, en algunos eventos se ha especificado el protocolo y/o método mediante el que se comunican dos procesos. Si se mencionan, es porque se considera que es un elemento importante dentro del flujo de eventos. Por ejemplo, cuando la aplicación contacta con el servidor web se menciona el protocolo (HTTP) y el método (POST) porque es relevante para el envío de información entre las dos partes. Sin embargo, cuando se accede a la base de datos no se dice de qué manera se efectúa la transacción porque, aunque se utilice el *driver* MySQLi, cualquier otro nos otorgaría el mismo resultado gracias a la abstracción que ofrece CodeIgniter al trabajar con bases de datos.

5.3.1. Solicitud de cuentas de cálculo

En la figura 5.3 se puede ver el diagrama de secuencia de este caso de uso. La validación se hace de manera automática con los *Reactive Form* de Angular, que evitarán que se envíe el formulario y mostrarán qué campos no son válidos. Como ya se ha dicho, el servicio Network Check es la capa de la aplicación encargada de la comunicación con el servidor web. La función *connected()* comprueba que el dispositivo disponga de conexión a Internet antes de permitir el envío de los datos. Como se podrá ver más adelante esta función solo se utiliza en ciertos casos de uso, únicamente en los que se considera que se está realizando una acción crítica.

A la hora de crear un formulario de registro hay una comprobación que no se puede realizar sin acceder a los servidores del DIPC, que es la acción de comprobar que el nombre de usuario introducido no exista. Esta comprobación se hace únicamente en los servidores del DIPC (donde se almacenan las cuentas ya existentes). Al parecer, es común que un usuario que quiera crear una cuenta envíe varios formularios similares (normalmente por introducir algún dato erróneo) y tendrán que ser capaces de enviar estos formularios con

un nombre de usuario similar. Por esto, se ha decidido que sea el administrador sea el que compruebe los campos y haga la ”mezcla” entre los diferentes formularios.

5.3.2. Inicio de sesión

En la figura 5.4 puede verse el diagrama de secuencia para el inicio de sesión en la aplicación. Se pueden apreciar ciertas peculiaridades. Por una parte, para comprobar si las credenciales introducidas son correctas no se hace contra una base de datos donde estén almacenados estos credenciales, sino que se intenta realizar una conexión mediante SSH con el servidor de acceso, que es la manera en la que se conecta un usuario cuando quiere acceder a los servidores de cálculo. Además, se compara el nombre de usuario con los nombres de usuario de los administradores (los que pertenecen a cierto grupo del servidor) para decidir si tendrá acceso o no a la parte de administración. También habrá que introducir el usuario en la tabla 'user_saved' de la base de datos en caso de que sea la primera vez que se conecte mediante la aplicación. Por último, este es el único momento en el que se genera un token, que se utilizará para dar acceso a gran parte de las funcionalidades de la aplicación.

Al final del flujo de eventos, si el usuario es un administrador se le mostrará una alerta en la que tendrá que decidir si quiere acceder al área de administración o no. No se va a permitir la navegación entre estas áreas sin cerrar sesión por motivos de seguridad.

5.3.3. Consulta de los trabajos de cálculo

Cuando un Investigador quiera consultar su lista de trabajos en la cola, podrá entrar en la página correspondiente (o clicar el botón de actualizar, si ya se encuentra en ella). El proceso es parecido al resto de casos de uso, hasta llegar al servidor web. El servidor web ejecutará un *script* que se ha diseñado específicamente para este caso de uso, que muestra los trabajos en cola de un usuario en formato JSON.

Por cada trabajo en cola se devuelven diferentes campos, dependiendo de si está en ejecución o en espera (los campos de los trabajos en espera son un subconjunto de los campos de ejecución). La lista entera es:

- Jobid: Id del trabajo.
- Job Name: Nombre que el usuario le ha dado al trabajo.

- Job State: Estado del trabajo (Running o Queued).
- Path: Ruta donde se generarán los ficheros de output y errores del trabajo.
- Host: Lista de nodos en los que se está ejecutando en trabajo.
- CPUT: Tiempo de CPU que el usuario ha reservado para el trabajo.
- Mem: Memoria que el usuario ha reservado para el trabajo.
- Nodes: Número de nodos que el trabajo está utilizando.
- PPN: Número de procesadores que el trabajo está utilizando por nodo.
- Start Time: Fecha de inicio del trabajo.
- Elapsed Time: Tiempo de ejecución actual del trabajo. Se calcula en base a Start Time y a la fecha actual del servidor.

Se puede ver el flujo de eventos del caso de uso en la figura 5.5.

5.3.4. Consulta de los directorios

Cuando un Investigador decida consultar su directorio de trabajo, el objetivo será simular de la manera más fiel posible un explorador de directorios/ficheros que permita explorar cualquier árbol de directorios (los directorios de los investigadores suelen tener gran cantidad de subdirectorios y nombres largos) y acceder al contenido de los archivos que estén ubicados allí. Es por esto que por cada fichero que contenga el directorio al que se está accediendo, habrá que cargar su nombre, tipo (fichero regular, directorio u otros) y tamaño, para poder establecer un máximo a la hora de mostrar el contenido y limitar así descargas muy lentas (mostrar el contenido de un fichero de 1GB supondría una descarga de aproximadamente 1GB, que puede tardar varios minutos). Se puede ver el diagrama de secuencia en la figura 5.6.

El contenido de cada fichero no se va a enviar en este caso de uso y se esperará a que el usuario decida qué fichero quiere ver para cargar su contenido. Si se supiese que el tamaño de estos ficheros es siempre bajo, cargarlos en este momento podría ser beneficioso para minimizar el tiempo de respuesta y el número de peticiones al servidor. Sin embargo, sabiendo que muchos de estos ficheros son pesados, se ha decidido cargar únicamente los que el usuario necesite y establecer un tamaño máximo de fichero de 25MB (se podrá modificar más adelante).

5.3.5. Ver contenido de un fichero

El diagrama de secuencia de este caso de uso se puede ver en la figura 5.7. Para cargar un fichero de los listados en el directorio, bastará con hacer *click* sobre él. El servidor web tendrá que comprobar si el tamaño del fichero entra dentro del rango válido y si el fichero es legible (deberá ser un fichero regular y de contenido no binario).

El contenido del fichero se cargará de una sola vez, pero en la aplicación no se mostrará entero, solo los primeros 10240 caracteres. Esta decisión se ha tomado por una limitación en el WebView que utiliza Cordova para la versión de Android, que tiene problemas para cargar contenidos más grandes. Si hay parte del fichero aún por mostrar, se añadirá un botón que permita cargar el siguiente trozo de caracteres, sin tener que contactar con el servidor. Se podrán seguir cargando trozos hasta que no quede contenido del fichero por mostrar.

5.3.6. Consulta de estadísticas personales

En la figura 5.8 se puede apreciar el comportamiento de este caso de uso. Es parecido al caso de uso 'Consulta de los trabajos de cálculo', pero se diferencia en el programa al que se llama desde el servidor web y en la información que devuelven.

En este caso, se llama a un programa llamado 'new-ur' con el parámetro '-j', que servirá para obtener el resultado en formato JSON. Este programa deberá cargar de los ficheros de *accounting* del Centro de Cálculo las estadísticas que se han almacenado del usuario que esté haciendo la consulta. Los datos que se cargan son:

- Credits: Número de créditos consumidos por el usuario.
- Max Credits: Número máximo de créditos de los que dispone el usuario.
- Computing Hours: Horas de cálculo que necesitado el usuario para sus trabajos.
- Memory: Memoria total que ha reservado el usuario para sus trabajos.
- Jobs: Número total de trabajos que ha lanzado el usuario.

5.3.7. Crear hilo

Para poder crear un hilo habrá que rellenar el formulario correspondiente en el que se introduce el administrador al que va dirigido, el título del hilo y el contenido del primer mensaje. Todo esto se envía a través del servicio Network Check al servidor web, que insertará el hilo en la tabla *thread* y el primer mensaje en la tabla *thread_message* de la base de datos asociado al hilo recién introducido. Para seguir el diagrama de secuencia, ver figura 5.9.

5.3.8. Ver hilos

Si seguimos la figura 5.10 veremos que nada más cargar la página *Report*, se mandará una petición al servidor web para cargar todos los hilos asociados al usuario que esté conectado. En este caso, a diferencia del caso de uso Consulta de los directorios, sí que se carga el contenido de todos los hilos a la vez. La razón es que, incluso un administrador que pueda tener múltiples hilos abiertos y que estos hilos contengan varios mensajes, probablemente no lleguen a un tamaño total superior a 25MB.

5.3.9. Responder al hilo

Cuando el usuario escriba una respuesta y decida enviarla, se mandará el texto de la respuesta junto con el id del hilo al que pertenece, como se puede comprobar en la figura 5.11. Cuando el mensaje llegue al servidor y se valide el token, se almacenará está información en la base de datos, junto con el usuario que ha enviado la respuesta y la fecha actual. Por último, se volverá a cargar la página para mostrar al usuario los datos actualizados.

5.3.10. Cerrar hilo

Cerrar un hilo es un caso de uso muy simple, donde haciendo clic sobre un botón se envía al servidor web el id del hilo que se quiere cerrar. Después de validar el token, se cambia el estado del campo *closed* a 1, indicando que está cerrado. Así, la próxima vez que cualquiera de los participantes del hilo accedan al caso de uso 'Ver hilos', no se mostrará el hilo que se acaba de cerrar, como cuando se actualiza la vista al final de este caso de uso.

5.3.11. Comprobar estado ATLAS

El flujo de eventos en este caso de uso también es parecido al caso de uso 'Consulta de los trabajos de cálculo' y se puede ver cómo funciona en la figura 5.13. Aquí también habrá que generar *script*s que nos devuelvan la información que necesitamos, en este caso dos de ellos: el primero para recoger la información relativa a cada nodo y el segundo para recoger la información global del sistema.

Nada más un Administrador entre en la página, se procederá a ejecutar estos dos *script*s. El primero obtendrá la información del comando *pbsnodes*, en concreto de la opción '-l', que lista el conjunto de nodos en un estado en concreto. Al final de la ejecución del *script* se deberá conocer a qué estado pertenece cada nodo y el número de nodos que hay en cada estado. El segundo *script* obtendrá la información del comando *showq*, aunque solo se obtendrán las dos líneas que nos interesan: número de procesadores activos del total y número de nodos activos del total (junto con sus porcentajes).

5.3.12. Comprobar temperaturas

Este caso de uso es el único de toda la aplicación que utiliza el protocolo SNMP, como se puede ver en la figura 5.14. Para poder usar este protocolo será necesario instalar un paquete en el servidor web que habilite el comando *snmpwalk* (en caso de CentOS 7 es *net-snmp*).

La llamada desde el controlador se hace de forma periódica, nada más cargar la página y cada 3 segundos, para dar una sensación al usuario de que los datos están modificándose en vivo. Las temperaturas suelen cambiar cada poco tiempo por lo que es probable que cada vez que se actualicen haya algún dato diferente.

En el diagrama no se ha indicado el OID en concreto que se está consultando, se comentarán en el capítulo 6, pero se puede ver cómo se consulta a la fuente de alimentación de la CPD para conocer la temperatura ambiente de la sala y cómo se consulta 2 veces a cada máquina de frío para conocer la temperatura del aire que reciben y que devuelven.

Toda esta información se codifica en formato JSON, estructurando cada dato en su posición correspondiente. Así, si en un futuro hubiese que realizar una ampliación o dar de baja alguna máquina, la aplicación sería capaz de interpretar el nuevo JSON que reciba si mantiene la estructura.

5.3.13. Comprobar estado de la micro-informática

Como se puede comprobar en la figura 5.15, este caso de uso es similar a otros hasta llegar al servidor web. Desde allí, se abrirá una conexión SSH con el servidor Nagios que monitoriza la micro-informática y se ejecutará un *script* que hayamos diseñado para este caso de uso en concreto.

El *script* tendrá que leer los ficheros de configuración *hosts.cfg* y *services.cfg* de Nagios y, por cada objeto que encuentre en ellos con la directiva *check_command*, ejecutar el comando asociado. Estos comandos devuelven el estado de ese host o servicio en concreto, por lo que habrá que ordenar correctamente cada resultado para más tarde poder interpretarlo. Un objeto JSON vuelve a ser la opción ideal, ya que podremos crear los atributos que necesitemos y trabajar con el objeto en cualquiera de los diferentes lenguajes que forman parte del sistema.

Se plantea como alternativa, en caso de que la ejecución del *script* sea lenta (dependerá del número de objetos que se estén monitorizando desde el servidor y de lo que tarde cada comprobación), programar la ejecución del *script* en el *cron*[6] de la máquina y que se escriba el resultado en un fichero. Así, cada vez que se quiera obtener el estado de la micro-informática, solo habrá que leer el contenido del fichero e interpretarlo. Esta alternativa minimiza el tiempo de respuesta del caso de uso, pero tiene como desventaja que los datos que se obtienen pueden no estar actualizados del todo (dependerá de la frecuencia con la que se ejecute el *script* en el *cron*). La decisión entre una u otra opción habrá que tomarla una vez se haya desarrollado el *script* y se haya medido su tiempo de ejecución.

5.3.14. Consultar estadísticas de la aplicación

Si miramos a la figura 5.16, podremos ver que el caso de uso para consultar las estadísticas de la aplicación es directo. Dependiendo del periodo que se vaya a consultar (2, 4, 6 ó 12 meses) habrá que realizar más o menos consultas a la base de datos. Se extrae el número de acciones que ha habido por mes, y se incluye en un *array*. A su vez, se incluye en otro *array* el mes y año al que pertenece el número extraído. Se podría haber estructurado de diferente manera para que ambos datos estuviesen asociados de una manera más directa en vez de por la posición en el *array*, pero se ha decidido hacerlo así porque a la hora

[6] cron: Servicio del sistema operativo Unix que permite la programación de procesos en segundo plano.

de presentarlo en un gráfico, habrá que proveer un *array* para el conjunto de datos y otro para las etiquetas del gráfico, por lo que es más cómodo (se evita tener que ciclar entre objetos y asignar sus atributos a *array*s en el cliente).

5.3.15. Ver solicitudes de creación de cuentas

Se puede ver el flujo de eventos de esta funcionalidad en la figura 5.17. Al entrar en la página *List*, el controlador solicitará la información de los formularios de registro que aún estén pendientes. Si se valida el token y se comprueba que es un administrador, se obtendrá esta información de la base de datos y se devolverá al cliente. Se mostrará al Administrador una lista con los nombres de las cuentas solicitantes y, si el Administrador decide que quiere ver los campos de alguna, podrá hacer clic en su nombre y se accederá a la vista *Register-Admin*. Desde esta vista se da pie a dos casos de uso: **Crear cuenta** y **Denegar solicitud**.

5.3.16. Crear cuenta y realizar comprobaciones

La figura 5.18 muestra cómo funciona el proceso de dar de alta una solicitud de cuenta. Cuando a un Administrador se le muestre el formulario enviado, tendrá que revisar todos los campos, modificar los que crea convenientes y rellenar nuevos campos que solo puede rellenar un Administrador, como el grupo al que va a pertenecer o los privilegios que se le van a dar a la cuenta.

Si se ha modificado el nombre de usuario de la cuenta, habrá que comprobar que no existe una cuenta con ese nombre, como se hace en el registro inicial. También habrá que comprobar que los grupos introducidos existen, para poder asignárselos más tarde.

Si todas las comprobaciones se cumplen, se procederá a enviar el formulario al servidor web. Con los campos del formulario se ejecutará un *script* alojado un servidor del DIPC que está especialmente diseñado para crear nuevas cuentas. Finalmente, se marca el formulario como cerrado y se actualizan las estadísticas de la aplicación. Si el resultado de la ejecución del *script* es 0, significará que todo ha ido bien (y en principio indicaría que la cuenta se ha creado correctamente), por lo que podría acabar el proceso de creación de cuentas aquí. Aún así, cuando se crean cuentas después se comprueba que todo se ha realizado correctamente, mirando que el usuario tenga un directorio *home* o que aparezca en los ficheros de asignación de créditos de cálculo, por ejemplo.

Cuando acabe este proceso, se abrirá una nueva página donde se mostrarán las comprobaciones hechas por el *script*, aunque habrá que personalizar el *output* de manera que se asemeje a la apariencia del resto de la aplicación.

5.3.17. Denegar solicitud

Si al comprobar el formulario de registro el Administrador decide que el formulario no es válido, tendrá que hacer clic sobre el botón de rechazar, que contactará con el servidor web y borrará el formulario de registro de la base de datos, como se explica en la figura 5.19.

5.3.18. Figuras

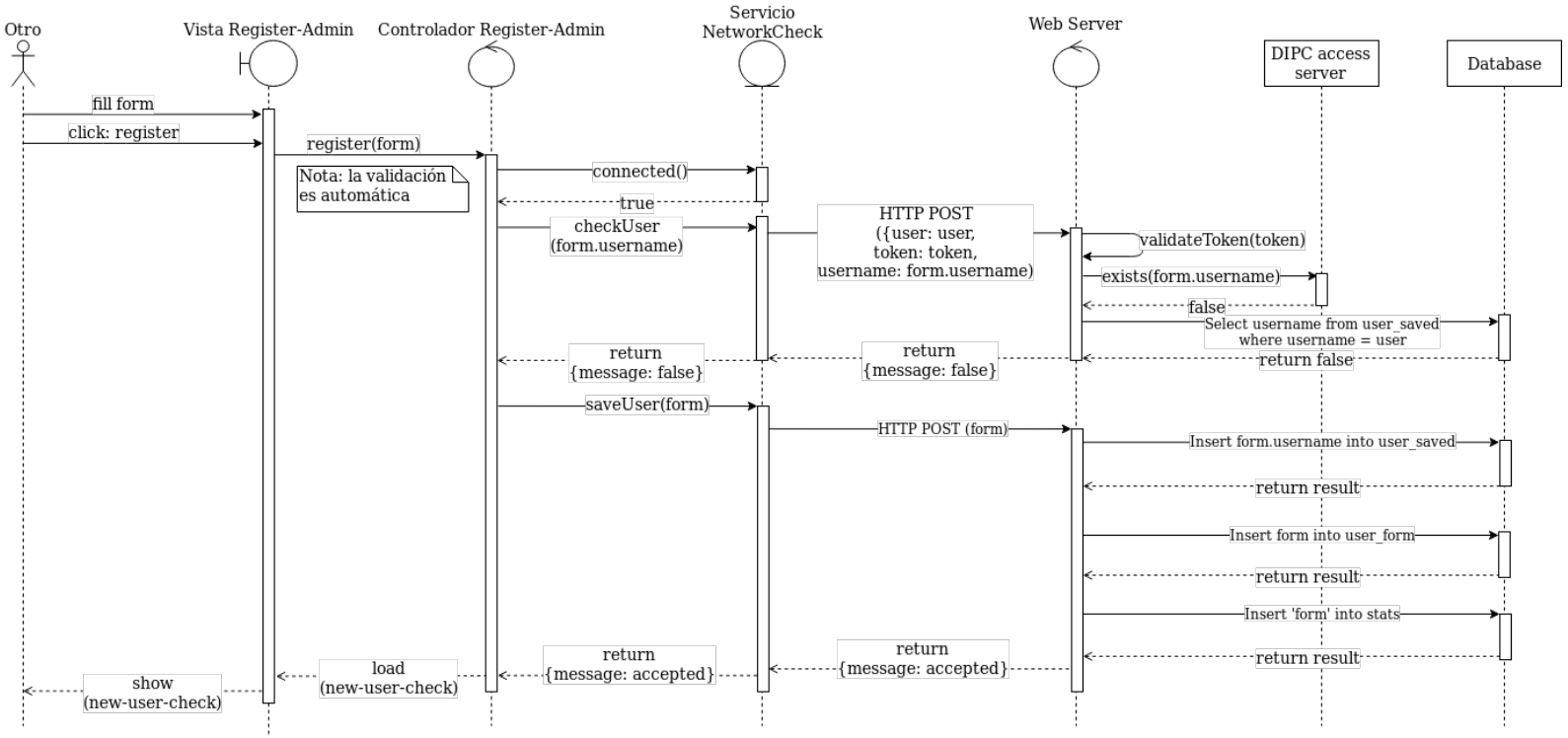

Figura 5.3: Diagrama de secuencia de la solicitud de cuentas de cálculo.

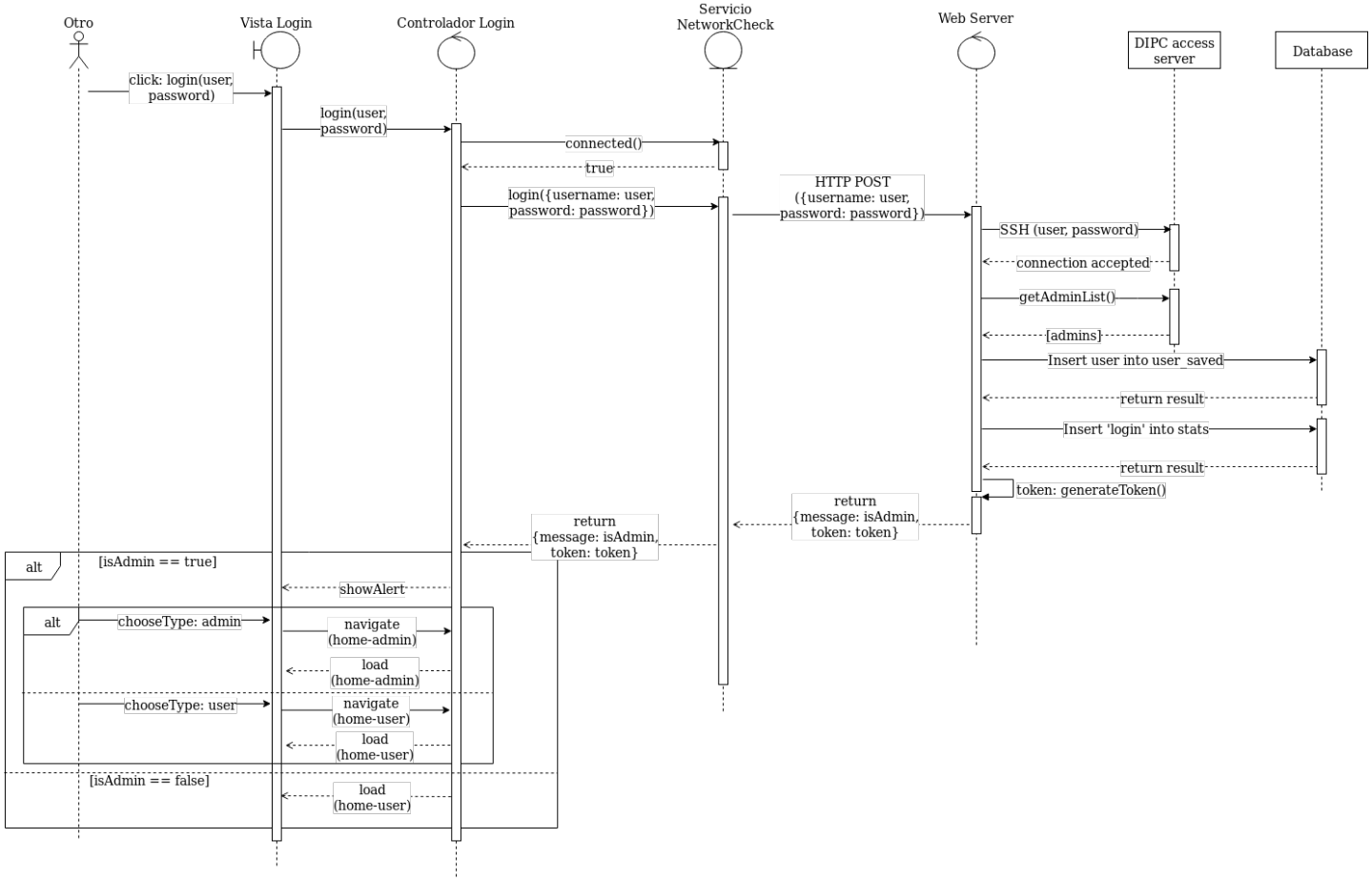

Figura 5.4: Diagrama de secuencia del inicio de sesión.

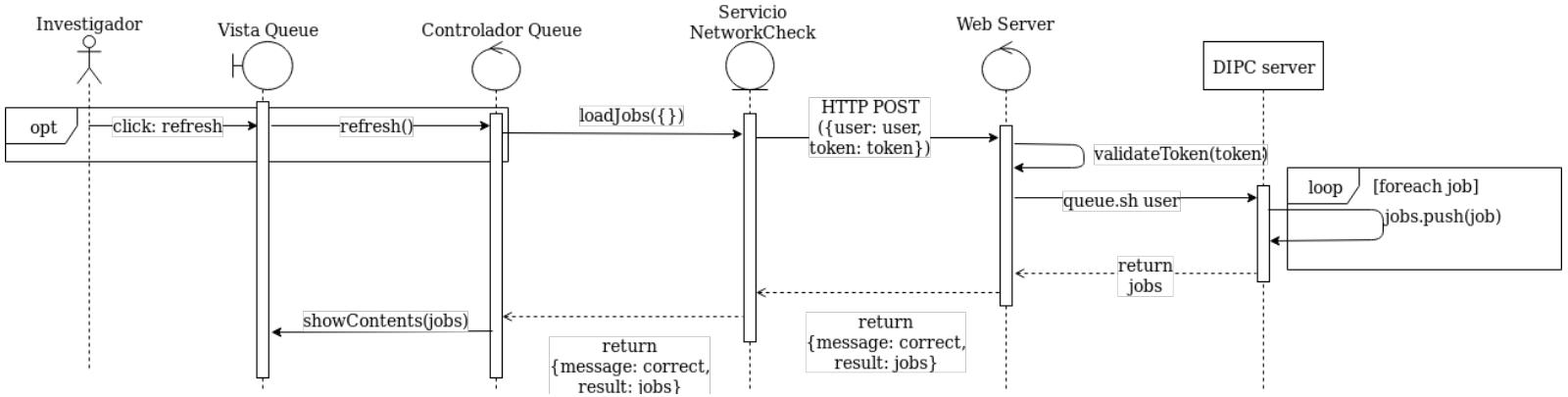

Figura 5.5: Diagrama de secuencia de la consulta de los trabajos en cola.

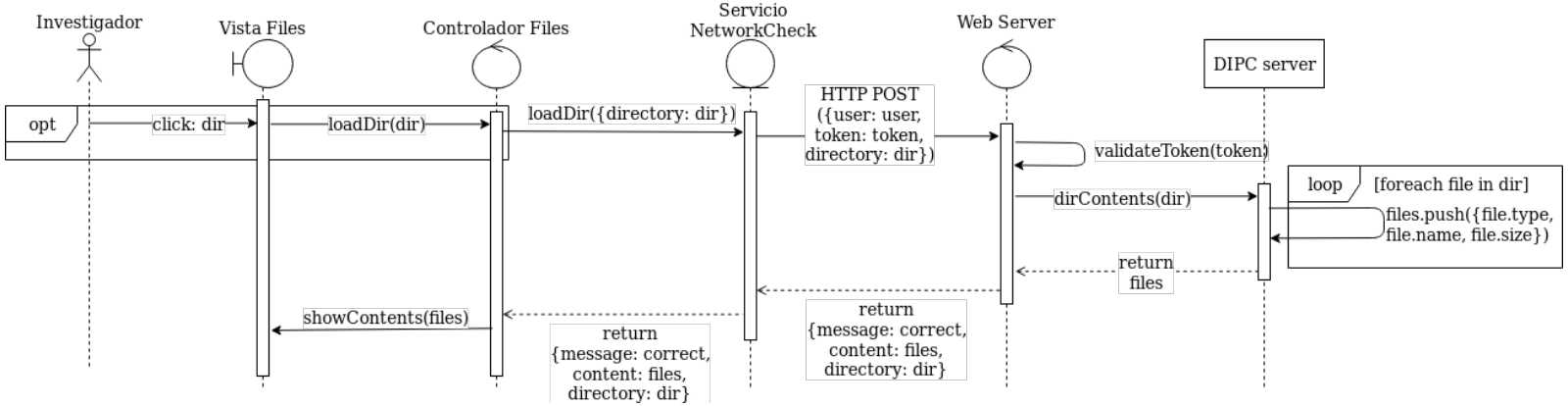

Figura 5.6: Diagrama de secuencia de la consulta de los directorios.

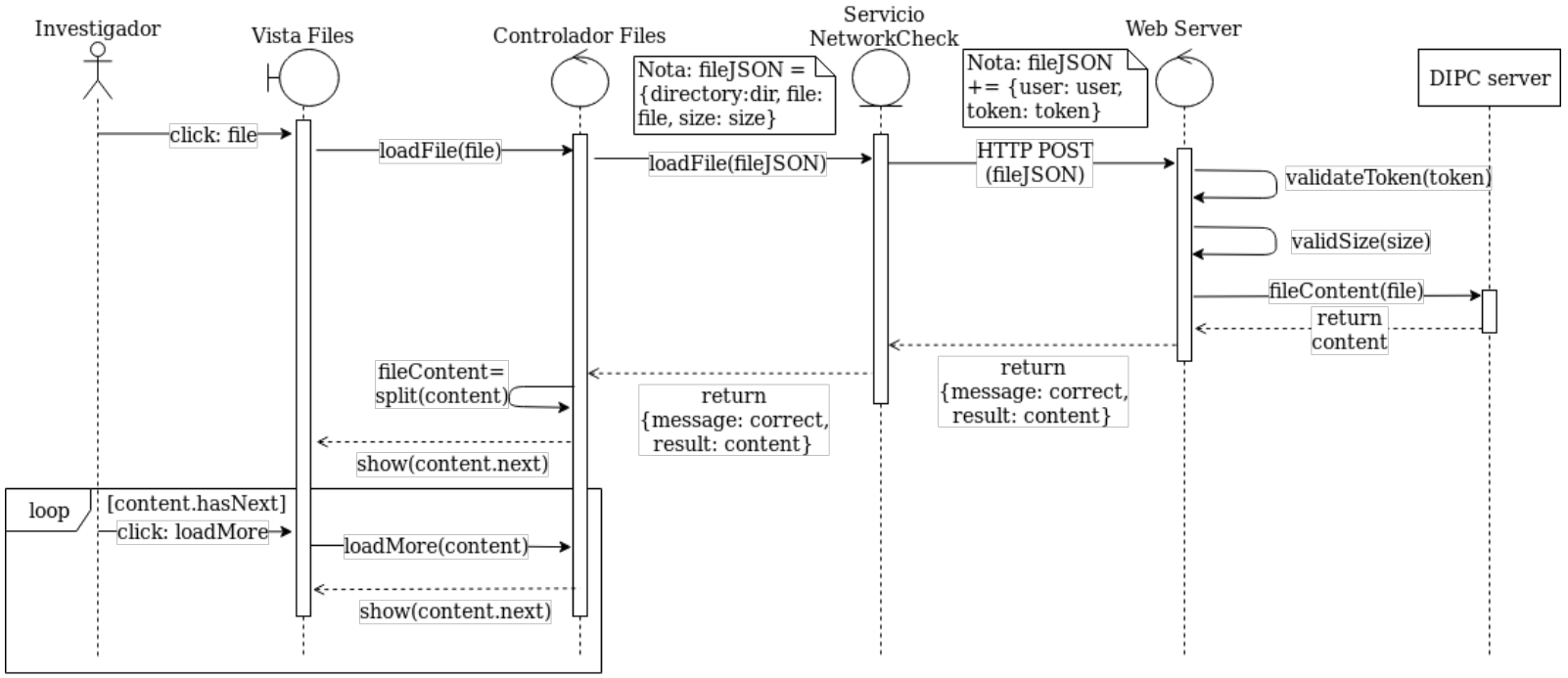

Figura 5.7: Diagrama de secuencia para ver el contenido de un fichero.

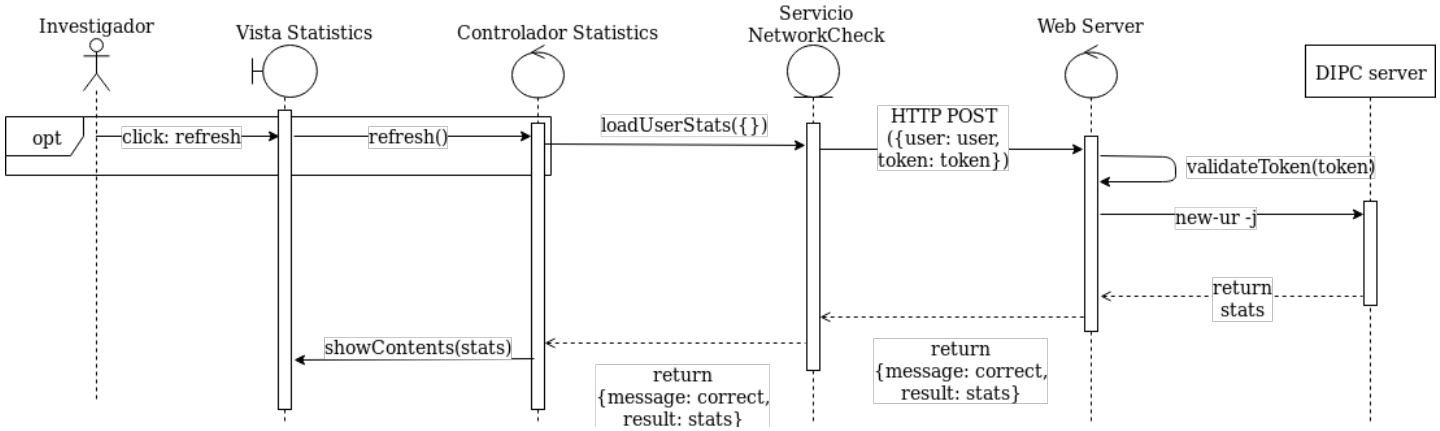

Figura 5.8: Diagrama de secuencia para consultar estadísticas personales.

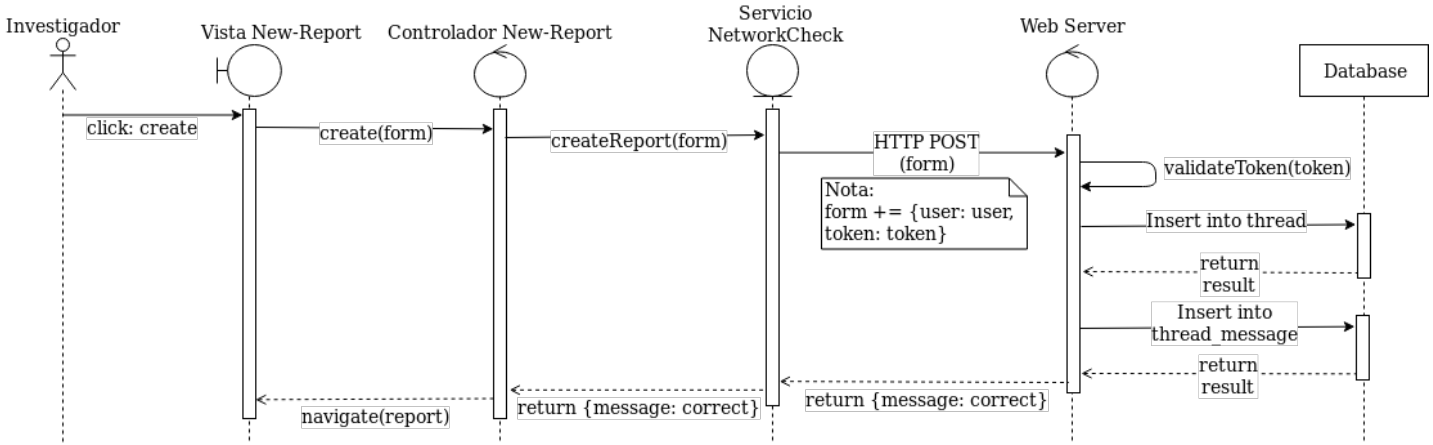

Figura 5.9: Diagrama de secuencia para crear un hilo.

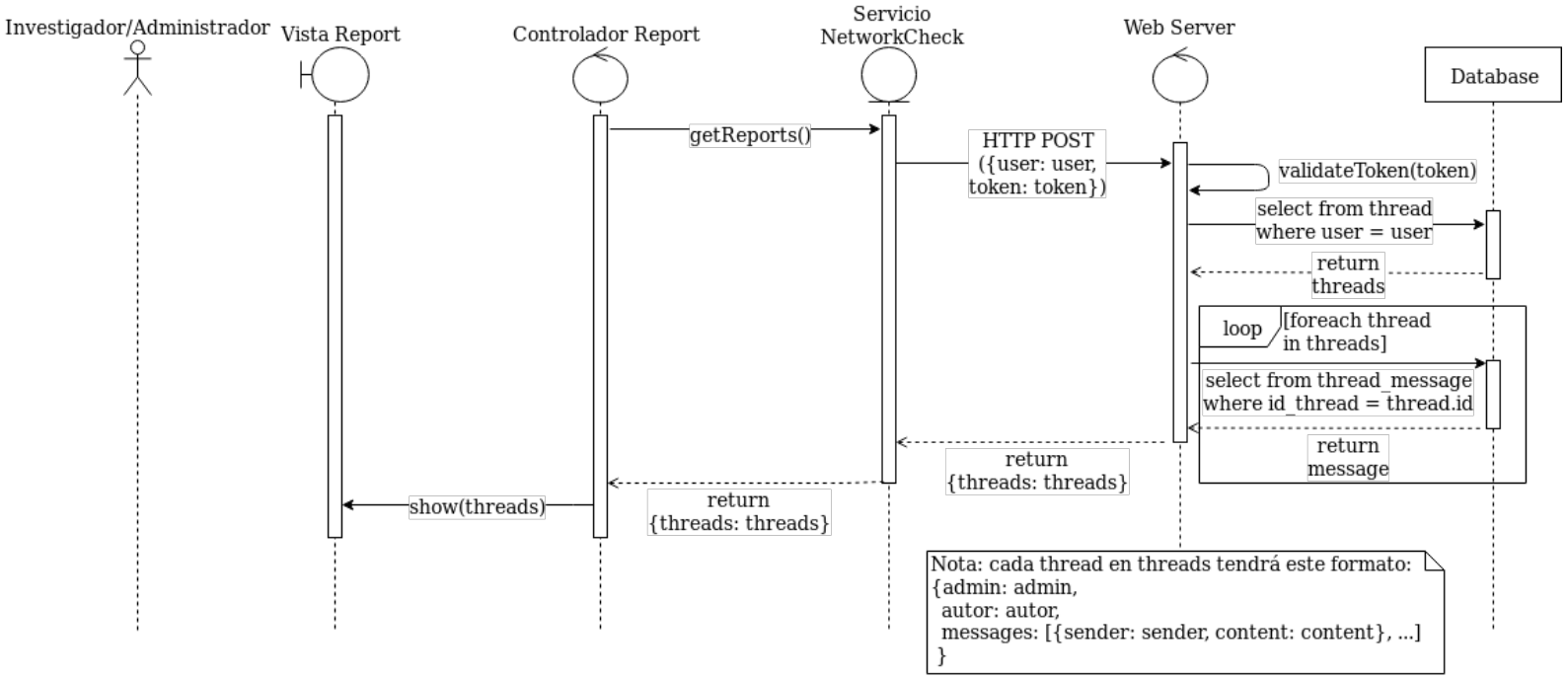

Figura 5.10: Diagrama de secuencia para ver los hilos del usuario.

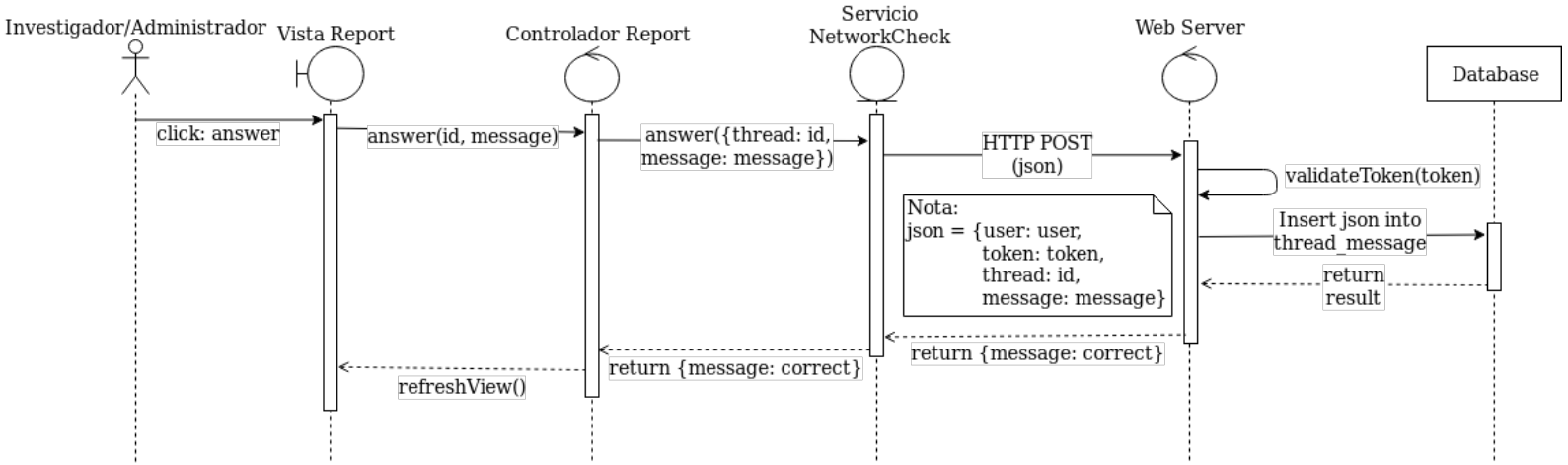

Figura 5.11: Diagrama de secuencia para responder a un hilo.

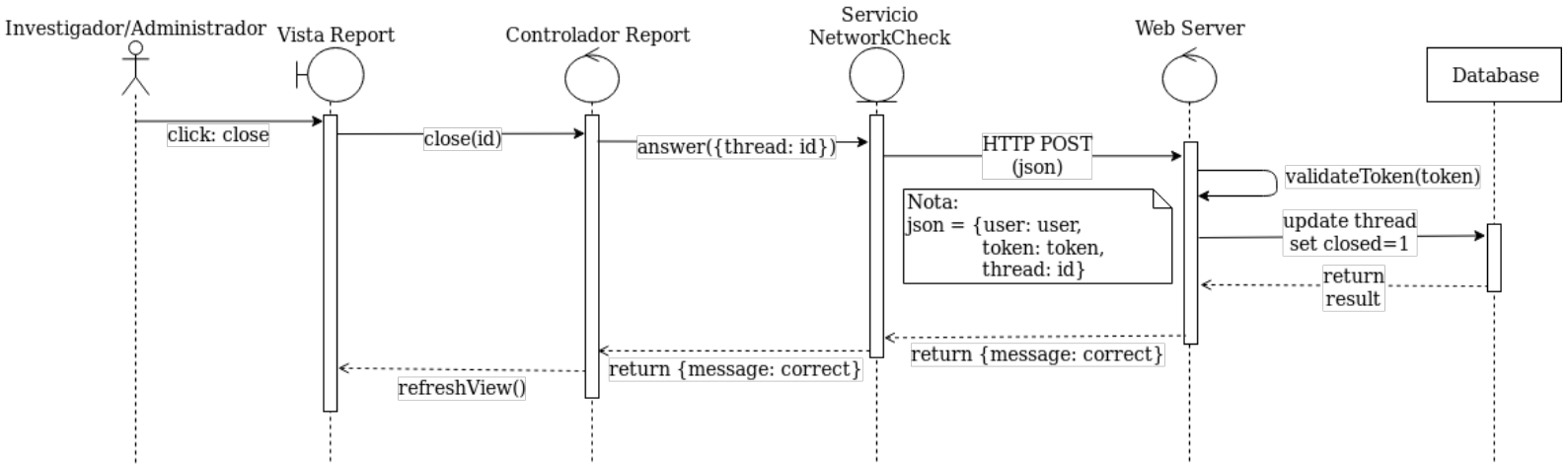

Figura 5.12: Diagrama de secuencia para cerrar un hilo.

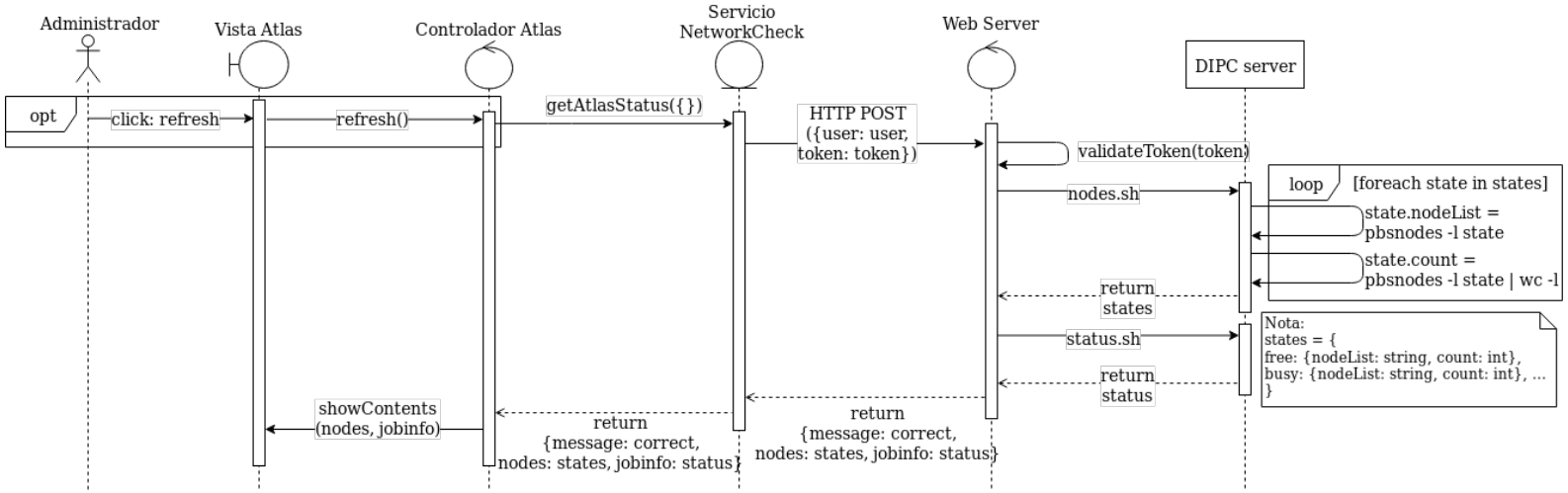

Figura 5.13: Diagrama de secuencia para comprobar el estado de ATLAS.

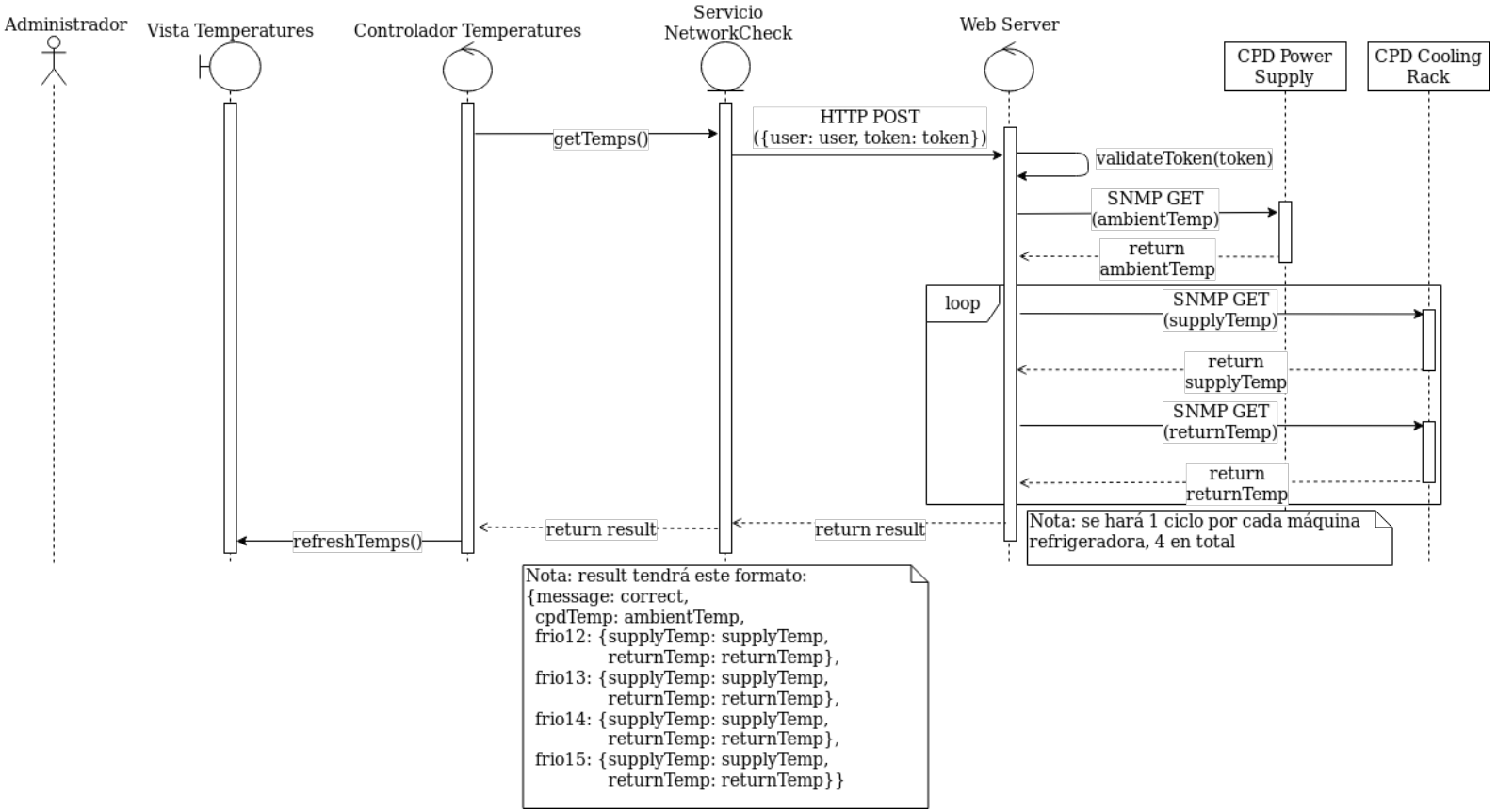

Figura 5.14: Diagrama de secuencia para comprobar las temperaturas.

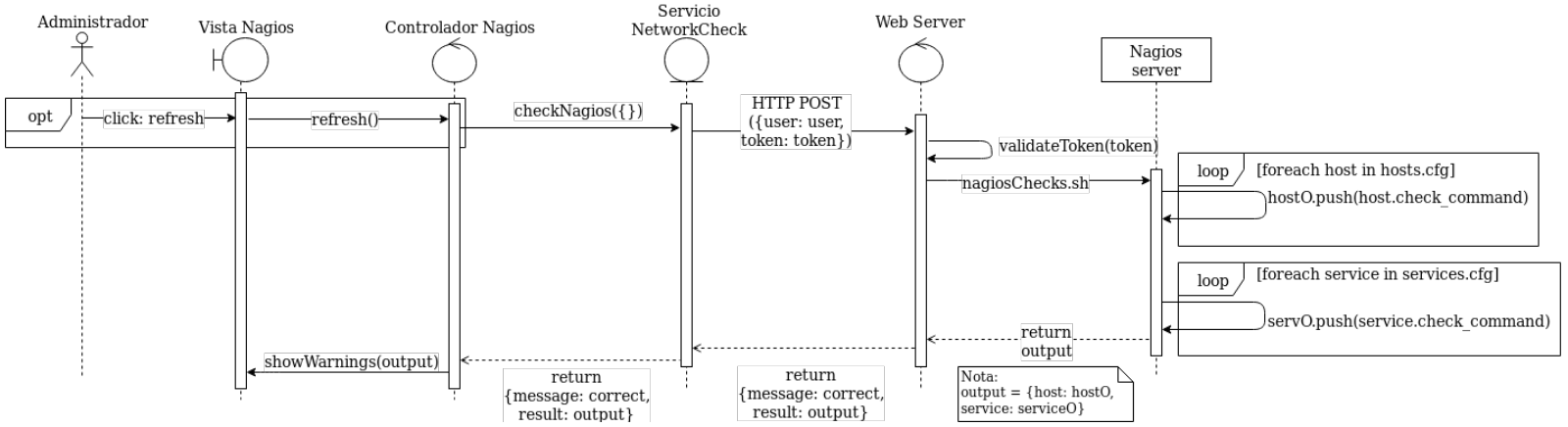

Figura 5.15: Diagrama de secuencia para comprobar el estado de la micro-informática.

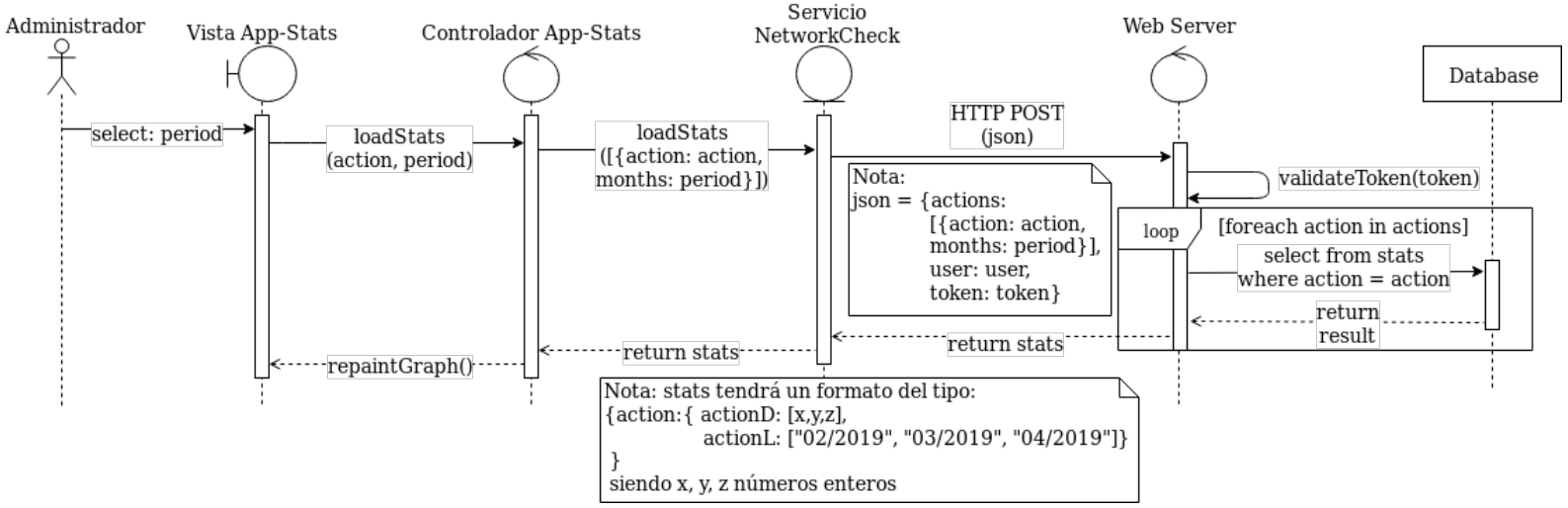

Figura 5.16: Diagrama de secuencia para consultar estadísticas de la aplicación.

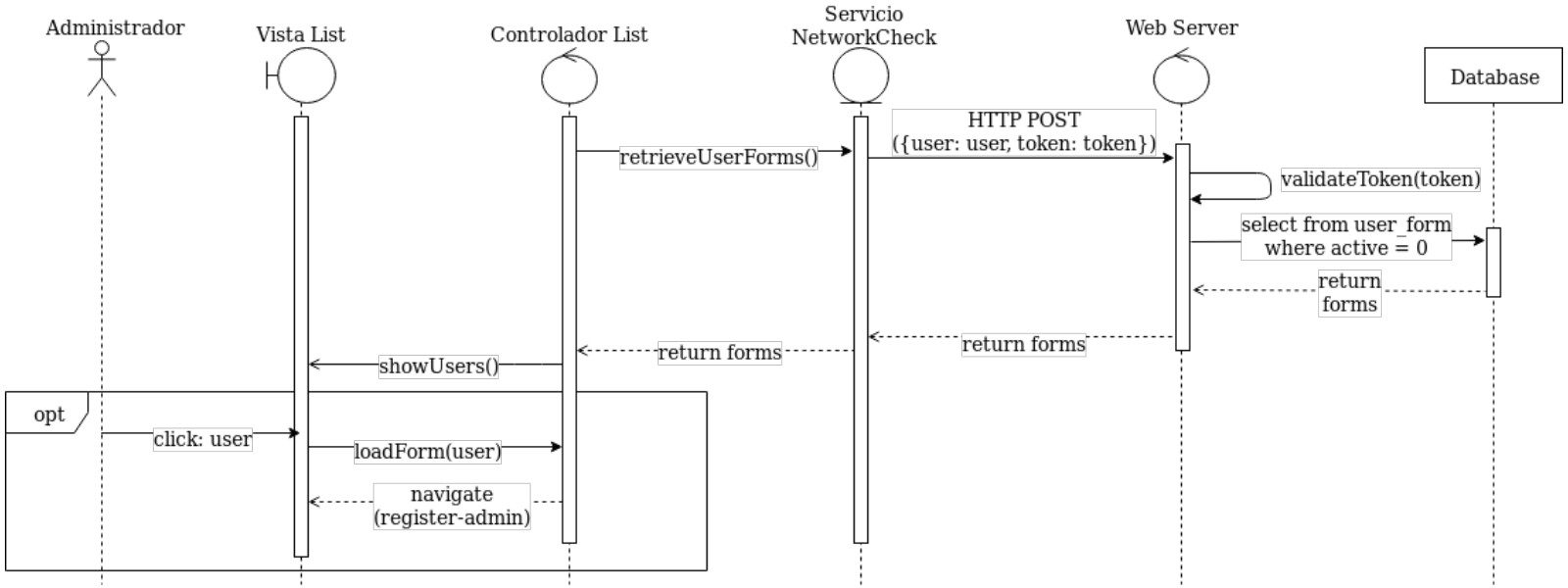

Figura 5.17: Diagrama de secuencia para ver solicitudes de creación de cuentas.

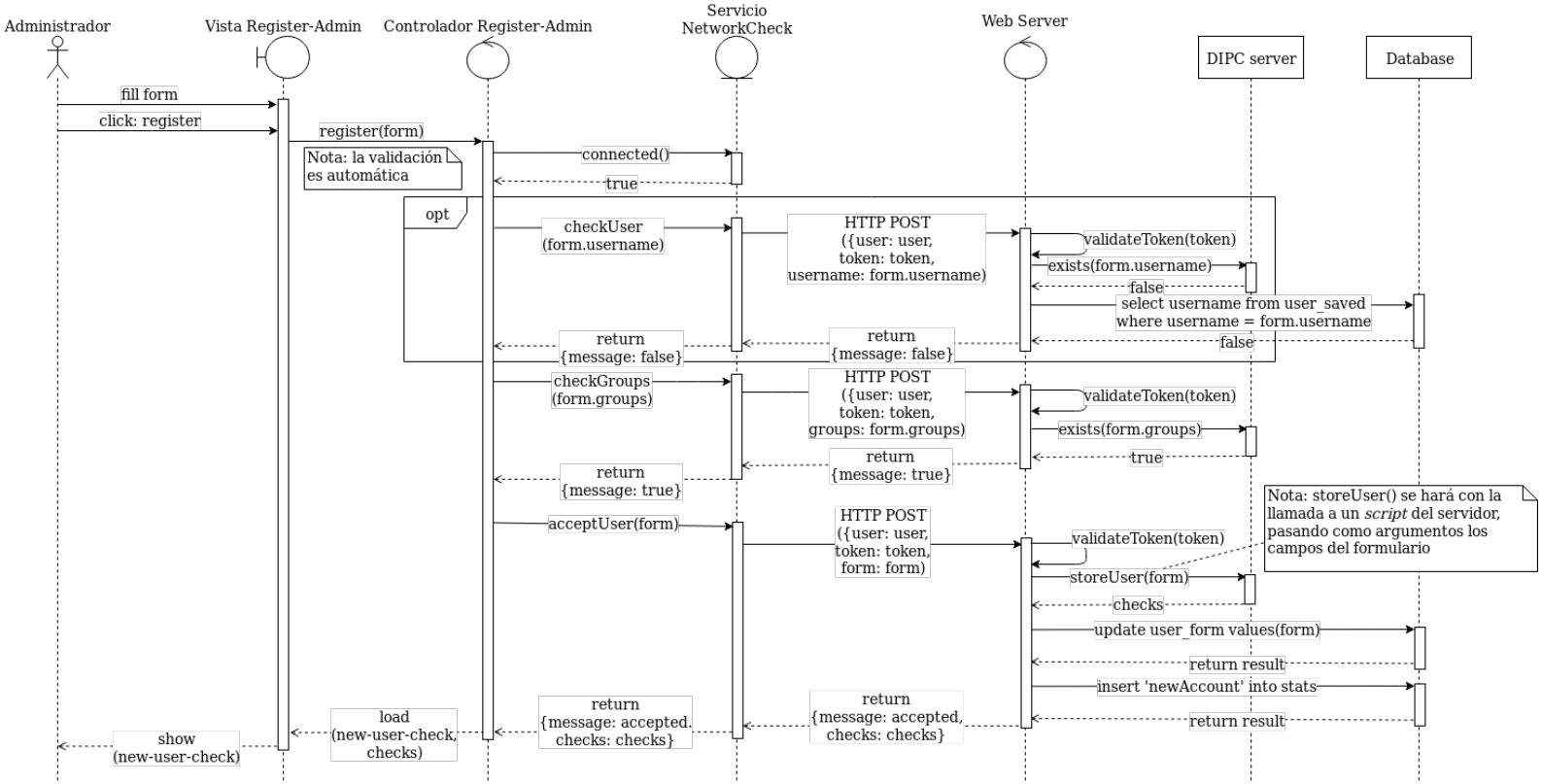

Figura 5.18: Diagrama de secuencia para crear cuentas.

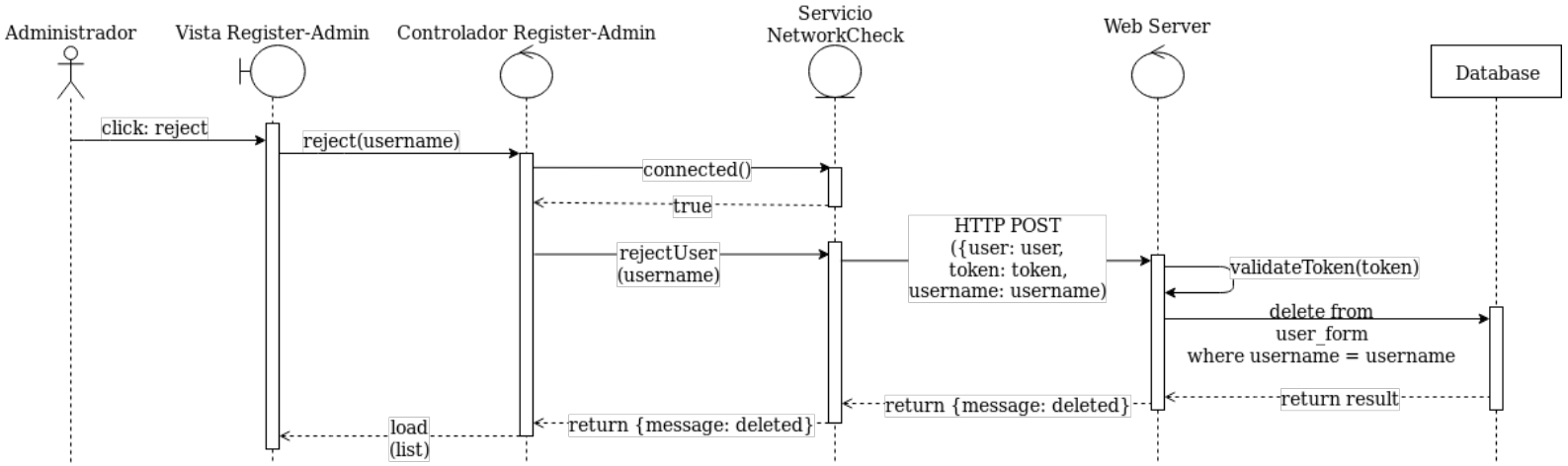

Figura 5.19: Diagrama de secuencia para denegar una solicitud de cuenta.

6. CAPÍTULO

Desarrollo del proyecto

En esta fase nos encargaremos de producir un código que concuerde con lo establecido en la fase de Diseño y responda a los requisitos identificados en el Análisis. Este código se escribirá en el lenguaje de programación más acorde a las necesidades del producto y deberá contener la menor cantidad de errores posibles, generando así un programa satisfactorio. También es buena práctica que el código esté lo más limpio y ordenado posible, para que si en un futuro hay que retomar el proyecto, el tiempo necesario para entenderlo y acomodarse al mismo sea el menor posible.

En este capítulo se hablará sobre la estructura de archivos del proyecto, así como del papel de cada uno de ellos. También se presentará un esquema de *routing* de la aplicación que muestre las diferentes páginas disponibles y las conecte con los casos de uso. Por último, se hablará sobre la implementación de cada caso de uso, explicando las peculiaridades que se encuentren, mostrando el diseño de la interfaz gráfica de cada uno de ellos, y siguiendo el flujo de eventos del diagrama de secuencia a nivel de código.

6.1. Estructura de archivos

La estructura de archivos del proyecto que se ha seguido es la estructura que utiliza por defecto Ionic, que a su vez está basada en la estructura de Angular pero con algunas modificaciones (como el directorio *platforms*), normalmente relacionadas con la integración de Cordova al proyecto.

Esta estructura se genera al crear un nuevo proyecto con Ionic CLI, y la mayoría de elementos los gestiona también este programa, permitiendo al desarrollador centrarse en el código y olvidarse de problemas como dependencias entre módulos, instalación de *plugins*, etc. En la figura 6.1 se puede ver la estructura de la raíz del proyecto.

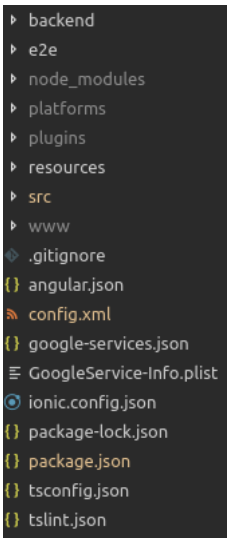

Figura 6.1: Estructura de archivos de la raíz.

El directorio ***backend*** se ha generado manualmente y contiene los archivos más importantes del *back-end* del proyecto, como el código de la API o los *script*s que se utilizan en el servidor. Este directorio está presente pero no juega ningún papel aquí, solo está para poder subirlo a un repositorio Git. El directorio ***e2e*** está generado por Angular y contiene las pruebas unitarias que utilicemos para comprobar el correcto funcionamiento de la aplicación. El directorio ***node_modules*** también está generado por Angular, y contiene todos los módulos que hayamos instalado en el proyecto. El directorio ***platforms*** contendrá los proyectos nativos de las plataformas que agreguemos al proyecto de Ionic. Está generado por Cordova y se modifica automáticamente al usar los comandos de Ionic CLI, por lo que no es necesario conocer sus contenidos. También está generado por Cordova el directorio ***plugins***, que es similar a *node_modules* de Angular pero con los *plugins* que necesita la aplicación para hacer uso de las funciones de Cordova. ***Resources*** es un directorio que Cordova usará para generar ciertos contenidos de la aplicación, como el icono o la pantalla de carga (*splash screen*). ***Src*** es el directorio que más se usa y contiene el código fuente de la aplicación. Se hablará de él más adelante. Por último, ***www*** es el directorio donde Angular genera el código web compilado (este código es el que después usa Cordova para generar el código nativo en *platforms*).

Respecto a los archivos de la raíz del proyecto, todos ellos son de configuración y solo se van a comentar los más importantes: *config.xml* y *package.json*.

- ***Config.xml*** es un fichero creado automáticamente por Cordova que contiene información vital para la generación de la aplicación en formato nativo, como la ruta hacia los recursos o los *plugins* de Cordova que se están utilizando. La mayoría de su contenido se gestiona automáticamente, pero se recomienda modificarlo nada más empezar el proyecto y rellenar campos como el nombre de la aplicación, el id o la versión actual (la versión habrá que modificarla también cada vez que se quiera publicar una actualización).

- ***Package.json*** es un fichero generado por Angular que contiene información sobre las dependencias del proyecto. No es necesario conocer el contenido, aunque sí que se recomienda modificarlo la primera vez para introducir los datos necesarios (nombre, autor, versión, ...). También es importante entender que es el fichero que utiliza *npm* para administrar los paquetes del proyecto (y las versiones de estos).

Cuando se ha hablado de los directorios se ha evitado entrar en detalle con el directorio *src*. Este directorio no contiene ningún elemento que haya habido que desarrollar (todo se auto-genera al crear el proyecto) y no es necesario conocer el papel de cada archivo. Exceptuando uno: su subdirectorio *app*.

Sobre el directorio ***app*** contenido en *src* hay que explicar varias cosas. Este es el directorio que contiene todos los componentes, servicios, páginas, etc, del proyecto y es, en consecuencia, el directorio donde más se trabaja. Todos sus elementos están ubicados bajo el directorio *app* excepto las páginas, que cada una tiene su subdirectorio propio donde están contenidos los ficheros que la forman. En la figura 6.2 se puede ver a la izquierda la lista de páginas (con la primera abierta para ver sus ficheros) y a la derecha el listado del resto de componentes.

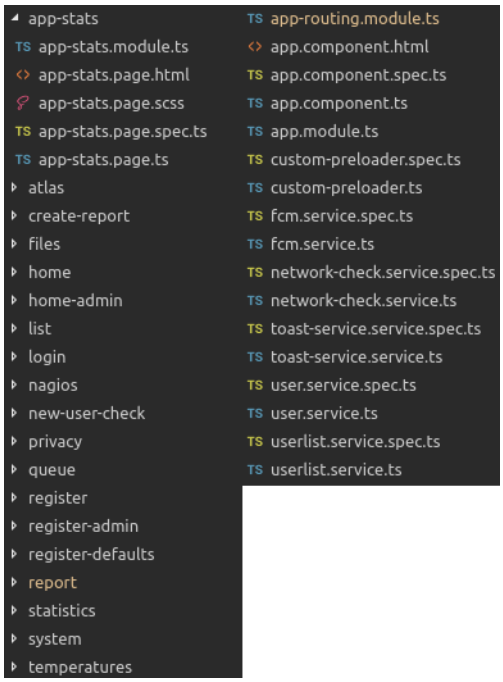

Figura 6.2: Estructura de archivos del directorio src/app.

Respecto a las páginas, cada una está compuesta (por defecto) de cinco ficheros: el **módulo**, que se encarga de ensamblar la página y sus dependencias cuando es hora de cargarla; la **plantilla**, que es un fichero en lenguaje HTML (aunque acepta directivas propias de Angular y definidas por el usuario) que da forma a la interfaz de la página; la **hoja de estilos**, que permite modificar la plantilla original y aplicar estilos diferentes al del resto de la aplicación, la **lógica** de la página, que asocia la plantilla con la hoja de estilos, exporta el componente e implementa el funcionamiento de la página; y el **módulo de pruebas**, que define los test unitarios de la página. Si nos fijamos en los ficheros del directorio *src/app*, podremos ver que hay cuatro ficheros que concuerdan con la descripción de una página que acabamos de explicar (todos los que empiezan por 'app.'), y es que estos cuatro ficheros forman una página más, la página raíz de la aplicación, que estará siempre presente en todo el resto de páginas (debido al funcionamiento de la jerarquía de vista de Angular) y encapsula el resto de componentes y servicios de la aplicación.

El resto de ficheros que se pueden ver en la figura 6.2 son componentes y servicios que han sido definidos para implementar diversas funcionalidades de la aplicación, exceptuando uno: **app-routing.module.ts**. Este fichero es el que define todas las rutas de la aplicación (que se van a explicar en la sección 'Esquema de routing') y a su vez enlaza una ruta con un módulo de una página, por lo que al acceder a una ruta se carga el módulo enlazado que monta la página que se debe mostrar. Además de esto, en este fichero también se

especifica el tipo de carga que se quiere aplicar para cada página. Existen dos tipos: la tradicional y la carga perezosa (*lazy loading*).

- La **carga tradicional** es una carga donde todas las páginas de la aplicación se cargan a la vez nada más se abra la aplicación. Este tipo de carga supone un tiempo de lanzamiento de la aplicación alto, una uso de memoria alto (se cargan todas las páginas siempre, independientemente de si se acaba usando o no) pero un tiempo de acceso a cada página rápido, al estar ya ensambladas.
- La **carga perezosa** implica cargar cada página solo la primera vez que se acceda a ella. Es un tipo de carga más innovador que optimiza el uso de memoria (ahora solo se cargarán las páginas que se vayan a usar) pero la experiencia de usuario puede verse afectada, si alguna de ellas tarda mucho en montarse.

En nuestra aplicación una carga tradicional no tiene mucho sentido, ya que la aplicación está claramente dividida en dos partes: zona investigador y zona administrador (excluimos la zona otro que solo son tres páginas). Hacer que un investigador tenga que cargar la zona de administrador entera sabiendo que solo va a acabar usando la zona del investigador no es la mejor solución. Sin embargo, al aplicar la carga perezosa se ha observado que ciertas páginas tienen un tiempo de carga muy alto, independientemente de las especificaciones técnicas del dispositivo, que perjudican la experiencia de uso del usuario. Estas páginas son la página de registro y la página de aceptar formularios del administrador (*register* y *register-admin*).

La solución que se ha aplicado ha sido implementar una estrategia de carga personalizada, donde se decide qué páginas se van a cargar al lanzarse la aplicación (carga tradicional) y qué páginas se van a cargar al abrirse por primera vez (carga perezosa). Esta estrategia está definida en el fichero *src/app/custom-preloader.ts* y es utilizada en *src/app/app-routing.module.ts* (las rutas que contengan la propiedad '*data: {preload: true}*' utilizarán la carga tradicional). En concreto, las páginas que utilizan la carga tradicional son las dos de registro que se han mencionado antes junto con la página de *login*, por ser la página inicial de la aplicación.

Los servicios que se ven en la figura 6.2 han sido creados con un propósito concreto en la aplicación. Los servicios de Angular son clases encargadas de acceder a los datos de la aplicación y enviárselos a diferentes componentes. De esta manera, dos componentes que no se conocen entre sí son capaces de comunicarse y compartir información. Nuestros servicios tienen juegan los siguientes papeles:

- ***fcm.service***: Servicio utilizado para integrar las notificaciones *push*. En un principio la idea era que generase el identificador del dispositivo necesario para utilizar las notificaciones, además de enviaro al servidor web. Al final, ha acabado por no utilizarse porque no se ha terminado de integrar el sistema de notificaciones en el proyecto.

- ***network-check.service***: Servicio encargado de realizar todas las comunicaciones necesarias con el servidor web. Este servidor guarda la información necesaria para conectarse al servidor (como su dirección IP, el protocolo de conexión o la cabecera HTTP). También contiene el conjunto de funciones que forman las llamadas a la API web, así como una función que comprueba el estado de la conexión a Internet del dispositivo.

- ***toast-service.service***: Servicio que se utiliza para mostrar las notificaciones *toast* dentro de la aplicación. No es del todo necesario, ya que se puede llamar a un controlador desde cada página, pero se ha implementado con el fin de reutilizar código y hacerlo más legible. Con él, cualquier página que quiera mostrar una notificación *toast* solo tendrá que inyectar el servicio y llamar a su función *showToast* con el mensaje a mostrar y la duración como parámetros.

- ***user.service***: Este servicio almacena la información del usuario conectado, para que sea accesible desde cualquier componente que la necesite. Esta información está compuesta por el nombre de usuario, un *boolean* que indica si el usuario es administrador o no, y el token JWT recibido del servidor web para que pueda ser utilizado en la próxima llamada a la API.

- ***userlist.service***: Este servicio se utiliza para compartir información entre dos páginas en concreto: *list* y *register-admin*. La idea es que, al cargarse *list*, se obtienen de la API todos los formularios de registro que estén pendientes y se listan para que el administrador pueda verlos y decidir cuál abrir. El problema es que al abrir uno, habría que compartir entre las páginas todos los campos de ese formulario para que se puedan rellenar en *register-admin*, y la única forma de hacerlo sin un servicio es mediante parámetros en la ruta (poco práctico). Mediante el servicio, se almacenan todos los valores de los formularios y se da acceso a esta información. Así, al entrar a un formulario desde *list*, solo habrá que enviar el identificador del formulario, y el servicio cargará el resto de campos utilizando este identificador. Más adelante también se ha utilizado para compartir la lista de administradores entre las páginas *report* y *create-report*, evitando así llamadas innecesarias a la API.

6.2. Esquema de routing

Nada más se inicie la aplicación y terminen de cargarse todos los módulos necesarios, se estará ubicado en la página *login*. Está página se considera la página raíz o *root* de la aplicación y actúa como padre del resto. Desde esta página un usuario tendrá tres opciones: acceder a la página *privacy*, donde se le mostrará la política de privacidad del centro; acceder a la página *register*, para rellenar un formulario de nueva cuenta; o iniciar sesión. Para las dos primeras opciones se provee al usuario de un botón atrás (*back-button*) para poder navegar de nuevo a la página raíz.

Si se decide iniciar sesión, se pueden seguir dos posibles rutas, dependiendo del tipo de cuenta con el que se inicie sesión y la decisión del usuario: acceder a la zona de investigadores *home* o a la zona de administración *home-admin*. Si nos fijamos en la figura 6.3 podremos ver que aparece un elemento llamado *toolbar Component*. Este elemento hace referencia a una barra de herramientas ubicada en la parte superior de la pantalla que nos permitirá abrir un menú lateral para navegar entre las diferentes secciones de la zona en la que nos encontremos.

Que el elemento *toolbar Component* aparezca aquí indica que está barra de herramientas aparecerá en todas las páginas hijas que tenga. Esto es así por la manera que tiene Angular de trabajar con las rutas. Por ejemplo, si tuviésemos una página A que contiene un campo *input* y otra página B que contiene un botón y hacemos que la ruta sea */A/B*, la página resultante tendría el campo *input* de A y el botón de B. Este ejemplo no tiene un sentido práctico, pero la funcionalidad permite definir una vez el componente *toolbar* y utilizarlo en cualquier página que hagamos hija suya o dejar de utilizarlo en una página en concreto sin tener que cambiar más que una línea de código, que cambie la ruta de */A/B* a */B*, por ejemplo.

Si se accede a la zona de investigadores (morado en la figura 6.3), veremos enlaces a sus diferentes secciones. Desde cualquiera de ellas se puede navegar a cualquier otra del mismo nivel utilizando el menú de navegación lateral. La página *report* también permite navegar a la página *create-report*.

Sin embargo, si se accede a la zona de administración (verde y naranja en la figura 6.3), los enlaces de los que dispondremos tanto en la página *home-admin* como en el menú lateral serán diferentes, aunque funcionarán igual que en la zona de investigadores. Desde la página *list* veremos todos los formularios de cuentas pendientes y si accedemos a uno de ellos se cargará la página *register-admin*. Desde esta página se puede acceder a *new-*

user-check si el administrador decide crear la cuenta. Además, en la figura se puede ver que en esta zona existe un componente llamado *tabs Component*, que es padre de todas las páginas en naranja. Este elemento hace referencia a unas pestañas de navegación, que estarán ubicadas en la zona inferior de la pantalla. Utilizando las pestañas, se puede navegar entre las 3 páginas hijas de *tabs Component* (naranjas en la figura 6.3), además de poder usar también el menú lateral para acceder a otra página de la zona.

Por lo tanto, para cargar la página *temperatures* existen 4 formas posibles:

Ruta	**Resultado**
/toolbar/tabs/temperatures	Se cargará el contenido de *temperatures* junto con la barra de herramientas, el menú lateral y las pestañas. Esta es la forma en la que se va a usar.
/toolbar/temperatures	Se cargará el contenido de *temperatures* junto con la barra de herramientas y el menú lateral.
/tabs/temperatures	Se cargará el contenido de *temperatures* junto con las pestañas.
/temperatures	Solo se cargará el contenido de *temperatures*.

Tabla 6.1: Ejemplo del funcionamiento de las rutas en Angular.

Por último, en el menú lateral se colocará un botón llamado *logout* que permitirá a cualquiera que haga clic en él acceda a la página *login*. De la manera en que está estructurada la navegación, el acceso a este botón solo se tendrá después de haber iniciado sesión.

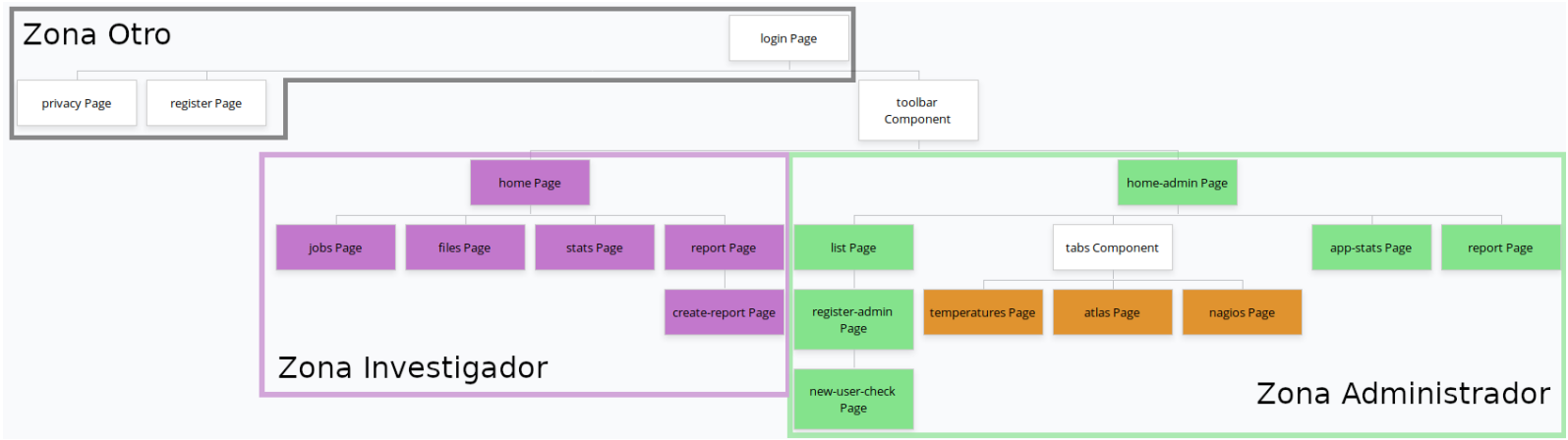

Figura 6.3: Esquema de routing de la aplicación.

6.3. Implementación de los casos de uso

A continuación se van a explicar los puntos clave de la implementación de cada caso de uso. Como se puede ver en los diagramas de secuencia del capítulo 5, muchos de ellos se comportan de forma parecida, por lo que la explicación se va a centrar en los casos de uso que se consideran más característicos o los que utilizan alguna tecnología única, con el fin de englobar todos los puntos más importantes de la aplicación en el menor número de casos de uso posibles.

Cada caso de uso se estructurará de la siguiente manera:

1. Páginas del caso de uso: Se hablará sobre el papel que juega cada página dentro del caso de uso. Se mostrará el diseño de la interfaz de usuario y se explicarán los aspectos más relevantes de la lógica de la página.
2. Servicios: Se explicará la función de todos los servicios de la aplicación que tomen parte en el caso de uso. Este punto junto con el anterior formarían el *front-end* del sistema (la aplicación).
3. *Back-end*: Explicación sobre los puntos clave por los que pasa la API al recibir la petición desde la aplicación.

6.3.1. Inicio de sesión

El inicio de sesión es uno de los casos de uso más importantes de cualquier sistema que tenga uno. Además, en concreto en nuestro caso se hace uso de cantidad de tecnologías, como conexiones SSH, accesos a la base de datos y generación de objetos JWT.

Páginas del caso de uso

Solo hay una página presente en el inicio sesión, la página *login*. Esta página tiene la forma de un inicio de sesión convencional, con un campo para el nombre de usuario, otro para la contraseña y un botón para iniciar sesión. Al ser la página raíz de la aplicación se ha añadido también una imagen del logotipo del centro, para mejorar la presentación.

Hay que mencionar que todos los elementos que se ven en la figura 6.4 están ubicados bajo una directiva **ngIf* de Angular, que solo añadirá los elementos al DOM de la página

si se cumple la condición. Esto se ha hecho para que cuando se pulse el botón para iniciar sesión, mientras se espera la respuesta de la API, no se muestren ninguno de los elementos y, en cambio, se muestre una barra de progreso que indica que está iniciándose la sesión.

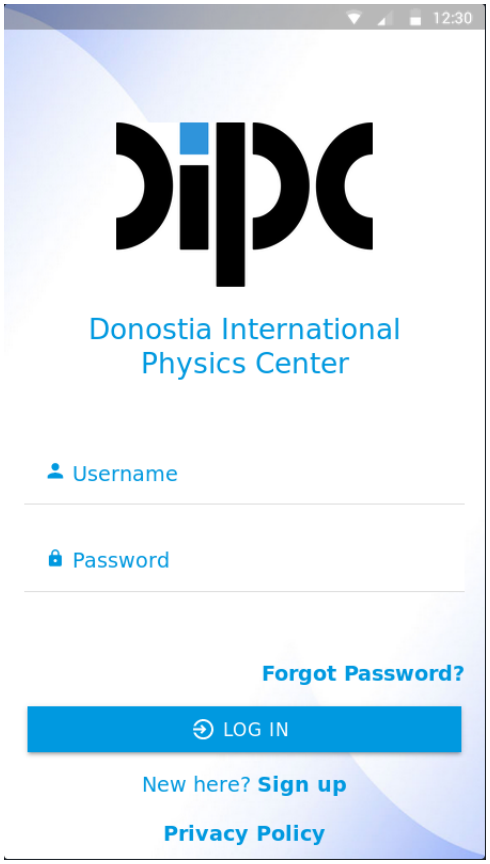

Figura 6.4: Interfaz de usuario de la página *login*.

Para contactar con la API, la aplicación utiliza una clase llamada *Observable* de Angular. Esta clase proporciona nuevas funcionalidades al paso de mensajes entre un *publisher* y un *subscriber*. Creamos un *Observable* al realizar la petición HTTP a la API, que se hace en segundo plano (es una llamada asíncrona) y permite seguir ejecutando código y utilizando la aplicación. Al suscribirse a la llamada se envían por parámetros dos funciones. La primera de ellas se ejecuta al recibir un valor de la suscripción y ese valor se recibe por parámetro (su tipo es `(value: Object) =>void`). Como la API devuelve objetos JSON, recibir un valor implica que se ha obtenido un valor correcto de la API (aunque puede ser un mensaje de error, el código de respuesta sigue siendo '200 OK'). Habrá que convertir la respuesta JSON en un objeto de TypeScript y analizar el contenido de la propiedad 'message' para conocer el verdadero estado de la respuesta. El segundo parámetro de la suscripción es otra función (con tipo `(error: any) =>void`) que se ejecutará si el

resultado de la petición HTTP es un mensaje de error (como ’404 - Not Found’ o ’500 - Internal Server Error’). En este caso se podría tratar de diferente manera dependiendo del código de error, pero no se han diferenciado y se notifica al usuario siempre con el mismo mensaje.

```
const loginSub = this.networkCheck.login({ username: this.username, password: this.pass })
  .subscribe(data => {…
  }, error => {
    loginSub.unsubscribe();
    console.log(error);
    this.loading = false;
    this.toastService.showToast('Internal server error.', 3000);
  });
setTimeout(() => {
  loginSub.unsubscribe();
  if (this.loading) {
    this.loading = false;
    this.toastService.showToast('Timeout reached. Try again later.', 3000);
  }
}, 10000);
```

Figura 6.5: Suscripción de la página *login*.

Al ser una función asíncrona, el trozo de código que va después de la suscripción al *Observable* se ejecuta inmediatamente. Es otra llamada a otra función asíncrona, llamada *setTimeout*, que programa la ejecución de una sub-función (que recibe como primer parámetro) dentro de un tiempo determinado (segundo parámetro). La sub-función está programada para que se ejecute a los 10 segundos de empezar la suscripción al *Observable*, y su único papel es cancelar la suscripción en caso de que siga activa y, si la página aún está cargando (no se ha recibido respuesta alguna de la API, ni exitosa ni de error), parar la carga y mostrar un mensaje al usuario, indicando que se ha excedido el tiempo límite de espera. En la figura 6.5 se puede apreciar mejor lo que se ha explicado (el código del primer parámetro de la suscripción se ha ocultado por su longitud y porque el objetivo de la figura es mostrar un ejemplo de cómo funcionan las suscripciones y la programación asíncrona de Angular).

Servicios

En esta página toman parte tres servicios: *user.service*, *toast.service* y *network-check.service*.

- *user.service*: Sirve para almacenar la información del usuario que haya iniciado sesión. Como si es administrador o no, el nombre de usuario o el token.
- *toast-service.service*: Se utiliza para mostrar mensajes al usuario, sobre todo si algo

no ha funcionado como se esperaba (errores de conexión, credenciales incorrectos, ...).

- *network-check.service*: Envía la petición HTTP a la API. También genera el objeto JSON en el que se encapsula toda la información que se quiere enviar.

Back-end

Lo primero que se hace en la API es leer la petición y ejecutar la función correspondiente. Dentro de la función, se comienza leyendo `php://input`, que es de donde se obtiene el cuerpo de la petición, y decodificando el objeto JSON leído en un *array* asociativo[1] que utilizaremos para acceder a los campos del cuerpo de la petición.

El siguiente paso es conectarnos a los *cluster*s del centro utilizando los credenciales que se hayan provisto en la petición para comprobar que son válidos. Además, aprovechando que estamos conectados, extraemos un *string* como los nombres de usuario de todos los administradores del sistema (los que pertenecen al grupo 'cc') separados por un espacio, que usaremos más tarde para determinar si el usuario intentando conectarse es administrador o no. En la figura 6.6 se puede ver mejor el proceso de leer el cuerpo de la petición, conectarse mediante SSH y ejecutar comandos (este proceso se va a realizar en varios casos de uso).

```
$input_data = json_decode(file_get_contents('php://input'), true);
if (is_null($input_data)){
        echo json_encode(array('message' => 'Bad Request'));
        die();
}
// Connect with given credentials to see if they're valid
$ssh = $this->_sshConnection($input_data['username'], $input_data['password'], 'atlas.sw.ehu.es');
if (!$ssh){
        echo json_encode(array('message' => 'SSH Fail'));
        die();
}
// As we are connected, we take advantage of it and retrieve all the admin names
// (usernames from groups with gid 1700 a.k.a. cc)
$r = $ssh->exec("ldapsearch -x -LLL -b ou=groups,dc=sw,dc=ehu,dc=es \"(gidnumber=1700)\"
                | grep memberUid: | sed -n -e 's/^.*memberUid: //p' | tr '\r\n' ' '");
$ssh->disconnect();
unset($ssh);
```

Figura 6.6: Fragmento de código de la API para el inicio de sesión.

Se acaba generando el *token* que dará acceso más tarde a diferentes funciones de la API, utilizando el nombre de usuario, la fecha actual del servidor y un campo que determina si

[1] Según Wikipedia: "Tipo abstracto de dato formado por una colección de claves únicas y una colección de valores, con una asociación uno a uno."

el usuario tiene privilegios de administrador, y se crea la respuesta HTTP que recibirá la aplicación. Por último, se almacena en la base de datos el inicio de sesión, que se utilizará en el caso de uso 'Consultar estadísticas de la aplicación'.

Si nos fijamos en la figura 6.6, se puede ver que la conexión SSH se hace mediante una llamada a una función privada de la API con nombre *_sshConnection*. Esta función se ha creado para utilizar la librería *phpseclib* y reintentar la conexión varias veces antes de decidir que no se puede realizar la conexión. Véase la figura 6.7.

```php
private function _sshConnection($user, $pass, $server){
        // PHP errors are hidden to avoid unnecessary output
        ini_set('display_errors', 0);
        // 3 retries, just in case
        for ($i = 0; $i < 3; $i++){
                $ssh = new Net_SSH2($server);
                if ($ssh->login($user, $pass)){
                        ini_set('display_errors', 1);
                        return $ssh;
                }
        }
        ini_set('display_errors', 1);
        return false;
}
```

Figura 6.7: Conexión SSH mediante *phpseclib*.

6.3.2. Consulta de los trabajos de cálculo

Este caso de uso se ha escogido para ejemplificar la ejecución de comandos mediante SSH de forma remota (aunque ya se ha explicado en el caso de uso anterior), haciendo uso de un *script* ubicado en el clúster y dándole formato a su salida para que pueda ser utilizada. También se pretende dar una demostración de la directiva **ngFor*. Además, se considera uno de los casos de uso más característicos, ya que la cola de trabajos tiene una importancia alta dentro del funcionamiento de los clústeres.

Páginas del caso de uso

Solo hay una página que forma parte del caso de uso, *queue*. Esta página contendrá una lista de 'cartas' o 'tarjetas' (*ion-card*), que son componentes que integra Ionic y permite encapsular contenidos en cajas de forma visual. En nuestro caso, estas cajas se van a hacer ampliables, es decir, en su estado inicial mostrarán ciertos datos representativos del trabajo y cuando se decida expandirlas mostrarán todo el resto de datos relacionados

con ese trabajo. También hay que señalar que solo se va a permitir tener una 'tarjeta' expandida a la vez, y que al expandir una hará que la anterior expandida se colapse.

Para mostrar un número indeterminado de trabajos (un investigador puede llegar a tener cientos de trabajos activos al mismo tiempo) se va a utilizar la directiva **ngFor* de Angular. Esta directiva es muy potente y se puede comparar con un bucle *for* que recorra todos los elementos de un *array* y ejecute un código por cada elemento, pero en este caso en lenguaje HTML. En concreto permite hacer que por cada elemento del *array* que reciba como valor se añadan al DOM de la página los componentes que sean hijos del componente que tiene la directiva. Se puede apreciar mejor visualmente en la figura 6.8.

```
25  <ion-list *ngIf="!error && !showBar && jobs !== undefined && jobs.length != 0">
26    <div *ngFor="let job of jobs;">
27      <ion-card>
28        <ion-card-header (click)="toggleJob(job)">
29          <ion-item>
30            <ion-icon slot="start" [name]="isJobShown(job) ? 'remove' : 'add'"></ion-icon>
31             
32            <h5>{{job.jobid}}</h5>
33          </ion-item>
34          <ion-item>
35            <ion-label text-wrap color="medium"><strong>Name:</strong> <br />{{job.Job_Name }}</ion-label>
36            <ion-label slot="end" text-wrap color="medium"><strong>State:</strong> <br />{{job.job_state }}</ion-label>
37          </ion-item>
38        </ion-card-header>
```

Figura 6.8: Ejemplo de **ngFor*.

Se puede ver como en la línea 25 se añade una lista de elementos que solo se mostrará en caso de que se cumpla la condición (no hay error, no se está cargando, el *array* esta definido y tiene algún elemento) y debajo suyo se encuentra la directiva **ngFor* que añadirá por cada *job* del *array jobs* un *div* que contiene una 'tarjeta' con múltiples propiedades de *job*, como el *jobid* o un icono que cambia dependiendo de si *job* está expandido o no. El *div* se extiende más líneas que las que se ven en la figura pero con las que se muestran se ve suficientemente bien el funcionamiento de **ngFor*.

El trabajo que está expandido se decide en base al valor de la variable *shownJob*, que contendrá un *job* de *jobs* o estará sin definir. El valor de esta variable se modifica con la función *toggleJob(job)* y se consulta con *isJobShown(job)*. Se le está dando tanta importancia a esta funcionalidad porque se utiliza en varios casos de uso. El resultado final de la página con varios trabajos cargados se puede ver en la figura 6.9.

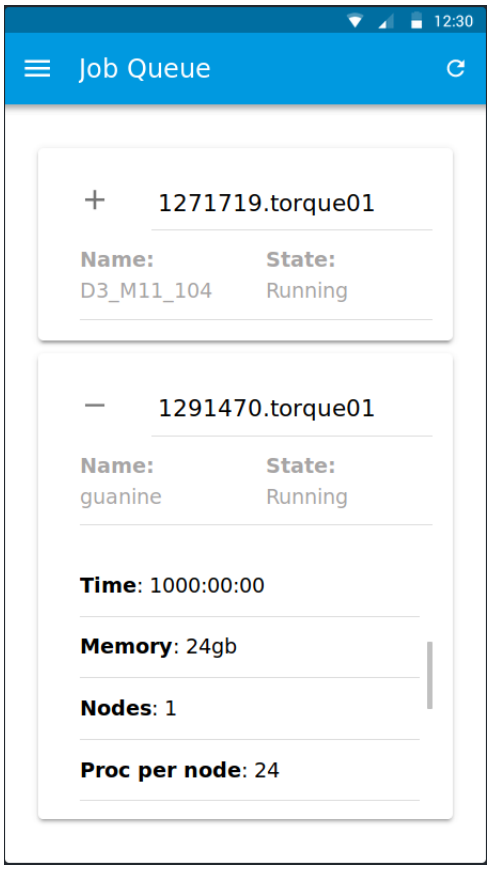

Figura 6.9: Interfaz de usuario de la página *queue*.

La lógica de la página tiene un papel muy simple: llama a el servicio que contacta con la API para pedirle los trabajos del usuario y se encarga de asignar los valores de la respuesta a las diferentes variables de la clase. Del resto se encarga el *Two-Way Data Binding*. Esto se hará siempre que se entre en la página o cada vez que se pulse el botón de recargar ubicado en la esquina superior derecha.

Servicios

En esta página solo se hace uso de un servicio, *network-check.service*. Como en el caos de uso anterior, construye el objeto JSON necesario para realizar la petición y recibe la respuesta de la API.

Back-end

Al igual que en el caso de uso anterior, se lee el cuerpo de la petición y se intenta acceder al clúster del centro. Un pequeño cambio es que a la sesión SSH se le establece un tiempo de expiración de 100 segundos. Esto se hace porque el *script* que se va a ejecutar tarda alrededor de medio segundo en obtener la información de cada trabajo (se va a proponer optimizarlo como una de las mejoras pendientes), por lo que si el usuario tuviese cientos de trabajos tardaría en obtener los datos mucho tiempo.

El comando que se ejecuta en este caso es el *script queue.sh* con el nombre de usuario que hace la petición como parámetro. Este *script* es el que obtiene una lista de trabajos con sus propiedades en formato JSON, y se puede ver el contenido del *script* en los Anexos.

Una vez se obtenga el resultado del *script*, que es un *string* (aunque tenga formato JSON), se transforma en un *array* de objetos, que es el que vamos a enviar a la aplicación y acaba siendo *jobs*.

6.3.3. Comprobar temperaturas

El caso de uso donde se consultan las temperaturas del CPD del centro es único en toda la aplicación porque es único lugar donde se utiliza el protocolo SNMP. Además, es uno de los pocos que hace uso de una barra de pestañas de navegación.

Páginas del caso de uso

Aunque de manera visual solo se parezca que se este usando una página, en cuanto a componentes de Angular se están usando dos: la página *temperatures* y la página *system*. Se puede ver el resultado de combinar las páginas en la figura 6.10.

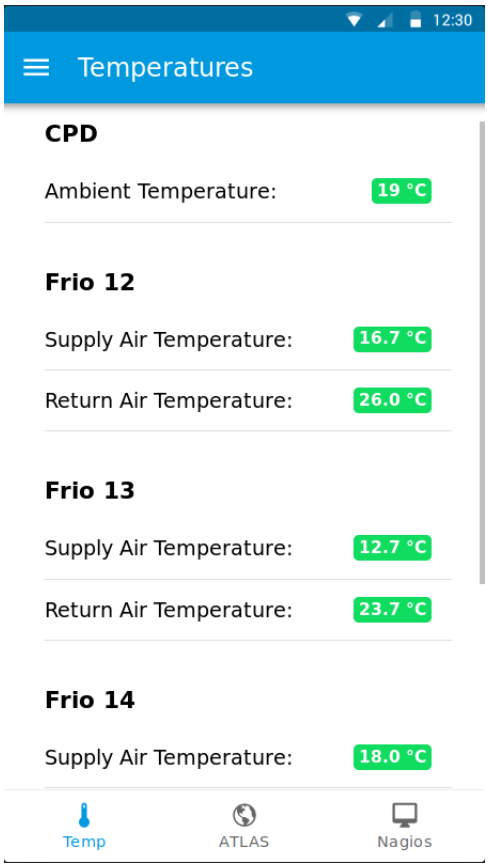

Figura 6.10: Interfaz de usuario de la página *temperatures*.

Temperatures, en cuanto al apartado visual, no aporta nada nuevo. Como mucho se podría destacar que hace uso del componente gráfico *ion-badge* para mostrar el valor de la temperatura y que se modifica el color en función de la temperatura a la que esté su respectivo elemento. Para esto se han establecido unos umbrales a cada elemento de manera arbitraria (se han acordado con el centro de cálculo). Por ejemplo, si una máquina de frío está expulsando aire a 20 grados Celsius o menos se indicará con color verde, si está expulsándolo entre 20 y 30 se indicará con color amarillo, y si está por encima de los 30 grados se indicará con color rojo. Algo que también habría que mencionar es que se ha añadido una suscripción que recarga la página en intervalos de 3 segundos, lo que hace que las temperaturas que se muestran siempre estén actualizadas.

System es la página que contiene la barra de navegación que se usa en las páginas *temperatures*, *atlas* y *nagios*. Su única función es posibilitar la navegación entre estas tres páginas y se hace mediante el uso de los componentes gráficos *ion-tabs* e *ion-tab-bar*, además de modificando el módulo de está página para añadir las otras tres como sus hijas.

Servicios

Como en el caso anterior, el único servicio que se usa es *network-check.service*, para acceder a la API.

Back-end

La API web, al recibir la petición desde la aplicación, tiene que consultar a cada máquina que se esté monitorizando de forma individual mediante SNMP. El problema es que, dependiendo de la máquina, la IP a la que habrá que acceder y el OID que habrá que obtener varía, por lo que es complicado generalizar y hacerlo con código limpio. Lo que se ha acabado haciendo es crear un objeto de PHP por cada máquina, cada uno con sus valores determinados, y insertar todos estos objetos en un *array*. Después se recorrer dicho *array* y por cada máquina, acceder a sus OIDs correspondientes.

Este es el punto en el que se utiliza el protocolo SNMP. Para hacerlo, PHP ofrece una librería opcional que permite utilizar este protocolo y da acceso a cantidad de comandos. Sin embargo, el paquete[2] asociado a esta librería de la distribución de PHP que se está utilizando (*rh-php70*) parece no funcionar correctamente o no se ha conseguido hacer que funcione. La alternativa ha sido instalar el paquete de SNMP de CentOS (*net-snmp* y *net-snmp-utils*) y hacer uso de ellos mediante la ejecución de comandos de PHOP, *shell_exec*. El único problema de esta alternativa es que hay que tratar el *output* del comando para conseguir el valor que nos interesa, mientras que con la función que ofrece el paquete de PHP no debería ser necesario.

6.3.4. Otros aspectos importantes

Acceso a la base de datos

Un aspecto que no se ha mencionado en los casos de uso es el acceso a la base de datos. Como ya se ha explicado antes, CodeIgniter provee una clase llamada *Query Builder* que da acceso a cualquier base de datos que tengamos configurada. Esta clase tiene principalmente tres ventajas:

[2]Debería ser https://centos.pkgs.org/7/centos-sclo-rh-testing-x86_64/rh-php70-php-snmp-7.0.10-2.el7.x86_64.rpm.html

1. **Simplicidad**: Con la sintaxis que se usa para construir sentencias se pueden realizar consultas a la base de datos muy sencillas. Por ejemplo, si queremos obtener todos campos de todos los elementos de la tabla *foo* (`SELECT * FROM foo`), será suficiente con la línea `$this->db->get('foo');`. En cualquier caso, hay que decir que esta clase no está diseñada para generar sentencias muy complejas (como múltiples *join*s) y puede llegar a ser más molesto que utilizando SQL. Es decir, el potencial de *Query Builder* destaca con sentencias sencillas, como en el caso de nuestra base de datos.

2. **Aplicaciones independientes de la base de datos**: Esta clase inserta un nivel de abstracción entre la base de datos y el programador. Ya no hay que preocuparse de que la sintaxis de la sentencia sea correcta. Todas las sentencias que generamos son compatibles con cualquier SGBD[3] que sea compatible con CodeIgniter. Será la clase *Query Builder* la que se encargue de traducir nuestra sentencia al lenguaje correspondiente.

3. **Escape de caracteres**: Las sentencias que genera la clase son más seguras que si lo hiciésemos de forma manual, ya que el sistema se encarga de escapar todos los caracteres.

En la figura 6.11 se puede ver cómo se extraen de la base de datos todos los hilos y sus mensajes utilizando la clase *Query Builder*. Este es el ejemplo de acceso a la base de datos más complejo de todos los casos de uso, por la cardinalidad 1 a N entre las tablas *thread* y *thread_message*. Al principio se crea un *array $dataO* que hará el papel del *where* de la sentencia. Dependiendo de si el usuario es administrador o no, cambiará la condición. Más adelante se ejecuta la sentencia y se almacena el resultado en *$open_query*. Por cada tupla del resultado de la sentencia se empieza a llenar el *array* que se enviará al final del caso de uso, $open. Además, también habrá que seleccionar todos los mensajes de cada tupla y añadirlos al elemento actual de *$open*.

[3] Sistema de Gestión de Bases de Datos.

```php
$dataO = array('closed' => 0);
$dataC = array('closed' => 1);
// If user is admin or not, load threads for admin or from user (token validation changes too)
if ($input_data['admin']){
        $this->_validateToken($input_data['user'], $input_data['token'], true);
        $dataO['admin'] = $input_data['user'];
        $dataC['admin'] = $input_data['user'];
} else{
        $this->_validateToken($input_data['user'], $input_data['token']);
        $dataO['autor'] = $input_data['user'];
        $dataC['autor'] = $input_data['user'];
}

$this->load->database();
// Select relevant data from thread
$this->db->select('id, autor, admin, title');
$open_query = $this->db->get_where('thread', $dataO);
$closed_query = $this->db->get_where('thread', $dataC);
$open = array();
$closed = array();
// For each thread, load all messages
foreach ($open_query->result_array() as $row){
        // Assign result index with thread id
        $open[$row['id']] = $row;
        $this->db->select('id, sender, content, date');
        $message_query = $this->db->get_where('thread_message', array('id_thread' => $row['id']));
        $open[$row['id']]['messages'] = array();
        // For each thread_message, insert it into result
        foreach ($message_query->result_array() as $message){
                array_push($open[$row['id']]['messages'], $message);
        }
}
```

Figura 6.11: Obtención de los hilos de la base de datos.

En la figura también se pueden ver algunos trozos de código que están relacionados con los hilos cerrados. Estos trozos no tienen ningún papel dentro de lo que se pretende explicar con la imagen y están ahí porque se han acabado extrayendo también los hilos cerrados y sus mensajes (al igual que se hace con los abiertos), para que los usuarios puedan consultarlos también. La diferencia entre los dos tipos es que los cerrados no podrán cerrarse ni podrá añadirse nuevos mensajes a ellos.

*Script*s del servidor

En los casos de uso donde se accede a los nodos de cálculo del centro, a menudo se ejecutan unos *script*s, como se puede ver en los diagramas de secuencia del capítulo 5. Estos *script*s ha habido que implementarlos, ya que son necesarios para extraer la información que la aplicación necesita y, al ser información específica del centro, no se ha podido encontrar nada parecido en Internet.

En el caso de uso **'Consulta de los trabajos de cálculo'** había que extraer ciertos cam-

pos del *output* del comando `qstat -f -u $usuario`[4]. El *output* del comando obtiene múltiples parámetros de cada trabajo del usuario que esté en la cola. El problema es que, aunque visualmente sea muy fácil de interpretar, en un programa ha resultado complicado extraer el valor de los campos que nos interesan y convertirlo todo a formato JSON. Este *script* realiza lo que se propone correctamente, pero tiene la pega de tardar demasiado, por lo que se propondrá como mejora en el capítulo 9.

El caso de uso **'Consulta de estadísticas personales'** era especial entre todos porque requería acceso de lectura a un fichero privado del Centro de Cálculo donde se almacena información privada de los usuarios. Parte de esta información era necesaria para calcular las estadísticas, como los créditos consumidos o la memoria reservada. Para solventar el problema, ha habido que implementar un programa en C que tenga el *setuid* activado y que lea e interprete el contenido del fichero y extraiga la información necesaria. En un principio no se esperaba tener que realizar este programa porque ya existía, pero durante el desarrollo del proyecto el Centro de Cálculo cambió de versión de los ficheros y el programa existente se quedó obsoleto. La parte buena es que se ha podido reutilizar parte del código del programa original para implementar el nuevo.

Para finalizar, el caso de uso **'Comprobar estado de ATLAS'** necesita ejecutar dos *scripts* para conseguir toda la información necesaria. El primero de ellos solo son dos líneas, ya que es ejecutar el comando *showq* de Maui y obtener (y transformar) el contenido de las dos líneas del *output* que nos interesan. El segundo *script* es algo más complejo, ya que hay que ejecutar el comando `pbsnodes -l`[5] una vez por cada tipo de estado en el que se puede encontrar un nodo, obtener el resultado (y contar el número de líneas) de cada estado y organizarlo todo en formato JSON.

[4]Comando de torque, documentación en http://docs.adaptivecomputing.com/torque/4-1-3/Content/topics/commands/qstat.htm

[5]Página MAN: https://linux.die.net/man/8/pbsnodes

7. CAPÍTULO

Pruebas

En este capítulo se hablará sobre las pruebas que se han realizado sobre el proyecto para demostrar su correcto funcionamiento. Las pruebas son una de las partes más importantes del desarrollo de un proyecto de software, ya que ayudan a encontrar errores y comportamiento inesperados.

La forma de realizar pruebas para un determinado programa es muy amplía. Se pueden realizar pruebas a bajo nivel, como son las pruebas caja negra y caja blanca, se puede guiar todo el desarrollo del proyecto en base a las pruebas (*Test driven development*) y utilizar las especificaciones del proyecto como documento de validación, etc. Para el caso de una aplicación en Ionic se nos ofrecían además dos opciones: las **pruebas unitarias** de cada componente y las pruebas **e2e** (*end-to-end testing*) de todo el sistema. Al tener que hacer que la aplicación y la API web trabajen correctamente de manera conjunta, se ha decidido hacer pruebas de la API web hasta que devuelva el resultado esperado en cualquier petición y después realizar pruebas sobre la aplicación utilizando los resultados esperados de la API como base. Al igual que en la sección 6.3, se hablará solo sobre las pruebas más notables o únicas, aunque realmente se hayan tenido que realizar pruebas para todos los casos de uso.

Para hacer pruebas sobre la API se ha utilizado una herramienta llamada Postman[1], que permite crear manualmente peticiones HTTP. Esto incluye peticiones POST (las que nos interesan), modificación de los *headers*, calcular tiempos de respuesta, etc. Para el caso

[1] Página web: https://www.getpostman.com/

de Ionic, se nos ofrecen herramientas potentes como Karma[2] o el *framework* Jasmine[3], pero el diseño de este tipo de pruebas habría alargado el proceso de desarrollo, por lo que se ha evitado usarlas. En cambio, se han realizado pruebas de sistema (*system testing*), que se centran en probar 'en vivo' varias funcionalidades, una detrás de otra, intentando buscar los casos extremos en los que el código pueda no comportarse como se espera.

Por último, hay que hablar sobre los dispositivos donde se han realizado las pruebas en conjunto. Se han involucrado dos dispositivos físicos: un iPhone 5S con versión iOS 12.2 y un Xiaomi MiA1 con versión Android 9. También se han probado en dispositivos virtuales: simulador de Android/iOS de Ionic CLI (comando `ionic serve -l`) y emulador de Android Studio Nexus 4 con versión Android 7.0. Para el dispositivo físico Android también se ha probado la versión de producción (que tiene optimizaciones de uso de memoria). Los resultados que se van a mostrar indicarán el peor de todos los resultados obtenidos y, en caso de error, se indicará en qué dispositivo falla. Hay que indicar que en pocas ocasiones se han encontrado diferencias entre los dispositivos en cuanto a funcionalidades (aunque sí en rendimiento).

7.1. Inicio de sesión

Para comprobar que se puede iniciar sesión correctamente utilizando la API y que la respuesta del servidor es la correcta, se han realizado las siguientes pruebas con Postman:

Petición	**Esperado**	**Respuesta**
Sin campo de *username*, o de *password*, o ninguno de los dos	Error: Bad request	Error: Bad request
Con los campos necesarios pero credenciales incorrectos	Error: SSH Fail	Error: SSH Fail
Credenciales correctos como administrador	Autenticación correcta (administrador)	Autenticación correcta (administrador)
Credenciales correctos como investigador	Autenticación correcta (investigador)	Autenticación correcta (investigador)

Tabla 7.1: Pruebas de la API para el inicio de sesión.

Una vez que hemos conseguido el resultado esperado de la API, se procede a probar la

[2]Página web: https://karma-runner.github.io/latest/index.html
[3]Página web: https://jasmine.github.io/

aplicación y la API en conjunto. Se va a agrupar también la acción de cerrar sesión, para probar que no es posible seguir conectándonos después:

Prueba	**Esperado**	**Resultado**
Alguno de los campos vacíos, o los dos	No se envía la petición y se notifica	No se envía la petición y se notifica
Campos rellenos pero sin conexión	No se envía la petición y se notifica	No se envía la petición y se notifica
Credenciales incorrectos (usuario y contraseña no coinciden)	Error: SSH Fail (y se notifica)	Error: SSH Fail (y se notifica)
Credenciales correctos como investigador	Navegación a la página principal	Navegación a la página principal
Credenciales correctos como administrador	Elección de la zona a la que se desea acceder y acceso a ella	Elección de la zona a la que se desea acceder y acceso a ella
Cerrar sesión mediante el botón del menú	Navegación a *login* y eliminación del token	Confirmación seguida de navegación a *login* y eliminación del token
Cerrar sesión mediante el botón *atrás* (Android)	Igual que el caso anterior	Igual que el caso anterior
Cerrar sesión mediante el *right swipe* (iOS)	Igual que el caso anterior	Se cierra la sesión pero se salta la confirmación

Tabla 7.2: Pruebas en conjunto para el inicio de sesión.

Como se puede ver en la tabla 7.2, no se ha encontrado un caso donde el inicio de sesión no funcione como se esperaba. Este caso de uso es crítico y era necesario que su comportamiento fuese el que deseado. Aún así, se deberían hacer más pruebas con dispositivo iOS, ya que tienen una funcionalidad al deslizar hacia la derecha la pantalla *right swipe* que permite volver a la última página visitada. En el dispositivo iOS físico este 'gesto' (*gesture*) hace que se vuelva a la página de *login*, pero no se pregunta antes de cerrar la sesión, como en el resto de casos. Además, habría que probar que no se pueda volver atrás desde *login* a *home* o *home-admin* después de haber cerrado sesión (no se ha conseguido recrear pero no se descarta que sea posible).

7.2. Consulta de los directorios

Hay que comprobar que se pueden cargar correctamente todos los directorios a los que tiene permiso. También hay que comprobar que se aceptan cualquier tipo de ruta, tanto relativa como absoluta, y hacer que el caracter '.' y '..' funcionen como deben ('.' se refiere a la carpeta actual y '..' a la carpeta padre). Además, también se van a añadir el caso de uso 'Ver contenido de un fichero', ya que extiende a este y se sabe que tiene un error en los dispositivos Android.

Lo primero es comprobar que la API funciona de la forma esperada. Las pruebas han sido:

Para un directorio

Petición	Esperado	Respuesta
Sin alguno de los campos necesarios	Error: Bad request	Error: Bad request
Con los campos necesarios pero credenciales incorrectos	Error: SSH Fail	Error: SSH Fail
Credenciales correctos pero token incorrecto / caducado	Error: Invalid token	Error: Invalid token
Credenciales y token correctos pero directorio inexistente	Error: incorrect	Error: incorrect
Credenciales y token correctos pero directorio con espacios	Carga hasta el primer espacio	Carga hasta el primer espacio
Credenciales y token correctos pero directorio con comando concatenado	Al tener que haber espacios, carga hasta el primer espacio	Al tener que haber espacios, carga hasta el primer espacio
Credenciales y token correctos pero directorio sin permisos	Error: incorrect	Error: incorrect
Credenciales y token correctos y directorio correcto	Carga correcta de los contenidos	Carga correcta de los contenidos
Credenciales y token correctos y directorio correcto con múltiples '.' y '..'	Carga correcta de los contenidos	Carga correcta de los contenidos

Tabla 7.3: Pruebas de la API para cargar un directorio.

Para un fichero (se han omitido algunas comprobaciones porque funcionan de la misma manera)

Petición	Esperado	Respuesta
Tamaño del fichero superior al máximo	Error: tooLarge	Error: tooLarge
Tamaño del fichero correcto pero fichero muy grande[4]	Carga exitosa	Carga exitosa
Fichero sin permisos	Error: incorrect	Error: incorrect
Fichero inexistente[5]	Error: binary	Error: binary
Fichero binario	Error: binary	Error: binary
Fichero existente y no binario	Carga correcta del contenido	Carga correcta del contenido

Tabla 7.4: Pruebas de la API para cargar un fichero.

Desde la aplicación solo hay dos maneras de cargar un directorio: haciendo clic sobre uno que se muestre en pantalla (hijos del directorio actual o su padre) o escribiendo una ruta (relativa o absoluta) en la barra inferior. La primera de las dos alternativas no tiene más prueba que probar con un hijo y con el padre. Para la segunda, habrá que hacer las mismas pruebas que en la API. También va a haber que probar el botón de recarga.

Prueba	Esperado	Resultado
Clic sobre hijo	Carga correcta	Carga correcta
Clic sobre padre	Carga correcta	Carga correcta
Directorio inexistente	Error loading directory	Error loading directory
Directorio con espacios	Carga hasta el primer espacio	Carga hasta el primer espacio
Directorio con comando concatenado	Al tener que haber espacios, carga hasta el primer espacio	Al tener que haber espacios, carga hasta el primer espacio
Directorio sin permisos	Error loading directory	Error loading directory
Directorio correcto	Carga correcta de los contenidos	Carga correcta de los contenidos
Directorio correcto con múltiples '.' y '..'	Carga correcta de los contenidos	Carga correcta de los contenidos
Recarga de la página	Carga correcta del directorio actual	Carga correcta del directorio actual

Tabla 7.5: Pruebas en conjunto para cargar un directorio.

Para cargar un fichero, la única manera de hacerlo es haciendo clic sobre él. Una vez abierto, habrá que poder navegar por los contenidos del mismo (arrastrando con el dedo) y ocultarlo. Si se abre otro fichero, el que estaba abierto se cierra y se desplazará la vista al nuevo.

Prueba	**Esperado**	**Resultado**
Clic sobre un fichero	Carga y desplazamiento correcto	Carga y desplazamiento correcto
Clic sobre un fichero de más de 1MB	Carga correcta	Aplicación deja de responder (Android)
Desplazamiento vertical del texto	Se desplaza	Se desplaza
Desplazamiento horizontal del texto	Se desplaza	Se desplaza
Clic sobre un fichero abierto	Se cierra	Se cierra
Clic sobre un fichero estando otro abierto	Se cierra el abierto. Carga y desplazamiento correcto	Se cierra el abierto. Carga y desplazamiento correcto

Tabla 7.6: Pruebas en conjunto para cargar un fichero.

El error que sucede en Android al cargar ficheros de tamaño superior a 1MB (aproximadamente) es ocasionado porque el *WebView* de Android es inferior al de iOS. Esto ocurre incluso cuando el dispositivo físico de Android es notablemente superior al de iOS. La solución que se ha implementado es separar el contenido del fichero en trozos (*chunks*) y cargar el primero de estos pedazos. Se ha añadido un botón que permite cargar el siguiente trozo del fichero. Así, se ha conseguido solventar el error a costa de pérdida en la interacción con el usuario.

[4]Este caso solo es posible creando la petición de forma manual (no desde la aplicación) y estableciendo que el tamaño del fichero que se quiere cargar sea menor que el máximo, aunque realmente supere ese límite. Esto no es un problema, porque la restricción se ha establecido para evitar cargas de ficheros muy grandes y consumir datos del cliente.

[5]Solo es posible creando la petición de forma manual (no desde la aplicación). El mensaje es el mismo que el del binario porque el resultado es el mismo: no se carga el contenido.

7.3. Consulta estadísticas de la aplicación

En este caso de uso hay que realizar dos comprobaciones clave. La primera es que, junto con las acciones que se quieran consultar hay que enviar a la API un número, que es el número de meses de los que se quiere calcular las estadísticas (si es un 6, indica que habrá que calcularlas de los últimos 6 meses). Este número deberá estar entre 2 y 12[6]. Si es mayor, debería tratarse como 12 y si es menor debería tratarse como 2. La segunda comprobación es probar que la petición funciona con un *array* de acciones de cualquier tamaño. Esta comprobación deberá realizarse tanto en la API como en la aplicación (aunque la aplicación solo está preparada para mostrar las estadísticas de las acciones 'login', 'form' y 'newAccount').

Las pruebas para la API han sido:

Petición	**Esperado**	**Respuesta**
Sin alguno de los campos necesarios	Error: Bad request	Error: Bad request
Con los campos pero token incorrecto / caducado	Error: Invalid token	Error: Invalid token
Campos y token correctos con acciones inexistentes	Carga correcta. Los datos de las acciones que no existen serán 0	Carga correcta. Los datos de las acciones que no existen serán 0
Campos y token correctos pero mes mayor que 12	Carga hasta el mes 12	Carga hasta el mes 12
Campos y token correctos pero mes menor que 2	Carga hasta el mes 2	Carga hasta el mes 2
Campos y token correctos pero mes y acción incorrectos	Carga correcta (mes dentro del rango y datos a 0)	Carga correcta (mes dentro del rango y datos a 0)
Campos y token correctos y algunas acciones correctas, otras mezcla de errores	Carga correcta de todas	Carga correcta de todas

Tabla 7.7: Pruebas de la API para obtener estadísticas de la aplicación.

El resultado para la última prueba se puede ver en la figura 7.1. En esta figura se puede ver como de la acción 'login' ha cargado correctamente los datos de los 6 últimos meses, de

[6]La restricción de 12 meses como máximo está por la forma en la que se hace el cálculo de meses en la API. Se propondrá como mejora aumentar este tamaño.

la acción 'forma' ha cargado todos los datos de los 6 últimos meses a 0 (no hay ninguna acción 'forma' en la base de datos) y de la acción 'newAccount' ha cargado correctamente los datos de los 12 últimos meses (se superaba el número de meses así que se coge el máximo).

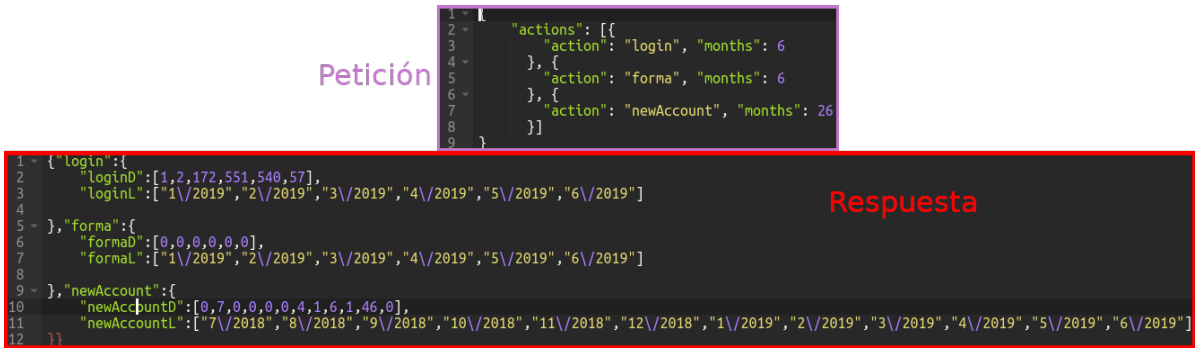

Figura 7.1: Petición y respuesta con mezcla para las estadísticas.

Para la aplicación, se han hecho pruebas con los casos posibles, es decir, con las acciones que se pueden consultar desde la aplicación y los meses también.

Prueba	Esperado	Resultado
Al abrir la página	Se cargan todas las acciones con los valores por defecto	Se cargan todas las acciones con los valores por defecto
Cambiar el valor de una acción a X meses	Carga correcta de esa acción	Carga correcta de esa acción
Peticiones con *arrays* de distintos tamaños	Carga correcta de las acciones	Carga correcta de las acciones
Múltiples pruebas en la misma sesión	Mismo resultado por prueba que en el caso individual	Mismo resultado por prueba que en el caso individual

Tabla 7.8: Pruebas en conjunto para obtener estadísticas de la aplicación.

La última prueba realizada es una prueba de resistencia, donde en la misma página se realizan diferentes acciones sin un patrón concreto para intentar simular la forma de actuar de un usuario. Se puede ver que los resultados tanto de la API como de la aplicación son los esperados, por lo que se concluye que el caso de uso está implementado satisfactoriamente. Además, de la manera en la que se ha implementado el *back-end*, no habrá que hacer ningún cambio para que funcione con nuevas acciones, está totalmente preparado para ello. En el caso de la aplicación, habría que realizar algún cambio, ya que no se ha conseguido generar gráficos de forma automática, según el número de acciones que se quieran

monitorizar (es necesario obtener la referencia de cada uno para poder modificarlo si se necesitase).

8. CAPÍTULO

Gestión del proyecto

La gestión del proyecto de un proyecto de software es diferente de la gestión de otros tipos de proyecto en el sentido de que los proyectos de software tienen un ciclo de vida único que requiere múltiples iteraciones de pruebas, actualizaciones y retroalimentación de los usuarios.

En este capítulo se hablará principalmente sobre las desviaciones que ha habido entre las dedicaciones estimadas y las reales, y sobre las incidencias que han ocurrido durante el desarrollo del proyecto, ya sea por decisión de los interesados o una mala planificación del equipo de desarrollo.

8.1. Desviaciones en las dedicaciones

Al planificar el proyecto (capítulo 2) se hicieron unas estimaciones sobre la dedicación que cada tarea del proyecto iba a necesitar. Esto era importante para calcular si el proyecto era viable y para ser capaces de estructurar el desarrollo del proyecto en el tiempo.

En esta sección el objetivo es contemplar los tiempos previstos de la planificación y compararlos con los tiempos invertidos reales. Así, se pretende calcular lo acertadas que fueron las estimaciones y aprender de los errores para futuros proyectos.

En cuanto al método para comparar ambos tiempos, se van a agrupar los tiempos por tipos de trabajo (como en el capítulo 2) y, como el objetivo es identificar y destacar las diferencias, se va hablar únicamente de las desviaciones que se hayan encontrado.

Desviaciones en el Trabajo de Desarrollo

El trabajo de desarrollo era el tipo de trabajo que más horas se estimaba que iba a requerir, y así ha acabado siendo. También es el tipo que mayor desviación absoluta ha tenido, lo cual es lógico. Las desviaciones han sido:

- El Análisis de Requisitos ha terminado requiriendo 10 horas de trabajo, lo cual supone una desviación de -5 horas.
- La Implementación ha llegado a suponer 240 horas de trabajo, 70 horas por encima de lo estimado. Además de una previsión errónea, han acabado surgiendo algunas incidencias que se van a explicar en la sección 8.2.
- Las Pruebas no se han podido calcular correctamente, ya que siempre se han hecho junto con la Implementación.

En total, ha habido una disparidad de 65 horas, habiendo dedicado realmente alrededor de 275 horas al trabajo de desarrollo.

Desviaciones en el Trabajo Organizativo

Este tipo de trabajo era el que menos horas de dedicación se preveía que iba a necesitar. Ha sido complicado calcular las dedicaciones, ya que muchas veces implica trabajo abstracto y se hace durante otros trabajos (como la gestión, que incluye tareas como comunicación con los interesados). La desviaciones han acabado siendo:

- El tiempo de Gestión ha acabado ampliándose a 25 horas, que es casi el doble del tiempo previsto.

El total de horas dedicadas han sido 45 horas, que son 10 horas más que las que se preveían.

Desviaciones en el Trabajo Formativo

Este tipo de trabajo era el que más tareas tenía, aunque muchas de ellas eran de pocas horas de dedicación (entre 2 y 5). Es por esto que no se esperaba una diferencia muy grande entre los dos recuentos de horas. Las diferencias que ha habido:

- La estructura de la instalación del DIPC no ha necesitado apenas ser estudiada, por lo que se ha acabado dedicando 1 hora a ello, 2 menos de lo esperado.

- Como se descartó pronto en el proyecto la opción de hacer dos aplicaciones nativas, tanto para iOS como para Android, el estudio de los lenguajes correspondientes ha descendido a 5 horas cada uno. Estas horas se corresponden al estudio del entorno de desarrollo (XCode y Android Studio) de estas plataformas, que ha sido necesario para el desarrollo de la aplicación híbrida (en la Guía de Administración presente en los Anexos se puede comprobar).

Al final, las horas invertidas han sido 12 menos de las que se previó, que suman un total de 48 horas.

Desviaciones en el Trabajo Académico

El Trabajo Académico solo englobaba dos tareas: la realización de la memoria del proyecto y la preparación de la defensa del mismo. Esta última tarea no se ha realizado aún, por lo que no se puede comparar el tiempo previsto con el real.

En el caso de la memoria sí que ha habido una desviación. Se previeron 40 horas de dedicación, que han acabado ampliándose a alrededor de 80 horas (es complicado calcularlo, ya que incluye tanto tiempo de redacción como las múltiples revisiones que se han realizado).

Teniendo esto en cuento, el tiempo total real se ha visto ampliado de 50 horas a 90 horas, casi el doble del estimado.

8.2. Incidencias

A lo largo del ciclo de vida del proyecto han ocurrido múltiples incidencias que es necesario comentar para entender bien las conclusiones del proyecto. Algunas de ellas estaban identificadas en el análisis de riesgos del capítulo 2, otras han surgido por un error en la planificación, etc. Estas incidencias han resultado ser una de las causas más importantes para la desviación en el Trabajo de Desarrollo, en concreto en la tarea de Implementación.

*Script*s del servidor

En algunos casos de uso, en concreto en los que es necesario conectarse a los nodos de cálculo del centro y extraer información de la cola de trabajos o información específica del Centro de Cálculo, ha sido necesario programar unos *script*s que cumplan la tarea que se necesita. Este requisito no se identificó en el Análisis del capítulo 4 y no fue hasta la fase de Diseño que se reconoció esta necesidad.

En un principio, se quería obtener la información necesaria mediante la combinación de algunos comandos que no acabase siendo muy complejo, algo que se pudiese realizar y entender en una línea. Sin embargo, al diseñar los casos de uso fue cuándo se vio que esto era imposible (uno de ellos tienes 380 líneas de código). En concreto, los casos de uso que se vieron afectados fueron 'Consulta de los trabajos de cálculo', 'Consulta de estadísticas personales' y 'Comprobar estado de ATLAS'.

Generar estos *script*s imprevistos ha acabado aumentando el tiempo de dedicación en 20 horas. Sin embargo, hay que decir que algunos de estos *script*s van a ser utilizados, además de por la aplicación, por los usuarios del clúster, lo cual supone un beneficio extra del que tenía como objetivo este proyecto.

Servidor de Nagios

Para consultar el estado de la micro-informática, el proceso estaba pensado hacerlo mediante los *script*s que almacena el servidor Nagios, que él mismo también usa. El diseño del caso de uso estaba hecho e incluso se había probado con algunas máquinas de prueba.

La incidencia ocurre cuando se recibe acceso al servidor Nagios real, del dónde había que realizar las consultas. Además de los *script*s, Nagios utiliza unos ficheros de configuración para almacenar la información de todas las máquinas y servicios que está monitorizando. Todo esto se comenta con mayor detalle en el capítulo 3. El servidor de donde se esperaba extraer información había sido preparado por un alumno de prácticas y no había ninguna máquina siendo monitorizada, excepto dos máquinas virtuales de pruebas.

Como configurar las máquinas reales en el servidor está fuera del alcance del proyecto (además de que supondría bastante trabajo, por la cantidad de equipos que hay que añadir) y sin estas máquinas es imposible implementar y probar el caso de uso, se ha decidido no terminar el caso de uso. Se mantendrá la fase de análisis y diseño, ya que pueden ser útiles en un futuro si se acaba configurando, pero la funcionalidad no implementará.

Servidor web real

Para cumplir con uno de los objetivos del proyecto y hacer que la aplicación funcione correctamente desde cualquier lugar con acceso a Internet, es necesario tener un servidor web con una IP pública, para que cada cliente (el *smartphone* ejecutando la aplicación) pueda acceder a él. Sobre este servidor ser puede leer más información en los capítulos 3 y 5.

Hasta el momento de finalización del proyecto, incluso habiendo desplegado la aplicación a Play Store, se ha utilizado el servidor web de prueba. El objetivo era realizar la migración como parte del proyecto, dejar el servidor web real bien configurado y poder hacer un despliegue donde se pueda utilizar la aplicación fuera del centro. Sin embargo, el servidor web no se ha finalizado a tiempo. La encargada de instalar el servidor web no es el propio centro, sino una empresa subcontratada que es también la que está diseñando la nueva página web, por lo que la comunicación con ella ha sido prácticamente nula, exceptuando una conversación al principio del proyecto donde se dio permiso para usar su servidor web (por clausulas contractuales la empresa no estaba obligada).

En el análisis de riesgos ya se contemplaba esta situación, aunque no había un plan de contingencia para el caso. Solo se decía que, dependiendo del tipo de incidencia, las consecuencias variarían. En este caso, las consecuencias han sido no poder hacer el despliegue planeado y que la aplicación sea solo utilizable desde la red del centro (o mediante VPN).

Aunque esto sea una incidencia importante, se ha intentado aliviar haciendo lo más cómoda posible a los administradores la migración en cuanto sea posible. Para la aplicación, con modificar el valor de un atributo del servicio *network-check* y publicar la actualización será suficiente. Para el caso del servidor, una de las razones de escoger CodeIgniter como *framework* fue amenizar el proceso de migración. Además, también se ha realizado una guía de instalación[1] que trivializa gran parte del despliegue (solo falta añadir compatibilidad con SSL/TLS y configurar el *webblock* para que trabaje de manera conjunta con el de la página web).

Cambios de opinión del cliente

Durante el desarrollo de la aplicación ha habido ciertas ocasiones donde el centro ha decidido modificar el funcionamiento de algún caso de uso o modificar la apariencia de

[1] Véase Capítulo 3 del Anexo C.

alguna de las interfaces. Este tipo de cambios de opinión son frecuentes en cualquier proyecto de software, como ya se ha comentado en el capítulo 3, en la sección sobre la metodología.

Cuando el cambio de opinión ha sido sobre la interfaz de usuario de la aplicación, las consecuencias han sido un pequeño retraso para diseñar la nueva interfaz o para añadir alguna funcionalidad nueva. Este tipo de cambios han sido bastantes, aunque ninguno ha sido suficientemente destacable como para hablar en profundidad de él.

Sin embargo, cuando los cambios de opinión suponían cambios en el método de trabajar de algún caso de uso, las consecuencias han sido mayores. Es señalable el cambio que hubo en el caso de uso 'Crear cuenta', realizado por un Administrador.

En un principio, el caso de uso estaba diseñado para que no se crease la cuenta en los servidores del centro, sino que se generasen unos ficheros donde se iban a almacenar los datos del usuario, y que el Administrador pudiese crear la cuenta desde su equipo de trabajo ejecutando un simple programa. Este método de funcionamiento eliminaba el caso de uso 'Realizar comprobaciones', ya que se haría cuando el Administrador crease la cuenta.

No obstante, una vez ya se había implementado el caso de uso casi totalmente y estaban realizándose las últimas pruebas y modificaciones, se decidió cambiar completamente el funcionamiento del caso de uso, haciendo que se comportase tal y como está ahora. Esto supuso un retraso considerable, ya que hubo que rediseñar el funcionamiento y no se pudo aprovechar mucho de lo que ya había hecho, además de añadir un nuevo caso de uso donde había que comprobar que la nueva cuenta estuviese bien creada.

9. CAPÍTULO

Conclusiones

En este capítulo se hablará sobre el estado final del proyecto a la fecha de entrega. Se dividirá en tres apartados, que serán: Objetivos logrados, Propuestas de mejora, y Lecciones aprendidas.

9.1. Objetivos logrados

Los objetivos del proyecto se recogen en el capítulo 2 de esta memoria. Ahora, habrá que repasar esos objetivos y ver cuáles se han conseguido lograr y de qué manera. El contenido de este apartado será indicativo para entender el éxito del proyecto, dependiendo del número de objetivos logrados del total. Además, el cliente real es quién más claramente puede comprobar si se han alcanzado los objetivos iniciales, y su opinión se recoge en el Anexo B de esta memoria.

Se ha conseguido hacer una aplicación para ambos tipos de usuarios, tanto investigadores como la plantilla del centro. La aplicación está disponible para los dos sistemas operativos más populares en el mundo móvil: iOS y Android. En el caso de iOS, la aplicación es compatible desde la versión 10, lo cual hace que el 95 % de los usuarios puedan utilizarla. Android tiene un porcentaje menor, siendo la aplicación compatible desde la versión 7.0, que recoge a alrededor del 70 % de estos móviles. Siendo porcentajes altos, se considera que el objetivo se ha realizado satisfactoriamente.

Respecto a los casos de uso, de los propuestos originalmente se han logrado implementar

todos ellos excepto uno, que se ha llegado a diseñar pero falta por pasarlo a código y probarlo. Además, en algunos de ellos se ha ofrecido más que el objetivo inicial, como en el caso de consultar el informe DCRAB de un trabajo, ya que ahora también se puede abrir cualquier otro fichero. También se ha comentado que ha habido que crear unos *script*s no previstos para realizar ciertas funciones. Con todo esto, se considera que el objetivo que representaba implementar todos los casos de uso se ha cumplido con creces.

Los requisitos listados en el apartado 2.3 también se han cumplido, lo cual era básico para entregar un producto satisfactorio. Algunos de estos requisitos eran subjetivos y solo se considera que se han logrado porque se han seguido las recomendaciones del centro (como en la apariencia de la aplicación) o porque los usuarios que han probado la aplicación han dado su visto bueno (como en la accesibilidad). Otros, eran más objetivos. Se considera que se ha cumplido la normativa establecida por la LOPD en todos los aspectos que incumben al producto. Además, la aplicación funciona correctamente (como se ha demostrado con las pruebas realizadas) y de forma rápida y fluida. El único tiempo de espera que puede hacerse largo es el que carga la cola de trabajos del investigador, que se propondrá como mejora.

Por último, el poder usar la aplicación desde fuera del centro está estrechamente ligado con la incidencia del capítulo 8 que habla de la migración del servidor, ya que el servidor público era una necesidad para cumplir con el objetivo. Ahora mismo, la aplicación es utilizable con cualquier tipo de conexión, siempre que sea desde dentro de la red del DIPC. Sino, es necesario conectarse de antemano mediante VPN. De esta manera, sí que se puede acceder a todas sus funcionalidades en cualquier lugar. Debería ser la primera prioridad de cara al futuro solventar este problema, pero se puede considerar que está cumplido parcialmente, al sí haber una manera de conectarse de forma externa. Además, al haber preparado la migración, lo único que queda para cumplir con el objetivo es esperar a que la empresa encargada de montar el servidor web termine su trabajo. Teniendo todo esto en cuenta, consideraremos que el objetivo está mayormente conseguido.

En conclusión, se ha conseguido cumplir con todo lo que había posibilidad de cumplir. Además, el propio Centro de Cálculo ha considerado el producto satisfactorio, que en última instancia viene a ser el objetivo más importante de todos: que el cliente quede complacido con el producto. Así, podemos concluir que **el proyecto se ha finalizado de forma exitosa**.

9.2. Propuestas de mejora

Aunque los objetivos hayan sido mayormente logrados, sigue habiendo puntos donde el producto puede mejorar. Si no se han realizado estas mejoras y se proponen para el futuro es porque no ha sido posible incluirlas dentro del plazo del proyecto, ya sea por cantidad de trabajo o por falta de conocimiento. Las mejoras son opcionales y no incluyen los objetivos no logrados, ya que es obvio que estos tienen prioridad.

- **Añadir notificaciones *push***: Un elementos muy común en las aplicaciones para teléfonos son las notificaciones *push*. Estas notificaciones aportan a cualquier aplicación mucho dinamismo, ya que permiten avisar al usuario de eventos relacionados con la aplicación e interactuar con ella sin tener que abrirla. En nuestro producto, se podría haber usado para notificar a un usuario cuando su cuenta sea activada/rechazada, para notificar a los administradores cuando las temperaturas se eleven por encima de cierto umbral (para esto habría que usar también los *traps*[1] de SNMP), o cada vez que se responda a un hilo relacionado con el usuario, por ejemplo.

- **Añadir códigos de error a la API**: Una parte básica del protocolo HTTP son sus códigos de respuesta. Al diseñar una API web, es recomendable utilizar estos códigos para responder a las peticiones. En concreto, los códigos que se deberían añadir son los errores del cliente (4XX), que indicarán que el problema es del cliente, como una petición errónea, accesos no autorizados, etc.

- **Mejorar el rendimiento del *script* de la cola**: Ya se ha comentado que el *script* que obtiene información sobre los trabajos en la cola de un investigador es lento cuando la cantidad de trabajos es elevada. Se recomienda reescribir el código del programa desde cero, haciendo que el *output* sea el mismo. Quizás incluso cambiar el lenguaje del código a uno como Python (actualmente está en Bash), que puede ayudar con el complejo tratamiento de *string*s que es necesario.

- **Aumentar el periodo de cálculo de estadísticas**: Al calcular las estadísticas de uso de la aplicación, se ofrecen como posibles periodos 2, 4, 6 y 12 meses. Entre 2 y 12 meses cualquier número es válido, pero con números mayores a 12 el comportamiento no está contemplado. El problema reside en un trozo de código de la función privada de la API *_getStats()*, que habría que modificar.

[1] Introducción de Cisco: https://www.cisco.com/c/en/us/support/docs/ip/simple-network-management-protocol-snmp/7244-snmp-trap.html

- **Monitorizar nuevas estadísticas**: Hasta el momento, las estadísticas de la aplicación que se monitorizan solo son 3: inicios de sesión, peticiones de cuentas y nuevas cuentas. A esta lista podrían añadirse nuevas acciones, como el número de hilo abiertos y cerrados, el número de mensajes enviados (incluso se podría especificar un usuario), número de veces que se ha accedido a alguna funcionalidad en concreto, etc. Las posibilidades son muchas y serviría a los administradores a recoger más información sobre la aplicación que les puede interesar.
- **Editar ficheros**: La idea de poder acceder a los ficheros servía para que un investigador comprobase el resultado de sus trabajos, los errores que haya podido haber, etc. Pero si además de esto se permitiese modificar los ficheros también, permitiría a un investigador modificar los parámetros de los *script*s de ejecución de sus trabajos (es muy común ejecutar el mismo trabajo con ciertos parámetros modificados para después comparar resultados). Además, en un futuro se podría habilitar una opción que permita ejecutar trabajos mediante la aplicación. De todas maneras, estas potenciales funcionalidades son arriesgadas, porque ya no supone únicamente consultar información, sino modificarla también.

9.3. Lecciones aprendidas

Además de los objetivos logrados con el producto final, que son a los que se hacía referencia en el capítulo 2, existen otros objetivos igual de importante que también se han alcanzado. Estas son las lecciones aprendidas durante el ciclo de vida del proyecto.

Durante estos meses, una de las enseñanzas más importantes ha sido la de trabajar de forma autónoma. Es cierto que ha habido constante apoyo por parte de la tutora y muchas veces ha habido que recurrir a los trabajadores del centro en busca de ayuda, pero mayoritariamente el trabajo ha sido unipersonal. Trabajar solo puede no ser algo nuevo por sí mismo, pero desde luego sí que lo es si hablamos de proyectos de tal importancia como tiene este. Realizar el proyecto de forma individual ha supuesto tener que aprender a gestionar el tiempo adecuadamente y ser capaz de organizarse, aprender a resolver cantidad de problemas de los que no se sabía nada, aprender a leer la documentación de las herramientas y tecnologías que se pretenden utilizar antes de utilizarlas, etc. En conclusión, era un mundo nuevo de donde se han extraído múltiples lecciones.

Realizar este proyecto también ha supuesto aprender a utilizar muchas tecnologías completamente nuevas. Entre ellas se encuentran el protocolo SNMP, el monitor de sistemas

Nagios, aprender a programar *script*s de Unix, y, desde luego, el *framework* Ionic y Angular, sobre los que no se sabía nada y han acabado siendo las herramientas con las que más ha habido que trabajar. Todo esto ha supuesto una experiencia nueva, donde de forma independiente ha habido que buscar estas herramientas, analizar si eran lo que el proyecto necesitaba, y aprender a utilizarlas sin ninguna garantía de que vayan a acabar sirviendo (no como en el resto de proyectos del grado).

Por último, es importante mencionar que, al haber realizado el proyecto en una empresa, hay otro tipo de aprendizaje que se ha obtenido. Se ha aprendido lo que es un entorno de trabajo real, donde hay una jerarquía de puestos establecida y donde cada trabajador tiene un rol. Integrarse en la dinámica de trabajo del centro ha sido parte del proyecto, y no solo ha sido un beneficio, sino también una necesidad.

Bibliografía

[1] Documentación de la version 4 de Ionic
https://ionicframework.com/docs
©2019 Ionic, Licencia *MIT*

[2] Documentación oficial de Angular
https://angular.io/docs
Super-powered by Google ©2010-2019. Code licensed under an MIT-style License. Documentation licensed under CC BY 4.0.

[3] Shyam Seshadri.
Angular: Up and Running. (Inglés)
O'Reilly Media Inc, Junio 2018.

[4] Documentación oficial de Cordova
https://cordova.apache.org/docs/en/latest/
© 2012, 2013, 2015 The Apache Software Foundation, Licensed under the Apache License, Version 2.0.

[5] Página oficial de JWT
https://jwt.io/
©2013-2014 Auth0 Inc., Licencia *MIT*

[6] Estándar sobre JWT
https://tools.ietf.org/html/rfc7519

[7] Documentación oficial de Nginx
https://nginx.org/en/docs/
©2002-2019 Igor Sysoev, ©2011-2019 Nginx, Inc., 2-clause BSD-like license

[8] Documentación oficial de CodeIgniter
https://www.codeigniter.com/user_guide/
©2014-2019, British Columbia Institute of Technology

[9] Documentación oficial de MariaDB
https://mariadb.com/kb/en/library/documentation/
©2019, MariaDB

[10] Foro de programación Stack Overflow
https://stackoverflow.com/

Anexos

Acuerdo entre las partes

Con el fin de dejar claras las intenciones del DIPC de cara al proyecto, se ha firmado un acuerdo entre el alumno y la empresa que lista los requerimientos que deberá cumplir el proyecto y las condiciones bajo las que se acepta realizarlo. Este acuerdo es un acuerdo amistoso, que no incluye consecuencias en caso de que se incumpla alguna de las clausulas.

ACUERDO PARA LA ELABORACIÓN DEL TRABAJO DE FIN DE GRADO EN LA FUNDACIÓN DONOSTIA INTERNATIONAL PHYSICS CENTER

En Donostia/San Sebastián, a 25 de febrero de 2019.

INTERVIENEN

De una parte, Urko Lekuona Rico, mayor de edad, con DNI 72547657E y domicilio Pª de Mons 106 2A, 20015, Donostia/San Sebastián, actuando en su condición de alumno del Grado de Ingeniería Informática de la Universidad del País Vasco y autor del Trabajo de Fin de Grado Aplicación para smartphones de DIPC.

De otra parte, Domingo Romero Asturiano, mayor de edad, con DNI 34087590A, actuando en su condición de Director del Centro de Cálculo de la Fundación Donostia International Physics Center (en adelante, DIPC), con CIF G20662292 y domicilio Paseo Manuel de Lardizábal 4, 20018, Donostia/San Sebastián.

EXPONEN

El 28 de enero de 2019 el DIPC propuso al alumno Urko Lekuona un TFG en el que se implementará una aplicación para smartphones que el Centro de Cálculo y los investigadores del DIPC puedan utilizar a diario para facilitarles algunas tareas de sus respectivos trabajos.

Y, en base a lo anterior,

ACUERDAN

Primero.- La aplicación deberá estar desarrollada tanto para Android como para iOS.

Segundo.- La aplicación deberá ser sencilla de usar y tener una apariencia actualizada, similar a la del resto de aplicaciones del mercado.

Tercero.- Una vez acabado el proyecto, el alumno no estará obligado a mantener la aplicación actualizada. También, el DIPC pasará a ser dueño de la aplicación y del código fuente de la misma.

Cuarto.- La aplicación deberá estar formada por las siguientes funcionalidades:

- Los investigadores podrán solicitar una cuenta de cálculo y deberán ser capaces de consultar el estado de sus trabajos de cálculo en la cola de trabajos del supercomputador ATLAS. Asimismo, podrán comprobar el contenido de sus

directorios y ver estadísticas de cómputo personales. Por último, la aplicación les facilitará el ponerse en contacto con un administrador para poder reportar problemas o hacer consultas.

- Los administradores de la aplicación (personal del Centro de Cálculo) podrán aceptar las solicitudes de creación de cuentas de cálculo enviadas por los investigadores, además de leer y responder a los mensajes que hayan recibido de los mismos. Además, podrán consultar el estado de ciertas instalaciones del DIPC, como la temperatura del CPD o información sobre el estado de la microinformática del centro.
- Para finalizar, se obtendrán estadísticas de uso de la aplicación que los administradores podrán consultar para recoger datos de la misma. Algunas de las estadísticas que se obtendrán serán el número de inicio de sesión y de registro mensuales, por ejemplo.

Para que así conste y en prueba de conformidad y aceptación con todo lo anteriormente expuesto, se firma el presente documento en el lugar y fecha indicados en el encabezamiento.

Urko Lekuona

Domingo Romero

Carta de la empresa

En este anexo se expone una carta escrita por el director del Centro de Cálculo y tutor del proyecto por parte de la empresa, donde se muestra su nivel de satisfacción con el producto y con el trabajo realizado.

Donostia-San Sebastián, 20-6-19

Por el presente documento, la Fundación Donostia International Physics Center (DIPC), como destinataria del proyecto fin de carrera "Aplicación para Smartphone para facilitar el uso y ejecución de los servicios del Centro de Cálculo por parte de los investigadores del DIPC y de los propios técnicos del Centro", desarrollado por el alumno Urko Lekuona Rico, manifiesta:

- Que la aplicación desarrollada por el alumno cumple a la perfección con las necesidades requeridas.
- Que su uso facilitará en gran medida la creación y uso de las cuentas de cálculo que dan acceso al principal supercomputador del DIPC (ATLAS).
- Que el alumno ha desarrollado su trabajo con un alto grado de profesionalidad en todas las fases del proyecto.
- Que la excelente documentación y actitud proactiva del alumno permitirá incrementar fácilmente las funcionalidades de la aplicación en un futuro próximo.

Por todo ello, la Fundación Donostia International Physics Center, y en su nombre Txomin Romero, como director del Centro de Supercomputación del DIPC y tutor del proyecto en la Empresa, valoramos excelentemente tanto al alumno como al trabajo realizado por el mismo en el centro.

Firmado

Txomin Romero Asturiano

Director del Centro de Supercomputación del DIPC

Manual de Administración

Para que los empleados del Centro de Cálculo sean capaces de hacer su rol como administradores de la aplicación, se ha redactado una guía que explica en detalle el proceso de montaje de los servidores que se han necesitado utilizar y un manual que explica la instalación de Ionic en un ordenador, para que sean capaces de desarrollar actualizaciones (y publicarlas) tanto para Android como para iOS.

Manual de administración para la aplicación para smartphones del DIPC

Urko lekuona

Mayo 2019

Índice General

1 — Introducción

Este documento tiene como objetivo explicar las diferentes tareas de administración relacionadas con la aplicación para smartphones del DIPC. Se hablará en detalle de las diferentes tareas envueltas en el proceso de montar y mantener el sistema necesario para que el producto funcione correctamente.

En el proyecto hay tres partes que hay que adminsitrar y se pueden diferenciar claramente, que son:

- **El proyecto de Ionic**: Se refiere a la instalación de las herramientas necesarias para el desarrollo de la aplicación, así como el proceso de publicar la aplicación en las tiendas (y actualizaciones) y otras peculiaridades.
- **El servidor web**: Se refiere a la instalación y configuración de la máquina que va a alojar la API que realiza el papel de *back-end* de la aplicación.
- **La base de datos**: Se refiere a la instalación y configuración de la máquina que almacena ciertos datos que la aplicación necesita.

2 — Proyecto de Ionic

2.1 Instalación de Ionic, dependencias y generación de proyectos

Para poder trabajar con Ionic es necesario primero instalar sus dependencias. En concreto son dos, pero vienen incluidas dentro del mismo paquete: NodeJS y npm. Para instalarlos, lo mejor será acceder a https://nodejs.org/en/download/ y descargar la última versión disponible para el sistema operativo en el que se vaya a trabajar. Para este manual se va a asumir que el sistema operativo está basado en Linux y se va a utilizar la instalación con binarios (x64).

Una vez descarguemos el archivo, extraemos su contenido al directorio donde queramos instalar Node (por ejemplo `/usr/local/lib/nodejs`):

```
$ VERSION=v10.15.0
$ DISTRO=linux-x64
$ sudo mkdir -p /usr/local/lib/nodejs
$ sudo tar -xJvf node-$VERSION-$DISTRO.tar.xz \
-C /usr/local/lib/nodejs
```

Ahora habrá que añadir este directorio al PATH y hacer que se cargue automáticamente con cada sesión. Añadir al final del archivo *~/.profile* o *~/.bashrc* el siguiente texto y después recargarlo (`. ~/.profile`):

```
# Nodejs
VERSION=v10.15.0
DISTRO=linux-x64
export PATH=/usr/local/lib/nodejs/node-$VERSION-$DISTRO/bin:$PATH
```

Se puede probar su correcto funcionamiento con los comandos: `node -v` y `npm version`. El siguiente paso es instalar Ionic CLI. Este es un paquete que se encuentra en el gestor de paquetes npm y nos permite crear y gestionar aplicaciones Ionic desde el terminal. Para instalarlo se puede ejecutar el comando `npm install -g ionic` con permisos de root o sudo.

Después creamos un directorio donde queramos instalar el proyecto y copiamos los ficheros fuente dentro (o realizamos un `git clone` al repositorio git). Si todo ha salido correctamente, podemos ejecutar el comando `ionic info` y veremos un *output* parecido a este:

```
Ionic:

   ionic (Ionic CLI)            : 4.12.0
```

```
Ionic Framework                : @ionic/angular 4.1.2
@angular-devkit/build-angular : 0.13.8
@angular-devkit/schematics     : 7.2.4
@angular/cli                   : 7.2.4
@ionic/angular-toolkit         : 1.3.0
```

Cordova:

```
cordova (Cordova CLI) : 8.1.2 (cordova-lib@8.1.1)
Cordova Platforms     : android 7.1.4, ios 4.5.5
Cordova Plugins       : cordova-plugin-ionic-keyboard 2.1.3,
                        cordova-plugin-ionic-webview 3.1.2,
                        (and 7 other plugins)
```

System:

```
Android SDK Tools : 26.1.1 (/home/pc-481/Android/Sdk)
NodeJS    : v10.15.1
            (/usr/local/lib/nodejs/node-v10.15.1/bin/node)
npm       : 6.4.1
OS        : Linux 4.15
```

Para ejecutar el proyecto en un navegador podemos usar el comando `ionic serve` o `ionic serve -l`. La opción -l indica que se quiere usar Ionic Labs, se recomienda esta opción para poder probar ambas plataformas. El problema del comando `ionic serve` es que no permite utilizar las funcionalidades que provee Cordova ni da una sensación correcta del rendimiento que tiene la aplicación en un dispositivo real, por lo que siempre se recomienda probar los cambios también en un dispositivo físico antes de publicarlos. En cambio, si queremos ejecutarlo en un emulador o en un móvil el proceso es más complejo.

2.2 Desarrollo y despliegue en Android

Para explicar el proceso de instalación de la plataforma Android se va a seguir la guía oficial de Ionic, que se puede encontrar en el siguiente enlace: `https://ionicframework.com/docs/installation/android`. El primer paso es instalar la versión 1.8 del JDK de Java. Es importante asegurarse de que la versión es correcta, ya que Cordova no es compatible con las últimas versiones. También se recomienda en este punto instalar el JRE de la misma versión. El proceso de instalación variará según el sistema operativo. En Ubuntu 18.04, se puede realizar utilizando el comando *apt* si se añade el repositorio *ppa:webupd8team/java* (que se encuentra en `https://launchpad.net/~webupd8team/+archive/ubuntu/java`), pero al haber públicado nuevas versiones de Java, Oracle ha impedido la descarga del JDK8 sin haber realizado un inicio de sesión previo, por lo que este método está obsoleto. Lo recomendado es crearse una cuenta de Oracle y descargarlo de la página oficial.

Si se descarga en formato *.tar.gz*, el fichero debería contener una guía de instalación. Independientemente del método de instalación escogido, al final lo importante es asegurarse de que el sistema operativo es capaz de reconocer nuestra versión Java. Al final de la instalación se recomienda ejecutar los siguientes comandos para comprobar que está todo correcto:

```
$ javac -version
javac 1.8.0_201

$ java -version
java version "1.8.0_201"
Java(TM) SE Runtime Environment (build 1.8.0_201-b09)
Java HotSpot(TM) 64-Bit Server VM (build 25.201-b09, mixed mode)
```

Si no aparece la versión que acabamos de instalar puede ser porque existan múltiples instalaciones de Java en el equipo y no está seleccionada por defecto la última. Para cambiarlo, se puede ejecutar el siguiente comando y elegir la opción que deseemos (se muestra la versión de javac, habrá que ejecutarlo para java también):

```
$ sudo update-alternatives --config javac
There are 2 choices for the alternative javac (providing /usr/bin/javac).

  Selection    Path                                  Priority   Status
------------------------------------------------------------
  0            /usr/lib/jvm/java-11-openjdk/bin/javac   1111      auto mode
  1            /usr/lib/jvm/java-11-openjdk/bin/javac   1111      manual mode
* 2            /usr/lib/jvm/java-8-oracle/bin/javac     1081      manual mode
```

En la ejecución del comando anterior se puede ver que existen dos alternativas para *javac* en el ordenador, la versión 8 de Oracle y la versión 11 de OpenJDK. Por el * podemos ver que está seleccionada la versión de Oracle.

Lo último que queda es modificar el valor de la variable de entorno *$JAVA_HOME*, que es utilizada por Android Studio para compilar el programa. El valor tendrá que coincidir con el directorio raíz de la instalación de Java que hayamos seleccionado. En nuestro caso sería `JAVA_HOME="/usr/lib/jvm/java-8-oracle"`. Es importante también modificar un fichero como *$HOME/.profile* o *$HOME/.bashrc* para que esta variable se cargue cada vez que abrimos una nueva sesión.

Una vez hayamos instalado Java, el siguiente paso es instalar Gradle, que es el compilador que utiliza Android Studio para generar la aplicación. Se puede descargar automáticamente utilizando un administrador de paquetes como SDKMAN! o Homebrew. Hay que tener cuidado con las versiones de Gradle que están disponibles en los repositorios de los administradores de paquetes de Linux porque no están distribuidas de manera oficial y pueden ser versiones no compatibles con la versión oficial. Si se desea instalar manualmente, lo mejor será seguir los pasos de la guía oficial: `https://gradle.org/install/`. Si al final de la instalación ejecutamos el comando *gradle -v* obtendremos un *output* similar al siguiente:

```
$ gradle -v

------------------------------------------------------------
Gradle 4.4.1
------------------------------------------------------------

Build time:   2012-12-21 00:00:00 UTC
Revision:     none

Groovy:       2.4.16
```

```
Ant:          Apache Ant(TM) version 1.10.5 compiled on March 28 2019
JVM:          1.8.0_201 (Oracle Corporation 25.201-b09)
OS:           Linux 4.15.0-50-generic amd64
```

Después de Gradle toca instalar Android Studio y su SDK. El SDK va incluido dentro de la instalación de Android Studio. La mejor forma de instalarlo es siguiendo la guía oficial de instalación, ya que mantendrán actualizado el proceso (`https://developer.android.com/studio/install`). Es importante que a la hora de instalarlo nos apuntemos el directorio de instalación del SDK. Por defecto, suele ser *$HOME/Android/Sdk*. Este directorio habrá que añadirlo a la variable de entorno *$ANDROID_HOME*, igual que antes con Java. Además, también habrá que añadir al *$PATH* unos subdirectorios de este directorio. Lo más sencillo es escribir las siguientes líneas al final del fichero *$HOME/.profile* y después ejecutar el comando `source $HOME/.profile`:

```
export ANDROID_HOME=$HOME/Android/Sdk
export PATH=${PATH}:$ANDROID_HOME/tools
export PATH=${PATH}:$ANDROID_HOME/platform-tools
export PATH=${PATH}:$ANDROID_HOME/emulator
```

El último paso para poder construir aplicaciones Android será aceptar las licencias del SDK. Para esto, será tan fácil como navegar al directorio de instalación del SDK y después a su subdirectorio */tools/bin* (*$HOME/Android/Sdk/tools/bin*. En este directorio, ejecutamos el comando `./sdkmanager --licenses` (¡Importante añadir *–licenses*!). Nos mostrará todas las licencias que quedan por aceptar, y podremos leerlas. Cuando acabemos de leer cada una de ellas habrá que aceptarlas si se quiere usar el SDK.

Ahora está el sistema preparado para transformar la aplicación del estado web a la versión Android. Desde la raíz del proyecto de Ionic se puede ejecutar el comando `ionic cordova platform add android`, que creará el proyecto de Android Studio bajo el directorio *platforms/android*. Ahora tendremos que decidir cómo queremos ejecutar nuestra aplicación en versión Android:

- **Emulador**: Lo primero que habrá que hacer será ejecutar el comando `ionic cordova build android`, que compilará la aplicación en formato *.apk* y dejará el fichero resultante en el proyecto de Android Studio del proyecto de Ionic. Después, habrá que abrir Android Studio ('directorio-instalación'/bin/studio.sh) y abrir el proyecto ya existente ubicado en el directorio del proyecto de Ionic ('proyecto-ionic'/platforms/android). Ahora, ejecutar el *AVD manager* (*Tools / AVD Manager* y añadir un nuevo dispositivo virtual (se puede elegir la opción que mejor nos venga). Con el dispositivo virtual configurado, ejecutar desde la raíz del proyecto de Ionic el comando `ionic cordova emulate android`, que compilará el proyecto y lo lanzará al emulador por defecto (si queremos alguno en concreto se puede iniciar primero el emulador desde el AVD Manager y después ejecutar este comando).

- **Dispositivo físico**: El proceso para instalarlo en un dispositivo físico debería ser algo más sencillo. Necesitaremos un teléfono con una versión Android compatible con la de nuestro proyecto (ver *config.xml*) y que esté habilitada la depuración por USB. Esta opción suele estar en el menú de desarrollador del teléfono (el menú suele estar oculto/desactivado por defecto, habrá que buscar en Internet cómo activarlo para nuestro dispositivo). Una vez esté activada la depuración por USB, conectar el teléfono al ordenador mediante USB. Para asegurarnos de que está detectado (no es

necesario), podemos acceder al *AVD Manager* (ver **Emulador**) y debería aparecer el dispositivo físico en la lista. Llegados a este punto, ejecutar el comando `ionic cordova android run` desde la raíz del proyecto Ionic. Se compilará, instalará y lanzará la aplicación en el dispositivo conectado.

Con esto es suficiente para desarrollar y hacer pruebas con la versión de Android. Si lo que queremos es también publicarlo en la Play Store de Google, es necesario saber que habrá que pagar 25 dólares para crear una cuenta de desarrollador de Google. (desde esta dirección: `https://play.google.com/apps/publish/signup/`) (este paso ya ha sido realizado para este proyecto). El siguiente paso será crear una nueva aplicación desde la consola del enlace anterior (habrá que haber pagado antes e iniciar sesión) y rellenar todos los campos necesarios. Estos campos son relativos a la APP que se esté publicando y serán útiles para calificar la APP con una etiqueta de edades o categorizarla en la tienda, entre muchas otras cosas. Cuando esté todo listo, faltará publicar la primera versión de la APK.

Para esto, lo primero será compilar la APK con los *flags* de producción. El comando es `ionic cordova build android --prod --release`. Este comando generará el *.apk* en la carpeta de *output* del proyecto de Android Studio (por defecto 'raíz-proyecto' /platforms/android/app/build/outputs/apk/release/). Es importante saber que si no es la primera vez que se publica esta aplicación (osea, se está actualizando), es obligatorio modificar el número de versión del fichero *config.xml* ubicado en la raíz del proyecto Ionic (se recomienda también modificar el número de versión del fichero *package.json* en el mismo directorio para hacer que coincidan).

El siguiente paso es generar una clave de firma (*signing key*) para firmar la APK (esta clave ya está generada para este proyecto y se va a proveer con la entrega final del mismo, se puede omitir este paso). El comando es el siguiente, reemplazando los campos *keystore* y *alias* con los que deseemos:

```
$ keytool -genkey -v -keystore my-release-key.keystore \
-alias alias_name -keyalg RSA -keysize 2048 -validity 10000
```

Es muy importante guardar la contraseña que le asignamos en esta clave en el proceso de crearla. Además, también es importante almacenar este fichero de forma segura, ya que nos identifica como autores de la APk. Ahora, habrá que firmar el fichero *.apk* con la clave. El comando es el siguiente, reemplazando los campos*keystore* con la ruta relativa a nuestra clave de firma y los dos últimos parámetros con el nombre del fichero *.apk* y el alias introducido en el paso anterior ('dipc_app' en nuestro caso):

```
$ jarsigner -verbose -sigalg SHA1withRSA -digestalg SHA1 \
-keystore my-release-key.keystore \
HelloWorld-release-unsigned.apk alias_name
```

Ya solo queda un paso para tener la versión final del *.apk* que habrá que publicar en la Play Store. Este paso aplica ciertas optimizaciones al APK que reducen el consumo de memoria RAM de la aplicación y es obligatorio para hacer la aplicación pública. El comando es el siguiente, reemplazando el primer parámetro por el nombre del fichero *.apk* que hemos firmado y el segundo parámetro por el nombre que queramos darle al fichero *.apk* final:

```
$ zipalign -v 4 HelloWorld-release-unsigned.apk HelloWorld.apk
```

Si por alguna razón no se encuentra el comando *zipalign*, está ubicado bajo el directorio *build-tools* del directorio de instalación del SDK de Android (se puede añadir al $PATH, como se ha hecho con otros).

Ya estamos en el punto final de la publicación de la aplicación. Habrá que entrar en la consola de desarrolladores de Google y seleccionar la aplicación que hemos creado antes. Ahora, bajo el menú lateral 'gestión de publicaciones' (*release management*, seleccionar la opción 'publicaciones de la app' (*app releases*). Nos saldrán todos los hilos que tiene disponibles nuestra aplicación. Seleccionamos el que deseemos y elegimos la opción 'crear publicación' (*create release*). En esta nueva página se nos solicitará rellenar nuevos campos de esta versión y cargar el fichero *.apk* final que hemos generado con el comando *zipalign*.

Si todo ha salido correctamente y Google acepta la nueva versión (puede tardar varias horas), tendremos la aplicación disponible en la Play Store desde cualquier dispositivo compatible. El proceso puede parecer largo o tedioso mientras se hace por primera vez, realmente no son muchos pasos y son siempre iguales.

2.3 Desarrollo y despliegue en iOS

Antes de nada, es importante saber que para desarrollar y desplegar la versión de iOS hay que cumplir ciertos requisitos previos. El primero es tener un ordenador iMac físico con una versión del sistema operativo compatible con XCode 9. El segundo es tener un teléfono iPhone físico. El tercero es tener una cuenta de desarrollador de Apple que este activa (cuesta 99 dólares anuales) y estar dado de alta en el programa de desarrolladores de Apple.

Lo primero para hacer pruebas con dispositivo iOS es instalar XCode 9. Este programa es el equivalente de Apple a Android Studio, osea permite desarrollador aplicaciones para iOS de forma nativa. La versión deberá ser la última versión de XCode 9, ya que es la única compatible con Cordova (al menos hasta el momento). Para instalarlo, al no ser la última versión, no está disponible en la tienda de aplicaciones de Apple, por lo que habrá que descargarlo desde el repositorio oficial de versiones antiguas de Apple (`https://developer.apple.com/download/more/`). Es necesario acceder con una cuenta de desarrollador de Apple y habrá que buscar la versión deseada. En este caso es la versión de XCode 9.4.1.

Es importante señalar que, como la versión de XCode no es la más reciente, no trae soporte para versiones de iOS nuevas (para ninguna desde que salio XCode 10). Lo más probable es que el dispositivo iPhone en el que vayamos a probar y desarrollar la aplicación esté actualizado, por lo que no nos va a dejar en un principio instalar la aplicación. La solución al problema es sencilla, hay que añadir los archivos que dan soporte a las nuevas versiones. Hay dos formas de hacerlo: 1. Instalar la última versión de XCode y mover los archivos de la nueva versión a la carpeta de instalación de XCode 9.4.1, y 2. Descargarse únicamente los archivos necesarios de repositorios no-oficiales. El directorio donde habrá que ubicar los archivos es *'directorio-instalacion-XCode9'/Contents/Developer/Platforms/iPhoneOS.platform/DeviceSupport*. Un repositorio de ejemplo donde se incluye soporte para la versión 12 de iOS es: `https://github.com/Yatko/iOS-device-support-files`.

También se recomienda abrir XCode y sincronizarlo con la cuenta de desarrollador de Apple. Esto va a ser necesario tanto para publicar la aplicación en la tienda como para instalarla en cualquier emulador o dispositivo físico a la hora de desarrollarla. Este proceso

se puede hacer desde XCode => Preferencias => Cuentas.

Por último, habrá que conectar el dispositivo físico o poner en marcha un simulador. Desde XCode => Window => Devices and Simulators. Si el dispositivo es físico, también habrá que dar permisos desde el propio teléfono.

Cuando todo esto esté puesto en marcha, lo que queda es instalar NodeJS y npm, Ionic, y poner en marcha el proyecto. El proceso es el mismo que se ha seguido en Linux: Se instala NodeJS desde la página oficial (esta vez la versión de Mac) y traerá incluido el gestor de paquetes npm. Desde npm instalamos Ionic. Una vez tengamos Ionic, clonamos el proyecto desde un repositorio o lo extraemos desde donde se esté almacenando. También habrá que añadir la plataforma, como se ha hecho en Linux con Android. El comando en este caso sería `ionic cordova platform add ios`. Cuando esté creada, ejecutamos también el comando `ionic cordova prepare ios`, que tiene la misma función que el comando *build*, pero sin llegar a compilar la aplicación (copia los archivos de */www* a */platforms*, prepara el fichero *config.xml*, genera los recursos de la plataforma, ...). La compilación se hará desde el propio XCode, para poder firmar la aplicación con la cuenta de desarrollador.

Llegados a este punto, queda compilar e instalar la aplicación en un dispositivo. Habrá que abrir el proyecto en XCode (igual que en el caso de Android, se encuentra bajo la carpeta *platforms/ios* en la raíz del proyecto de Ionic). Dentro del proyecto, se recomienda modificar el equipo de firmas del proyecto siguiendo el orden que aparece en la figura 2.1.

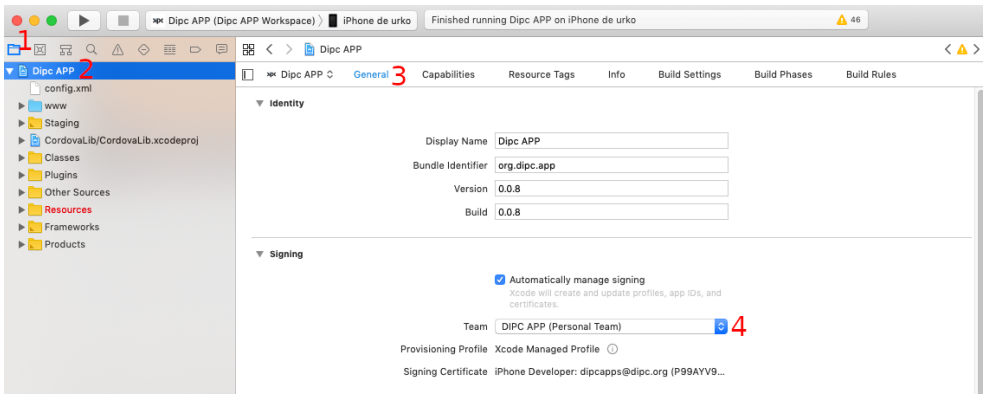

Figure 2.1: Pantalla de opciones del proyecto.

Ya debería estar todo listo. Seleccionamos el dispositivo al que queremos desplegar la aplicación y pulsamos el botón de ejecutar de la parte superior izquierda de la ventana. XCode empezará a compilar y firmar la aplicación y, cuando termine, pasará a instalarla y lanzarla en el dispositivo seleccionado.

3 — Instalación y puesta en marcha de los servidores

Para ambos servidores se va a utilizar el mismo sistema operativo: CentOS Linux release 7.6.1810 (Core). Para ambas instalaciones es necesario permisos de *root*. Se recomienda que ambas máquinas tengan al menos 2GB de memoria RAM y deberán tener acceso a la red.

3.1 Instalación del servidor web

Antes de nada, hay que explicar que en esta sección nos estamos refiriendo a la instalación del servidor web **antes de haber hecho la migración**. Es decir, este servidor web no estará pensado para albergar dos páginas web diferentes. No obstante, se puede extraer información de esta guía para realizar la migración al servidor web definitivo. Lo único que debería cambiar será la configuración del servidor Nginx, haciendo posible el *hosting* de varias páginas en el mismo servidor, habilitando las conexiones mediantes SSL/TLS (incluso forzando las conexiones mediantes estos protocolos), y asignando un dominio específico a las páginas. El resto de elementos de la guía, como los paquetes necesarios, el orden de los pasos, y los comandos que hay que ejecutar, seguirán siendo necesarios independientemente de si es el servidor definitivo o no.

Lo primero que vamos a hacer es instalar el servidor web Nginx y ponerlo en marcha. Después de actualizar los paquetes que haya instalados (es necesario tener permiso de *root* para este paso como para la mayoría), hay que añadir Nginx a la instalación. Nginx no es parte de los repositorios por defecto de CentOS, por lo que habrá que añadir el repositorio al que pertenece. También habrá que permitir las conexiones HTTP y HTTPS a través del *firewall*:

```
# Anadir repositorio y paquete
$ yum install epel-release
$ yum install nginx

# Habilitar el servicio e iniciarlo automaticamente
$ systemctl start nginx
$ systemctl enable nginx

# Permitir conexiones HTTP y HTTPS
$ firewall-cmd --permanent --zone=public --add-service=http
$ firewall-cmd --permanent --zone=public --add-service=https
```

```
$ firewall-cmd --reload
```

Con esto ya podríamos probar que el servidor esté funcionando. Si accedemos a la IP del servidor web mediante un navegador deberíamos ver la página por defecto de Nginx. Las maneras de configurar un servidor Nginx son infinitas y explicar el por qué de cada parámetro en esta guía es imposible. Por lo tanto, lo que se va a hacer es copiar la configuración que se ha usado en el servidor de prueba, que ha dado buenos resultados.

Lo primero será reemplazar la configuración de Nginx por defecto, que se encuentra en */etc/nginx*. Se hace un *backup* de este directorio (renombrarlo o copiar y eliminar el original) y se extrae la configuración del servidor de pruebas (fichero *backUp_nginx.tar.gz*, que supondremos que está en el directorio *home* de *root*). El resultado debería ser un directorio llamado */etc/nginx* que contendrá múltiples ficheros y subdirectorios. Para asegurarnos de que se ha hecho correctamente, acceder al subdirectorio *conf.d* y comprobar que el contenido del fichero *webprueba.sw.ehu.es* es:

```
server {
    listen          80 default_server;
    listen          [::]:80 default_server;
    server_name     webprueba.sw.ehu.es www.webprueba.sw.ehu.es;
    root            /var/www/webprueba.sw.ehu.es;
    index           index.php index.html;

    add_header 'Access-Control-Allow-Origin' *;
    add_header 'Access-Control-Allow-Headers' 'Content-Type';
    location / {

            try_files $uri $uri/ /index.php;

            location = /index.php {

                fastcgi_pass   127.0.0.1:9000;
                fastcgi_param   SCRIPT_FILENAME    $document_root$fastcgi_script_name;
                include        fastcgi_params;
            }
        }

        location ~ \.php$ {
            return 444;
        }
}
```

Figure 3.1: Configuración de la página.

En esta configuración se puede ver el puerto en el que está escuchando la página, el nombre que se le ha dado, los *headers* que queramos incluir, etc (de nuevo, esta configuración es para el servidor de prueba, para el . El parámetro *root* indica dónde está ubicada la raíz de nuestro servidor. Utilizar */var/www/'NOMBRE_DEL_SERVIDOR'* no es obligatorio, pero es la costumbre. Lo siguiente que vamos a hacer será crear ese directorio y llenarlo con los contenidos de la página web (ubicados en el fichero *backUp_webBlock.tar.gz*, que supondremos que está en el directorio *home* de *root*):

```
$ mkdir -p /var/www/webprueba.sw.ehu.es
$ tar -xvf $HOME/backUp_webBlock.tar.gz -C /var/www/
```

El resultado debería ser un directorio llamado */var/www/webprueba.sw.ehu.es* que contenga, entre otros, un fichero *index.php*. En este directorio se encontrará parte de la instalación de CodeIgniter. En una instalación por defecto, estaría todo contenido aquí. Sin embargo, CodeIgniter recomienda mover algunos de sus directorios fuera de la

raíz del sitio web, por seguridad. En nuestro caso, esos directorios están en el fichero *backUp_appCI.tar.gz* (en el *home* de *root*) y se ha decidido ubicarlos en el directorio */opt/appCI*:

```
$ mkdir -p /opt/appCI
$ tar -xvf $HOME/backUp_appCI.tar.gz -C /opt/
```

El resultado debería ser un directorio llamado */opt/appCI* que contenga dos subdirectorios, *application* y *system*. Si nos hemos fijado durante el proceso de instalación nos habremos dado cuenta de que el servidor usa lenguaje PHP (como en el fichero *index.php*). CentOS no tiene por defecto instalado un intérprete PHP, por lo que habrá que añadirlo. La versión PHP que se ha elegido es la 7.0, que se encuentra en un repositorio oficial pero no habilitado por defecto. Los pasos para habilitar e instalar PHP son:

```
# Instalar PHP
$ yum install centos-release-scl
$ yum install rh-php70
$ scl enable rh-php70 bash

# Paquetes necesarios para PHP
$ yum install rh-php70-php
$ yum install rh-php70-php-fpm
$ yum install rh-php70-php-mbstring
$ yum install rh-php70-php-mysqlnd

# Habilitar el servicio que se encarga de las peticiones PHP
$ systemctl start rh-php70-php-fpm
$ systemctl enable rh-php70-php-fpm

# Habilitar conexiones remotas (SELinux)
$ setsebool -P httpd_can_network_connect on
$ setsebool -P httpd_can_network_connect_db on

# Marcar como contenido HTTP de solo lectura
$ chcon -Rt httpd_sys_content_t /var/www/webprueba.sw.ehu.es
$ chcon -Rt httpd_sys_content_t /opt/appCI

# Paquetes para BD y SNMP
$ yum install mariadb
$ yum install net-snmp net-snmp-utils
```

Llegados a este punto, el servidor debería estar listo. Se recomienda reiniciar los servicios *nginx* y *rh-php70-php-fpm* después de hacer cambios sobre la configuración, para asegurarse de que los cambios han tomado efecto. También se recomienda probar el funcionamiento de la API web una vez llegados a este punto. Con una llamada las funciones que hacen uso de la base de datos y de SNMP debería ser suficiente (si funcionan, no solo indica que la base de datos y SNMP funcionan, sino también que el servidor web está correctamente configurado y que el interprete PHP también.

Como ya se ha dicho, la instalación de CodeIgniter está ubicada en */opt/appCI* y tiene la estructura de cualquier instalación de CodeIgniter. Se recomienda leer sobre ella

en la página oficial[1]. Los ficheros de configuración se encuentran en *application/config* y el código de la API en *application/controllers*.

3.2 Instalación del servidor con la base de datos

La instalación de la base de datos es más sencilla que la del servidor web. Requiere dos cosas: una máquina CentOS 7 con conexión a Internet y permisos de *root*, y el *dump* de la base de datos que se va a proporcionar con la entrega del proyecto (esto es opcional, pero evita tener que diseñarla, llenarla de datos, configurar los permisos, etc).

Lo primero que se recomienda hacer es actualizar todo los paquetes que el sistema operativo necesite. Usando el administrador de paquetes *yum* se hace así (como *root* o usuario con permisos): `yum update`. Este proceso es probable que tarde varios minutos.

Cuando acabe, lo siguiente será instalar los paquetes necesarios para utilizar MariaDB. En este caso son 2 (y sus dependencias) y se encuentran en los repositorios por defecto de *yum*. Se pueden instalar los 2 usando: `yum install mariadb mariadb-server`.

Al terminar, es necesario activar el servicio que gestiona la base de datos y habilitar su inicio automático al encender la máquina, para esto:

```
$ systemctl start mariadb
$ systemctl enable mariadb
```

Ahora ya está la base de datos puesta en marcha, se va a proceder a importar el *dump* que contiene el esquema de la base de datos que utiliza el servidor web. Para ello, lo primero es iniciar sesión en MariaDB como *root* y crear un esquema con el mismo nombre que el del *dump* (en este caso 'app'). Al iniciar sesión como *root* tendremos que introducir una contraseña. Por defecto, esta contraseña está vacía, así que podremos pulsar *enter* e iniciar sesión. Se recomienda cambiar esta contraseña también cuanto antes y establecer una segura. El listado de comandos para hacer todo esto sería:

```
# Paramos el servicio
$ systemctl stop mariadb

# Lo iniciamos en modo seguro, que otorga todos los permisos
# e impide conexiones remotas (& para hacerlo en background)
$ mysqld_safe --skip-grant-tables &

# Entramos en la base de datos
$ mysql

# Modificamos la contrasena de root
# (sustituir NEW-PASSWORD por la contrasena)
MariaDB [(none)]> UPDATE mysql.user SET Password=PASSWORD('NEW-PASSWORD')
                  WHERE User='root';
MariaDB [(none)]> FLUSH PRIVILEGES;
MariaDB [(none)]> exit

# Parar el servicio en modo seguro
```

[1] `https://www.codeigniter.com/user_guide/general/welcome.html`

```
$ mysqladmin -u root -p shutdown

# Reiniciarlo y probar la nueva contrasena
$ systemctl start mariadb
$ mysql -u root -p
```

Si todo ha salido bien, procedemos a crear el esquema e importarlo del *dump*:

```
MariaDB [(none)]> CREATE DATABASE app;
MariaDB [(none)]> exit

# app-dbdump.sql es el dump que contiene los datos de la base de datos
$ mysql -u root -p app < app-dbdump.sql
```

Se puede acceder al nuevo esquema utilizando el comando `mysql -u root -p app` y listar las tablas con `SHOW TABLES;`. El siguiente paso es configurar el servicio para deshabilitar conexiones remotas como *root*, crear un usuario para las conexiones remotas y dar a ese usuario los permisos justos y necesarios sobre el esquema 'app'.

```
# Deshabilitar conexiones remotas de root
MariaDB [(none)]> DELETE FROM mysql.user WHERE User='root' AND Host NOT IN (
MariaDB [(none)]> FLUSH PRIVILEGES;

# Crear el nuevo usuario (sustituir password por la contrasena)
MariaDB [(none)]> CREATE USER 'dbuser' IDENTIFIED BY 'password';

# Darle privilegios de uso sobre el esquema 'app'
MariaDB [(none)]> GRANT SELECT,INSERT,DELETE,UPDATE
                  ON `app`.* TO 'dbuser'@'%'
                  IDENTIFIED BY 'password';
# IMPORTANTE: app va rodeado del caracter ` (acento grave) y
#             no por ' (comilla simple)
# El '%' (porcentaje) indica cualquier direccion.
# Si queremos una en concreto
# (la del servidor web), sustituir donde corresponde
MariaDB [(none)]> FLUSH PRIVILEGES;
```

De esta manera, habremos conseguido asegurar el acceso a la base de datos de manera correcta. Los únicos usuarios que deberían poder conectarse serán *root* desde *localhost* y *dbuser* desde la IP que hayamos asignado (o todas si se mantiene el porcentaje).

Ahora podría modificarse la configuración del servicio. Los ficheros */etc/mysql/my.cnf*, */etc/my.cnf*, y */.my.cnf* son los que se encargan de ello, en ese orden (es posible que todos no existan). En */etc/my.cnf* se configuran las opciones de todo el sistema, mientras que en */.my.cnf* se podrán sobre-escribir estas opciones para configurarlas por usuario. En nuestro caso, no se ha creído necesario modificar ninguna de las opciones por defecto.

El servidor está listo. Falta comprobar que tenemos accesos desde el servidor web. Antes de modificar la configuración del servidor web se recomienda probarlo con la línea de comandos. Si el servidor web tiene instalado el paquete *mariadb*, se puede usar el comando `mysql -u dbuser -p -h 'IP-SERVIDOR-MARIADB'`. Después de introducir la contraseña, deberíamos ser capaces de acceder como si fuese una conexión local.

En caso de no ser así, habrá que aplicar una solución dependiendo del error que hayamos recibido. Es muy común el error `ERROR 2003 (HY000): Can't connect to MySQL server on 'IP-SERVIDOR-MARIADB'`. Este error surge porque el servidor no es accesible (al menos en el puerto por defecto, 3306). La solución más probable es admitir las conexiones a este puerto mediante *iptables* (se recomienda usar la cabeza y ver si la máquina es alcanzable con el comando *ping* o accesible desde *telnet*). Para aceptar las conexiones al puerto 3306, los comandos son los siguientes, aunque **también habrá que eliminar cualquier regla que pueda estar interrumpiendo esta conexión**:

```
$ iptables -A INPUT -i lo -p tcp --dport 3306 -j ACCEPT

$ iptables -A OUTPUT -p tcp --sport 3306 -j ACCEPT
```

En este punto se va a asumir que hemos sido capaces de conectarnos desde el servidor web al servidor MariaDB utilizando el cliente MariaDB (comando `mysql -u dbuser -p -h 'IP-SERVIDOR-MARIADB'`). Ahora habrá que configurar el servidor web para que se conecte automáticamente al servidor MariaDB. El fichero que gestiona las bases de datos se encuentra en *'raiz de CodeIgniter'/application/config/database.php*. Se recomienda seguir la guía de CodeIgniter[2] para configurar la base de datos. En nuestro caso, la configuración que se ha usado ha sido:

```
$db['default'] = array(
        'dsn'   => '',
        'hostname' => 'IP_SERVIDOR_MARIADB:3306',
        'username' => 'dbuser',
        'password' => 'CONTRASENA',
        'database' => 'app',
        'dbdriver' => 'mysqli',
        'dbprefix' => '',
        'pconnect' => FALSE,
        'db_debug' => (ENVIRONMENT !== 'production'),
        'cache_on' => FALSE,
        'cachedir' => '',
        'char_set' => 'utf8',
        'dbcollat' => 'utf8_general_ci',
        'swap_pre' => '',
        'encrypt' => FALSE,
        'compress' => FALSE,
        'stricton' => FALSE,
        'failover' => array(),
        'save_queries' => TRUE
);
```

Ya estaría configurado el servidor MariaDB que albergará nuestra base de datos. Lo siguiente que habría que hacer sería probar la API y comprobar que todos los accesos a la base de datos se realizan satisfactoriamente.

[2] `https://www.codeigniter.com/user_guide/database/configuration.html`