



TRABAJO DE FIN DE GRADO
GRADO EN INGENIERÍA INFORMÁTICA

Observatorio Remoto: un cliente INDI para Android

Autor

Jaime Torres Benavente

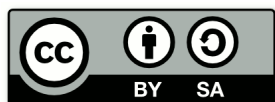
Tutor

Sergio Alonso Burgos



ESCUELA TÉCNICA SUPERIOR DE INGENIERÍAS INFORMÁTICA Y DE
TELECOMUNICACIÓN

Granada, 6 de septiembre de 2015



Observatorio Remoto: un cliente INDI para Android

Jaime Torres Benavente

Yo, **Jaime Torres Benavente**, alumno de la titulación **Grado en Ingeniería Informática** de la **Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación de la Universidad de Granada**, autorizo la ubicación de la siguiente copia de mi Trabajo Fin de Grado (*Observatorio Remoto: un cliente INDI para Android*) en la biblioteca del centro para que pueda ser consultada por las personas que lo deseen.

Además, este mismo trabajo es realizado bajo licencia **Creative Commons Attribution-ShareAlike 4.0** (<https://creativecommons.org/licenses/by-sa/4.0/>), dando permiso para copiarlo y redistribuirlo en cualquier medio o formato, también de adaptarlo de la forma que se quiera, pero todo esto siempre y cuando se reconozca la autoría y se distribuya con la misma licencia que el trabajo original. El documento en formato **LaTeX** así como el código del proyecto se puede encontrar en el siguiente repositorio de **GitHub**: <https://github.com/torresj/indi-android-ui>.

Fdo: Jaime Torres Benavente

Granada, a 6 de septiembre de 2015

D. Sergio Alonso Burgos, profesor del Departamento de Lenguajes y Sistemas Informáticos de la Universidad de Granada.

Informa:

Que el presente trabajo, titulado *Observatorio Remoto: un cliente INDI para Android*, ha sido realizado bajo su supervisión por **Jaime Torres Benavente**, y autoriza la defensa de dicho trabajo ante el tribunal que corresponda.

Y para que conste, expide y firma el presente informe en Granada a 6 de septiembre de 2015.

El tutor:

Sergio Alonso Burgos

Agradecimientos

A mi familia porque gracias a ellos soy como soy.

A mis amigos porque sin ellos no habría podido llegar hasta aquí

A mi tutor **Sergio Alonso Burgos**, por aguantarme durante tantos meses con una paciencia infinita.

Índice general

| | |
|---|-----------|
| 1. Introducción | 1 |
| 1.1. La Astronomía | 1 |
| 1.2. Instrumental Astronómico | 3 |
| 1.2.1. Telescopios | 3 |
| 1.2.2. Cámaras CCD | 4 |
| 1.2.3. Monturas | 5 |
| 1.2.4. Rueda portafiltro | 6 |
| 1.2.5. Enfocadores | 6 |
| 1.2.6. Cúpulas | 6 |
| 1.2.7. Ópticas adaptativas | 6 |
| 1.2.8. Estaciones meteorológicas | 7 |
| 1.3. Control de dispositivos astronómicos | 8 |
| 1.4. INDI | 9 |
| 1.4.1. INDI for Java | 9 |
| 1.5. Dispositivos Móviles | 10 |
| 1.5.1. iOS | 11 |
| 1.5.2. Android | 12 |
| 2. Objetivos | 15 |
| 2.1. Alcance de los objetivos | 17 |
| 2.2. Interdependencia de los objetivos | 17 |
| 3. Planificación | 19 |
| 3.1. Fases y entregas | 19 |
| 3.1.1. Fases | 19 |
| 3.1.2. Lista de entregas | 19 |
| 3.2. Estructura de descomposición del trabajo | 20 |
| 3.3. Lista de actividades | 21 |
| 3.4. Recursos humanos | 22 |
| 3.5. Presupuesto | 22 |
| 3.6. Temporización | 22 |

| | |
|--|-----------|
| 4. Análisis | 27 |
| 4.1. Análisis de requisitos | 27 |
| 4.1.1. Descripción de los actores | 27 |
| 4.1.2. Requisitos funcionales | 28 |
| 4.1.3. Requisitos no funcionales | 29 |
| 4.1.4. Requisitos de información | 29 |
| 4.2. Modelos de casos de uso | 30 |
| 4.2.1. Descripción básica de actores | 30 |
| 4.2.2. Descripción casos de uso | 30 |
| 4.3. Diagrama de paquetes | 40 |
| 4.4. Diagramas de casos de uso | 40 |
| 4.5. Diagramas de actividad | 44 |
| 4.6. Diagrama conceptual | 53 |
| 4.7. Otros diagramas o interfaces | 53 |
| 5. Diseño | 57 |
| 5.1. Diseño general del portal | 57 |
| 5.2. Diseño de una aplicación con desarrollo colaborativo | 59 |
| 5.3. Diseño de los tests unitarios y test de cobertura | 60 |
| 5.4. Diseño de la integración continua | 61 |
| 5.5. Diseño del despliegue automático | 62 |
| 5.6. Diseño del provisionamiento | 63 |
| 6. Implementación | 65 |
| 6.1. Uso de JSON como origen de datos | 66 |
| 6.2. Tests unitarios y de cobertura | 72 |
| 6.3. Integración continua | 74 |
| 6.4. Despliegue automático | 75 |
| 6.5. Provisionamiento | 78 |
| 6.6. Repositorio con el desarrollo | 80 |
| 7. Pruebas | 81 |
| 7.1. Pruebas unitarias | 81 |
| 7.2. Prueba de cobertura | 84 |
| 7.3. Integración continua | 86 |
| 7.4. Despliegue automático | 87 |
| 7.5. Provisionamiento | 89 |
| 7.6. Prueba de carga | 93 |
| 7.6.1. Métricas y parámetros que afectan al rendimiento | 94 |
| 7.6.2. Técnicas de evaluación, carga de trabajo y diseño de experimentos | 95 |
| 7.6.3. Presentación de los resultados | 96 |
| 7.6.4. Análisis e interpretación de los resultados | 100 |
| 7.6.5. Conclusiones sobre los resultados | 102 |

| | |
|--|------------|
| 7.7. Acceso web | 103 |
| 8. Conclusiones y trabajos futuros | 105 |
| Glosario de términos | 107 |
| Anexo. Realización del trabajo de fin de grado con licencia libre y formato abierto | 109 |
| Bibliografía | 113 |

Índice de figuras

| | |
|---|----|
| 1.1. ASCOM Standard (http://ascom-standards.org/) | 8 |
| 1.2. INDI Client (http://www.indilib.org/) | 10 |
| 1.3. INDI Server (http://www.indilib.org/) | 10 |
| 3.1. Diagrama de estructura de descomposición de trabajo | 23 |
| 3.2. Temporización de las tareas | 24 |
| 3.3. Diagrama de Gantt | 25 |
| 4.1. Diagrama de paquetes | 40 |
| 4.2. Diagrama de casos de uso: paquete Administración portal . . . | 41 |
| 4.3. Diagrama de casos de uso: paquete Acceso información | 42 |
| 4.4. Diagrama de casos de uso: paquete Pruebas de software | 43 |
| 4.5. Diagrama de casos de uso: paquete Configuración automática . . | 44 |
| 4.6. Diagrama de actividad CU-1. Inicio automático del servidor del portal | 45 |
| 4.7. Diagrama de actividad CU-2. Consultar información de Ad- ministración | 46 |
| 4.8. Diagrama de actividad CU-3. Consultar información de Do- cencia | 47 |
| 4.9. Diagrama de actividad CU-4. Consultar información de Ges- tión e Investigación | 48 |
| 4.10. Diagrama de actividad CU-5. Consultar información de Nor- mativa Legal | 49 |
| 4.11. Diagrama de actividad CU-6. Realizar tests unitarios | 50 |
| 4.12. Diagrama de actividad CU-7. Realizar test de cobertura | 51 |
| 4.13. Diagrama de actividad CU-8. Usar integración continua | 52 |
| 4.14. Diagrama de actividad CU-9. Usar despliegue automático . . . | 53 |
| 4.15. Diagrama de actividad CU-10. Usar provisionamiento | 54 |
| 4.16. Diagrama conceptual | 55 |
| 6.1. Activación de la integración continua | 74 |
| 6.2. Disparador de integración continua | 75 |
| 7.1. Ejecución de los test unitarios con Mocha (1/2) | 82 |

| | |
|--|-----|
| 7.2. Ejecución de los test unitarios con Mocha (2/2) | 83 |
| 7.3. Error en tests unitarios | 84 |
| 7.4. Resumen de resultados de test de cobertura | 85 |
| 7.5. Resultado de cobertura de la aplicación principal | 86 |
| 7.6. Resultado de cobertura de un módulo desarrollado | 87 |
| 7.7. Build en Travis CI generado con éxito | 88 |
| 7.8. Salida del build en Travis CI (despliegue) | 89 |
| 7.9. Salida del build en Travis CI (resultados tests) | 90 |
| 7.10. Ejecución de despliegue automático | 91 |
| 7.11. Prueba de provisionamiento con Ansible | 93 |
| 7.12. Gráfico de tiempo de ejecución | 97 |
| 7.13. Gráfico de solicitudes por segundo | 98 |
| 7.14. Gráfico de tiempo por solicitud concurrente | 99 |
| 7.15. Gráfico de velocidad de transferencia | 100 |
| 7.16. Visualización del portal de transparencia | 103 |

Índice de tablas

| | |
|---|----|
| 4.1. Curso normal de CU-1. Inicio automático del servidor del portal | 31 |
| 4.2. Curso alterno de CU-1. Inicio automático del servidor del portal | 31 |
| 4.3. Curso normal de CU-2. Consultar información de Administración | 32 |
| 4.4. Curso normal de CU-3. Consultar información de Docencia . | 33 |
| 4.5. Curso normal de CU-4. Consultar información de Gestión e Investigación | 34 |
| 4.6. Curso normal de CU-5. Consultar información de Normativa Legal | 35 |
| 4.7. Curso normal de CU-6. Realizar tests unitarios | 35 |
| 4.8. Curso alterno de CU-6. Realizar tests unitarios | 36 |
| 4.9. Curso normal de CU-7. Realizar test de cobertura | 36 |
| 4.10. Curso alterno de CU-7. Realizar test de cobertura | 36 |
| 4.11. Curso normal de CU-8. Usar integración continua | 37 |
| 4.12. Curso alterno de CU-8. Usar integración continua | 37 |
| 4.13. Curso normal de CU-9. Usar despliegue automático | 38 |
| 4.14. Curso alterno de CU-9. Usar despliegue automático | 38 |
| 4.15. Curso normal de CU-10. Usar provisionamiento | 39 |
| 4.16. Curso alterno de CU-10. Usar provisionamiento | 40 |
| 7.1. Resultados de tiempo de ejecución | 96 |
| 7.2. Resultados de solicitudes por segundo | 97 |
| 7.3. Resultados de tiempo por solicitud concurrente | 98 |
| 7.4. Resultados de velocidad de transferencia | 99 |

Índice de fragmentos de código

| | |
|---|----|
| 6.1. Archivo JSON con información de personal | 68 |
| 6.2. Módulo para cargar archivos JSON | 69 |
| 6.3. Módulo principal de la aplicación | 70 |
| 6.4. Archivo administracion.js | 71 |
| 6.5. Scripts de inicio y detención | 72 |
| 6.6. Módulo de tests unitarios | 73 |
| 6.7. Scripts de test | 74 |
| 6.8. Archivo de configuración de Travis CI | 75 |
| 6.9. Módulo de despliegue automático | 77 |
| 6.10. Scripts de despliegue automático | 77 |
| 6.11. Archivo de hosts de Ansible | 78 |
| 6.12. Playbook de Ansible | 79 |
| 6.13. Línea de comando de Ansible | 80 |
| 7.1. Órdenes para crear máquina virtual Vagrant | 89 |
| 7.2. Archivo Vagrantfile | 91 |
| 7.3. Órdenes para provisionar máquina Vagrant | 92 |

Capítulo 1

Introducción

1.1. La Astronomía

Desde el principio de los tiempos, el ser humano a mirado al cielo con incertidumbre, viendolo como una fuente inagotable de interrogantes sin resolver. En casi todas las religiones antiguas existía la “**cosmogonía**” que intentaba explicar el origen del universo, ligando este a los elementos mitológicos, dando paso esta a la “**astronomía**”:

“Ciencia que se ocupa del estudio de los cuerpos celestes del universo, incluidos los planetas y sus satélites, los cometas y meteoritos, las estrellas y la materia interestelar, los sistemas de materia oscura, estrellas, gas y polvo llamados galaxias y los cúmulos de galaxias; por lo que estudia sus movimientos y fenómenos ligados a ellos.”

(<https://es.wikipedia.org/wiki/Astronom%C3%ADa>)

La Astronomía es probablemente la más antigua de las ciencias naturales originándose en la antigüedad en casi todas las culturas humanas. Sus orígenes se pierden en prácticas religiosas de la prehistoria cuyos vestigios se encuentran en numerosos sitios arqueológicos (como Stonehenge) e incorporados todavía en la astrología una disciplina entrelazada con la astronomía y no separada de ella completamente hasta el siglo XVIII en el mundo occidental. La astronomía antigua constituyó las bases del calendario y la medida de periodos temporales como la semana el mes o el año. Los astrónomos antiguos eran capaces de distinguir entre estrellas y planetas dado que las primeras permanecen fijas en sus posiciones relativas mientras que los planetas se mueven una cantidad apreciable de espacio a lo largo de periodos relativamente cortos (Saturno el más lento de los planetas conocidos en la antigüedad describe un periodo orbital en 29 años). La Astronomía antigua culmina con el desarrollo ordenado del modelo heliocéntrico expuesto

en las obras de *Ptolomeo*. Previamente *Aristarco de Samos* había medido las distancias de la Tierra a la Luna y al Sol afirmando como consecuencia de éstas que el Sol era el centro del Universo alrededor del cual giraban los demás planetas incluyendo la Tierra. Otros logros destacados de la época clásica de la astronomía fueron los conseguidos por *Hiparco* quien realizó el primer catálogo estelar y propuso un sistema de clasificación estelar en 6 magnitudes basado en la luminosidad aparente de las diferentes estrellas. La Astronomía en la Europa medieval se produce un oscurantismo en todos los campos del conocimiento incluyendo la astronomía. Ésta permanece preservada en escasas copias de tratados antiguos de la astronomía griega y romana. La astronomía observacional tan sólo se conserva en el mundo árabe.

Tycho Brahe (1546-1601) introdujo la idea de la precisión de la medida en astronomía e inventó y produjo una gran cantidad de instrumental astronómico previo al telescopio. *Galileo Galilei* (1564-1642) construyó su propio telescopio a partir de un invento holandés y lo utilizó inmediatamente en el estudio astronómico descubriendo los cráteres de la Luna, las lunas de Júpiter y las manchas solares. Sus observaciones tan sólo eran compatibles con el modelo **copernicano**. Paralelamente *Johannes Kepler* expuso sus famosas **leyes de Kepler** para el movimiento de los planetas basando su trabajo en las detalladas observaciones de *Tycho Brahe*.

Una generación más tarde Isaac Newton fue el primer científico que unió la Física con la Astronomía proponiendo que las mismas fuerzas que hacían caer los cuerpos sobre la Tierra causaban el movimiento de los planetas y la Luna. Utilizando su Ley de la gravedad las leyes de Kepler resultan inmediatamente explicadas. Newton también descubrió que la Luz blanca del Sol está descompuesta en diferentes colors, un hecho importantísimo para el futuro desarrollo de la astronomía.

La **astronomía** es una de las pocas ciencias en las que los aficionados aún pueden desempeñar un papel activo, especialmente en el descubrimientos y seguimiento de fenómenos. Es por ello que existe una gran variedad de **herramientas** e **instrumental astronómico** que permiten a cualquier persona observar el universo.

1.2. Instrumental Astronómico

Existe una gran variedad de **instrumental astronómico** en la actualidad. A continuación se describen las familias más importantes.

1.2.1. Telescopios

“El telescopio es un instrumento óptico que permite ver objetos lejanos con mucho más detalle que a simple vista al captar radiación electromagnética, tal como la luz.”

(<https://es.wikipedia.org/wiki/Telescopio>)

Hace cuatro siglos nació un invento que habría de redefinir nuestro lugar en el universo. Tachado en su momento como el instrumento más diabólico de la historia, el telescopio sacudió la sociedad hasta las raíces. Al alzar los ojos al cielo nos convencimos de que éramos el centro de la creación, y había razones para ello: desde nuestra perspectiva, todo parece girar en torno a la Tierra

Los fabricantes de vidrio sabían desde la antigüedad que una esfera de vidrio podía aumentar imágenes, pero tuvieron que pasar siglos antes de que alguien ensamblara dos lentes en un tubo y mirara a través de ellas. Señalar la fecha, lugar y autor exactos de su invención es controvertido. Los holandeses se inclinan por el 2 de octubre de 1608, el día en que *Hans Lippershey* patentó un instrumento llamado **kijker**, que significa mirador. Un moledor de vidrio holandés aseguraba haber inventado un aparato similar, pero el primero en patentarlo fue *Lippershey*. Como era alemán, vivía en Holanda y registró la patente en Bélgica, más de un país ha pugnado por el honor de su autoría. Sin embargo, como dijo *Darwin*:

“en la ciencia el crédito es del que convence al mundo y no del primero en tener la idea”

(Charles Darwin)

Por eso la gloria se la llevó Italia, ya que fueron las mejoras que introdujo *Galileo* las que permitieron usar el aparato como instrumento astronómico. El diseño de *Galileo* consistía en una lente convexa para el objetivo y otra cóncava en el ocular. En 1611 el alemán *Johannes Kepler* fue el primero en usar dos lentes convexas que enfocaban los rayos en un mismo punto. La configuración de *Kepler* aún se usa en binoculares y cámaras fotográficas modernas y es la base del telescopio refractor.

Tras la muerte de *Galileo*, fue *Isaac Newton* quien nos dio una nueva imagen del universo que sobrevivió 250 años hasta la llegada de *Albert Einstein*.

“Si he logrado ver más lejos ha sido porque me he subido a hombros de gigantes”

(Isaac Newton)

Y así, sobre la herencia de *Galileo*, *Newton* inventó el **telescopio reflector**, que es la base de los actuales. La innovación consistía en usar espejos en lugar de lentes para enfocar la luz y formar imágenes. Entonces el universo se nos abrió en todo su esplendor.

1.2.2. Cámaras CCD

Un dispositivo de carga acoplada (en inglés **Charge-Coupled Device**, conocido también como **CCD**), es un circuito integrado que contiene un número determinado de condensadores enlazados o acoplados. Bajo el control de un circuito interno, cada condensador puede transferir su carga eléctrica a uno o a varios de los condensadores que estén a su lado en el circuito impreso.

El **CCD** se inventó a finales de los 60 por investigadores de **Bell Laboratories**. Originalmente se concibió como un nuevo tipo de memoria de ordenador pero pronto se observó que tenía muchas más aplicaciones potenciales tales como el proceso de señales y sobre todo la captación de imagen, esto último debido a la sensibilidad a la luz que presenta el silicio.

El sensor **CCD** de una cámara digital es como el motor de un coche, es la pieza principal. En su forma más elemental, el **CCD** es como un ojo electrónico que recoge la luz y la convierte en una señal eléctrica. Tienen dos diferencias básicas con los fotomultiplicadores:

Los sensores **CCD** son de menor tamaño y están contruidos de semiconductores lo que permite la integración de millones de dispositivos sensibles en un solo chip. La eficiencia cuántica de los **CCD** (sensibilidad) es mayor para los rojos. Los fotomultiplicadores son más sensibles a los azules.

Físicamente, un **CCD** es una malla muy empaquetada de electrodos de polisilicio colocados sobre la superficie de un chip. Al impactar los fotones sobre el silicio se generan electrones generados que pueden guardarse temporalmente. Periódicamente se lee el contenido de cada pixel haciendo que los electrones se desplacen físicamente desde la posición donde se originaron (en la superficie del chip), hacia el amplificador de señal con lo que se genera una corriente eléctrica que será proporcional al número de fotones que llegaron al pixel. Para coordinar los periodos de almacenamiento (tiempo de exposición) y vaciado del pixel (lectura del pixel) debe existir una fuente eléctrica externa que marque el ritmo de almacenamiento-lectura: el reloj

del sistema. La forma y amplitud de reloj son críticas en la operación de lectura del contenido de los píxeles.

Al tratarse el **CCD** de un dispositivo semiconductor, técnicamente es posible implementar en él todas las funciones electrónicas de un sistema de captación de imagen, pero esto no es rentable económicamente y por tanto se implementa en otros chips externos al **CCD**: la mayoría de **CCD** de cámaras tienen varios chips (de tres a ocho).

La necesidad de usar chips distintos implica dos desventajas importantes; la necesidad de voltajes múltiples de abastecimiento de los chips y un gran consumo de potencia de todo el sistema electrónico.

1.2.3. Monturas

La montura de un telescopio es la parte mecánica que une el trípode o base al instrumento óptico. Existen varios tipos de monturas, algunas muy simples, otras mas complejas, incluso con correctores electrónicos y dispositivos de localización y seguimiento muy sofisticados (sistemas **GOTO**)

La montura tiene como objetivo proveer de movimiento controlado al telescopio. Es muy importante la firmeza y suavidad de los movimientos, para que la observación sea confortable y las astrofotografías perfectas. Las monturas se clasifican en dos grandes grupos, según los planos de referencia que utilicen (coordenadas).

La más simple es la montura altacimutal, que realiza movimientos horizontales y verticales (acimut y altura, respectivamente). Este tipo de diseño lo traen incorporados los telescopios pequeños, por lo general telescopios refractores de uso terrestre, dado que su uso es simple, y también varios modelos de equipos automatizados (sistemas **GOTO**)

Le sigue la montura ecuatorial, que utiliza como plano fundamental el ecuador celeste (proyección del ecuador terrestre). Este diseño usa las coordenadas ecuatoriales, ascensión recta (A.R. o R.A.) y declinación (Dec.), que son proyecciones de las coordenadas terrestres longitud y latitud, respectivamente, sobre la esfera celeste.

Existen varios tipos de monturas basados en los dos diseños fundamentales anteriores. La montura **Dobson** por ejemplo (suelen llamarse telescopios *dobsonianos* a los que la poseen), es un modelo basado en la altacimutal, sin trípode y un telescopio de diseño newtoniano como instrumento de observación. Es muy utilizado por los que desean una gran apertura en reflectores, por ejemplo los que se construyen su propio espejo y no quieren tener grandes gastos en monturas sofisticadas.

1.2.4. Rueda portafiltro

La rueda porta-filtros consiste en un cuerpo, generalmente de aluminio, que en su interior puede alojar varios filtros, normalmente de 1,25" de diámetro. Lo aconsejable es que tenga, al menos, 4 huecos para filtros si queremos hacer astrofotografía con cámaras CCD blanco y negro, puesto que vamos a necesitar el azul, rojo y verde (RGB) y, posiblemente, un filtro para infrarrojos.

1.2.5. Enfocadores

El **enfocador** es una pieza fundamental del telescopio. Nos permitirá ver las imágenes formadas tras la reflexión de la luz en el espejo primario y su desviación por el espejo secundario. Para verlas necesitaremos un juego de oculares. La longitud focal de los oculares combinada con la longitud focal de nuestro telescopio nos dará el número de aumentos total del sistema. Dichos oculares están montados en el **enfocador**, un dispositivo móvil que permitirá mover la posición vertical del ocular para enfocar adecuadamente la imagen.

Un ejemplo de enfocador son los de tipo **Crayford** y los **rack and pinion**.

1.2.6. Cúpulas

Las **cúpulas** son recintos cerrados mas o menos grandes que nos permiten albergar y proteger el instrumental astronómico. De esta forma, las **cúpulas** pueden ser abiertas o cerradas para exponer los instrumentos en el momento de las observaciones.

1.2.7. Ópticas adaptativas

La **óptica adaptativa** es una técnica que permite corregir las perturbaciones más importantes que sufren las imágenes astronómicas debido a la atmósfera terrestre. Con este sistema es posible obtener imágenes más nítidas o de mejor resolución espacial. La diferencia que introduce esta técnica es comparable a la que existe entre mirar un objeto situado en el fondo de una piscina con agua o sin agua.

Las posibilidades que la óptica adaptativa ofrece a la astronomía son espectaculares. Eliminar las perturbaciones producidas por la atmósfera equivale esencialmente a observar desde el espacio. Las perturbaciones atmosféricas causan una pérdida en nitidez o resolución espacial. Esta pérdida se traduce, por un lado, en una disminuida capacidad para resolver objetos, es decir, para realizar estudios detallados de su morfología. Por otro lado,

influye también en la capacidad de detectar objetos débiles, dado que la imagen se dispersa en puntos de luz mayores.

La mejora que introduce la óptica adaptativa se puede cuantificar utilizando la relación entre el tamaño del telescopio y el tamaño de la mejor imagen que puede obtener. El poder de detección de un telescopio aumenta con el diámetro de su espejo primario y disminuye con el tamaño de la imagen que forma de un objeto puntual (de aquí la importancia de la calidad de imagen en un telescopio). Por tanto, la diferencia con un mismo espejo de 10 metros, entre conseguir enfocar imágenes de 0.4 segundos de arco (lo posible en una noche de visibilidad excelente) y una imagen de 0.04 segundos de arco, que debe ser posible con un sistema de óptica adaptativa, equivaldría a tener un espejo primario de 100 metros

1.2.8. Estaciones meteorológicas

Las **estaciones meteorológicas** son sistemas compuestos por un “*data logger*” y un conjunto de sensores que nos proporcionan datos de las distintas magnitudes meteorológicas, tales como la temperatura, humedad, presión barométrica, etc... permitiéndonos generar modelos a partir de los cuales conocer la situación climática y su posible evolución.

Gracias a los datos aportados por las **estaciones meteorológicas**, podemos conocer la climatología en el momento de realizar observaciones astronómicas. De esta forma podemos decidir si las condiciones son óptimas, o incluso decidir si debemos cerrar la cúpula para evitar daños en los instrumentos por lluvias o similar.

1.3. Control de dispositivos astronómicos

Actualmente existen diversas formas de controlar los dispositivos astronómicos pero la mayoría presenta los mismos inconvenientes:

- Normalmente se controlan los dispositivos directamente.
- Se conecta el dispositivo a un PC y se trabaja desde él.
- Se utilizan herramientas para el control remoto como el “escritorio remoto”.

Por otro lado, existen estándares como el de **ASCOM** para instrumental astronómico. Con él, se intenta crear una capa entre los programas para controlar dispositivos astronómicos y los propios dispositivos. La principal desventaja es que solo puede utilizarse en sistemas *Microsoft Windows*

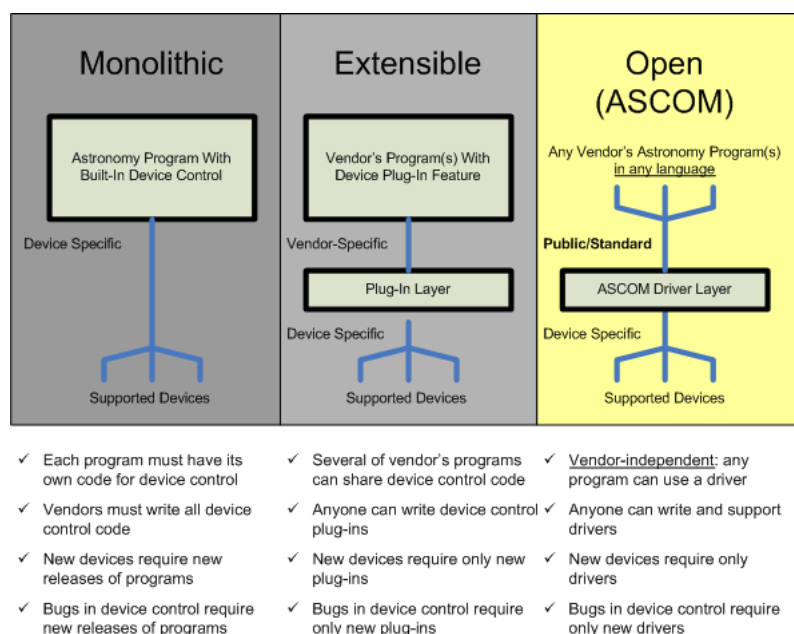


Figura 1.1: ASCOM Standard (<http://ascom-standards.org/>)

1.4. INDI

“The Instrument Neutral Distributed Interface (INDI) Library is a cross-platform software designed for automation control of astronomical instruments. It supports a wide variety of telescopes, CCDs, focusers, filter wheels..etc, and it has the capability to support virtually any device. INDI is small, flexible, easy to parse, and scalable. It supports common DCS functions such as remote control, data acquisition, monitoring, and a lot more. With INDI, you have a total transparent control over your instruments so you can get more science with less time.”
(<http://indilib.org/about.html>)

El protocolo **INDI** es una plataforma software diseñada para el control de instrumental astronómico. La biblioteca **INDI** permite controlar cualquier dispositivo con un driver **INDI** mediante el paso de archivos XML. Sus principales ventajas frente a otras soluciones para el control de dispositivos son:

- Es una biblioteca ligera, flexible y escalable.
- Es de código abierto por lo que cualquiera puede ver su código y mejorarlo o crear drivers para cualquier dispositivo
- El intercambio de información es mínimo.
- Es multiplataforma.
- Separa el cliente del servidor.
- Los fabricantes comienzan a desarrollar drivers para sus dispositivos o liberan las especificaciones para que la comunidad pueda desarrollarlos.
- Existen numerosos clientes INDI como <https://edu.kde.org/kstars/>

1.4.1. INDI for Java

La biblioteca **INDI** está escrita en lenguaje **C**, pero existe una implementación realizada en Java y que se encuentra en desarrollo. En la página oficial de **INDI** <http://indilib.org/develop/indiforjava.html> podemos encontrar toda la información sobre nuevas versiones y la documentación para poder utilizarla. La principal ventaja de poder usar Java es que podemos implementar drivers y clientes con la potencia de un lenguaje Orientado a Objetos y combinarlo con otras tecnologías como los dispositivos móviles basados en la plataforma **Android**

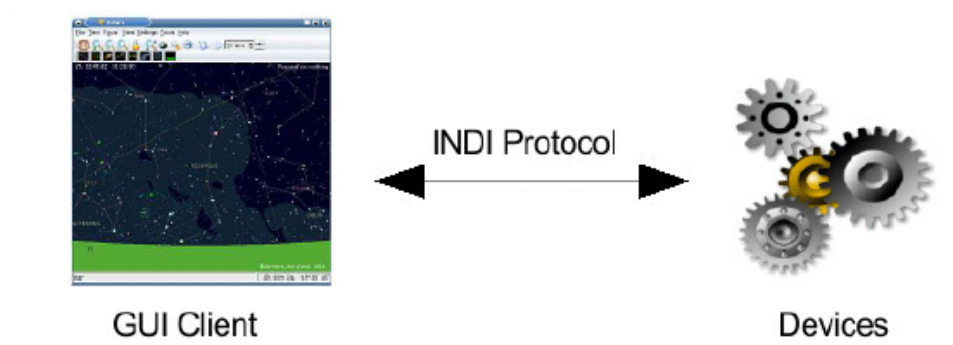


Figura 1.2: INDI Client (<http://www.indilib.org/>)

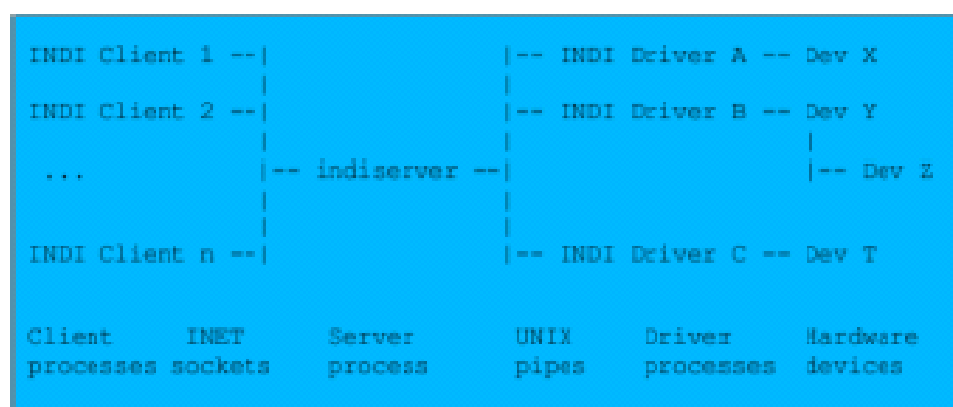


Figura 1.3: INDI Server (<http://www.indilib.org/>)

1.5. Dispositivos Móviles

Un **dispositivo móvil** es un tipo de computadora de tamaño pequeño, con capacidad de procesamiento, con conexión a internet, con memoria, diseñados específicamente para una función pero que pueden llevar a cabo otras funciones más generales.

Los **dispositivos móviles** hoy en día están integrados en la mayoría de tareas cotidianas de una persona. La tendencia de la sociedad actual nos empuja hacia un mundo cada vez más móvil donde necesitamos estar conectado e interactuar con otros sistemas. Es por ello que la mayoría de soluciones tecnológicas, hayan sido pensadas o no para el sector de los dispositivos móviles, siempre acaba teniendo una versión para éstos.

Paralelamente a la expansión de los **dispositivos móviles**, se han creado un gran número de sistemas operativos para estos dispositivos entre los

que se encuentra:

- Android.
- iOS.
- BlackBerry OS.
- Palm OS.
- Windows Mobile/Phone.
- Symbian.

Actualmente **Android** y **iOS** copan el 96.3 % del mercado¹. Por lo que nos centraremos principalmente en estos S.O.² y los **dispositivos móviles** compatibles con ellos.

1.5.1. iOS

iOS es un S.O. móvil de la compañía *Apple Inc* originalmente desarrollado para el *iPhone*³ y posteriormente introducido en otros **dispositivos móviles** de la compañía como el iPod touch⁴ y el iPad⁵. **iOS** no puede ser instalado en hardware de terceros.

Actualmente tiene una cuota de mercado aproximadamente del 19.7 %, siendo el segundo S.O. más utilizado.

iOS es un sistema muy estable, diseñado para un hardware muy concreto y por tanto, muy eficiente y depurado. Pero de cara a elegirlo como una opción a la hora de desarrollar una nueva aplicación para **dispositivos móviles** se debe tener en cuenta los siguientes aspectos:

- Hay que pagar una cuota anual de 99\$ para poder publicar aplicaciones en el *Apple Store*⁶. Además si esta licencia, no podremos desarrollar aplicación y cargarla en nuestros dispositivos Apple.
- Necesitamos un MAC⁷ ya que las herramientas para el desarrollador solo pueden utilizarse en sus equipos.

¹Fuente:<http://www.idc.com/getdoc.jsp?containerId=prUS25450615>.

²Sistema Operativo.

³Smartphone de la compañía Apple Inc.

⁴Dispositivo móvil para reproducir multimedia de la compañía Apple Inc

⁵Tablet de la compañía Apple Inc.

⁶Tienda de aplicaciones de la compañía Apple Inc.

⁷Computadoras personales de la compañía Apple Inc.

- Necesitaremos conocer el lenguaje de programación **Objective-C**
- **iOS** es un sistema de código cerrado que va en contra de la filosofía del **Software Libre** y el código abierto y reutilizable.

Aunque **iOS** es un sistema muy extendido y con un gran número de usuarios, creemos que no es la mejor opción para orientar una aplicación móvil basada en **Software Libre** además de la inversión anual requerida para poder publicar una aplicación que pretendamos sea gratuita, libre y accesible a cualquier usuario.

1.5.2. Android

Android es un Sistema Operativo basado en un **núcleo Linux**⁸. Fue diseñado principalmente para dispositivos móviles con pantalla táctil, relojes inteligentes, televisiones inteligentes y automóviles. Inicialmente pertenecía a la compañía **Android Inc.** que posteriormente sería adquirida por **Google**. Actualmente posee la mayor cuota de mercado de aproximadamente el 76.6 %.

Los principales componentes del sistema operativo **Android** son:

- **Aplicaciones:** Todas las aplicaciones están escritas en lenguaje de programación **Java**.
- **Framework**⁹: Los desarrolladores tienen acceso completo a las mismas API's¹⁰ que utiliza el sistema. La arquitectura está diseñada para simplificar la reutilización de componentes.
- **Runtime de Android:** Android incluye un set de bibliotecas base que proporcionan la mayor parte de las funciones disponibles en las bibliotecas base del lenguaje **Java**. Cada aplicación android corre su propio proceso, con su propia instancia de la máquina virtual *Dalvik*¹¹
- **Núcleo Linux:** Android depende de Linux para los servicios base del sistema como la seguridad, la gestión de memoria, la gestión de procesos, etc. El núcleo Linux también sirve como capa de abstracción entre el hardware y el software.

⁸Sistema operativo basado en Unix

⁹Marco de trabajo para los desarrolladores

¹⁰Interfaz de programación de aplicaciones (Application Programming Interface)

¹¹Máquina virtual que utiliza la plataforma Android para ejecutar aplicaciones Java.

Android no tiene restricciones de uso por lo que puede utilizarse en número muy extenso de dispositivos móviles. Además es un sistema parcialmente de código abierto. Está basado en Linux y la mayoría del código es abierto aunque no todo el sistema lo es.

De cara al desarrollo de aplicaciones móviles, **Android** es una opción muy recomendable por las siguientes razones:

- La arquitectura del sistema (basada en Linux, lenguaje de programación Java,...)
- La mayoría de los dispositivos móviles del mundo tienen como sistema operativo a **Android** por lo que la difusión será mayor que con otros sistemas.
- Las herramientas para desarrollar en Android son multiplataforma y gratuitas. Para poder crear y probar una aplicación solo necesitas un ordenador con cualquier sistema operativo, un dispositivo móvil con android y descargar la herramienta para desarrolladores.
- Para poder publicar aplicaciones en *Google Play*¹² hay que pagar 25\$ sin tener renovarlo anualmente y sin ninguna limitación.

¹²Tienda de aplicaciones para dispositivos Android

Capítulo 2

Objetivos

El objetivo de este proyecto es desarrollar una aplicación para la plataforma **Android** que implemente un cliente utilizando la biblioteca “**INDI for Java**” basado en el **Software Libre** y que sea fácilmente extensible.

A continuación se describen los objetivos principales a alcanzar:

- **OBJ-1.** Conseguir un cliente funcional capaz de controlar cualquier dispositivo **INDI**.
- **OBJ-2.** Poder gestionar múltiples conexiones con múltiples dispositivos simultáneamente.
- **OBJ-3.** Facilmente extensible, permitiendo añadir vistas para propiedades y dispositivos por parte de desarrolladores ajenos al proyecto.
- **OBJ-4.** Desarrollar la aplicación bajo una licencia de código abierto fomentando la filosofía del **Software Libre** y la publicación de todo el código.

Además de los objetivos principales, se persigue alcanzar los siguientes objetivos:

- **OBJ-S-1.** Desarrollar la aplicación siguiendo los estándares actuales y las recomendaciones para la plataforma **Android**.
- **OBJ-S-2.** Facilitar la usabilidad mediante un diseño adecuado de las interfaces, adaptándola a los distintos tamaños de pantalla y personalizándolas a las propiedades estándares de **INDI**.
- **OBJ-S-3.** Desarrollar para incluir un correcto funcionamiento en el mayor número posible de versiones de **Android**, maximizando el número de dispositivos compatibles.

- **OBJ-S-4.** Añadir una versión estable en *Google Play* y publicar el *APK*¹ para poder descargarlo a través de internet.

Para la realización de los objetivos se pondrán en practica los conocimientos alcanzados en;

- **Ingeniería del software** para el análisis del proyecto.
- **Programación orientada a objetos** para la estructura y la organización del código **Java**.
- **Programación concurrente y sistemas operativos** para la gestión de las distintas hebras y la comunicación entre ellas.
- **Programación de sistemas multimedia** para poder implementar las interfaces de usuario en **Android** y poder tratar y mostrar imágenes enviadas por los dispositivos.
- **Infraestructura virtual** para poder gestionar los sistemas para realización de test y simulaciones.
- **Transmisión de datos y redes de computadores** para comprender el comportamiento del protocolo **INDI** y configurar correctamente las redes para las pruebas.

Por otro lado, han sido necesarios alcanzar conocimientos en otras áreas:

- **Astronomía y equipos astronómicos** para entender a los usuarios potenciales y poder acomodar la aplicación a sus necesidades.
- **Android** para conocer las herramientas que ofrece la plataforma y usar las mas adecuadas según las necesidades concretas.
- **Raspberry Pi**² para montar un servidor permanente de pruebas o acceso público para probar la aplicación
- **Latex**³ para la realización del presente documento y la ampliación de conocimientos para futuros textos científicos.
- **Git** para la gestión de versiones y la publicación de código abierto que permita a otros desarrolladores participar.

¹Paquete para el sistema operativo Android (Application Package File)

²Ordenador de placa reducida y única de bajo coste.

³Sistema de composición de textos.

2.1. Alcance de los objetivos

La aplicación móvil desarrollada debe cumplir los objetivos principales para cubrir una necesidad existente. Actualmente no existe ninguna aplicación móvil basada en **INDI** para controlar dispositivos astronómicos. Con la realización del proyecto se pretende cubrir dicha necesidad, obteniendo una aplicación estable y que será mantenida y mejorada más allá de la finalización del Proyecto Fin de Grado. Se trata de un proyecto vivo y extensible en el tiempo.

La consecución de alcanzar también los objetivos secundarios tendrá un efecto directo en la difusión de la aplicación y en la satisfacción directa de los usuarios de la misma. Por ello, se comprará una licencia de desarrollador para *Google Play* y se publicará y dará difusión en distintos canales de comunicación como la página oficial **INDI** y a través de foros y páginas web.

2.2. Interdependencia de los objetivos

El principal objetivo que debe cumplir la aplicación es el *OBJ-1*, aunque todos los objetivos son independientes excepto los objetivos secundarios *OBJ-S-1*, *OBJ-S-2* y *OBJ-S-3*. Seguir los estándares y recomendaciones de la plataforma **Android** derivará en una mayor compatibilidad con versiones antiguas del sistema operativo y un diseño de la interfaz de usuario más amigable y fácil de usar.

Capítulo 3

Planificación

3.1. Fases y entregas

3.1.1. Fases

Como este va a ser un proyecto que ya cuenta con un trabajo previo realizado, la fase inicial de gestión va a ser muy breve porque se parte de que ya se han realizado varias reuniones y el proyecto está parcialmente funcionando, por lo que no se parte de cero, es una ampliación de un proyecto inicial.

- **Fase 1:** Especificaciones del proyecto.
- **Fase 2:** Planificación.
- **Fase 3:** Análisis y diseño.
- **Fase 4:** Implementación.
- **Fase 5:** Pruebas.
- **Fase 6:** Documentación.

3.1.2. Lista de entregas

Se harán una serie de breves informes sobre el contenido de cada una de las fases de planificación del proyecto.

- **Fase 1:** Especificación del proyecto.
 - Descripción: Se establecen los objetivos a cumplir para que el desarrollo del proyecto se considere completado.
 - Tipo: informe.

- **Fase 2:** Planificación.
 - Descripción: Se desarrolla la documentación con toda la planificación del desarrollo del proyecto.
 - Tipo: informe.
- **Fase 3:** Análisis y diseño.
 - Descripción: Todos los aspectos del proyectos son analizados para concretar la forma de desarrollarlo.
 - Tipo: informe.
- **Fase 4:** Implementación.
 - Descripción: Con el proyecto ya planificado y diseña se pasa programar todo lo necesario para cumplir los objetivos.
 - Tipo: software.
- **Fase 5:** Pruebas.
 - Descripción: Una vez este todo programado, se pasa a validar con diferentes procedimientos que el proyecto funciona correctamente tomando como referentes unas métricas propias.
 - Tipo: informe y software.
- **Fase 6:** Documentación.
 - Descripción: Para finalizar el proyecto se realiza toda la documentación informativa y explicativa.
 - Tipo: informe.

3.2. Estructura de descomposición del trabajo

El diagrama de estructura de descomposición del trabajo (*figura 3.1*) es una descomposición jerárquica de las diferentes fases y entregas en las que está planificado el proyecto.

3.3. Lista de actividades

Las actividades que se vayan a desarrollar en cada una de las fases para cada una de las entregas se va a listar junto con una estimación del tiempo que deberían tomar en ser cumplidas.

- **Especificaciones del proyecto:**

- Determinación de objetivos.
- Determinación de requisitos.
- *Estimación:* 9 horas.

- **Planificación:**

- Lista de actividades.
- Recursos humanos.
- Presupuesto.
- Temporización.
- *Estimación:* 18 horas.

- **Análisis y diseño:**

- Análisis de requisitos.
- Diagramas.
- Metodología de desarrollo.
- Descripción estructural.
- *Estimación:* 36 horas.

- **Implementación:**

- Herramientas seleccionadas.
- Solucionar problema de generación de páginas.
- Implementar tests unitarios.
- Implementar test de cobertura.
- Introducir integración continua.
- Agregar despliegue automático.
- Actualizar provisionamiento.
- *Estimación:* 90 horas.

- **Pruebas:**

- Pruebas de software.
- Pruebas de carga.
- Estimación: 30 horas.

- **Documentación:**

- Documentación de la aplicación.
- Manual de usuario.
- Documentación del proyecto.
- Estimación: 30 horas.

3.4. Recursos humanos

Como este es un proyecto que comenzó su desarrollo en la **Oficina de Software Libre de la Universidad de Granada**, cuento con el apoyo de todos sus colaboradores, además del resto de becarios que se encuentran realizando prácticas en empresa en ella como es mi situación actual, además, siendo el tutor de este proyecto el director de la propia oficina.

3.5. Presupuesto

Una de las ventajas de usar software libre es que no es necesario adquirir licencias por las que haya que pagar para realizar su uso, como todas las herramientas que usen (al igual que las que se generen) serán software libre, el coste en software para el desarrollo será cero.

El único recurso necesario es el servidor en que estará instalada el portal, servidor ya que adquirido anteriormente, por lo que tampoco es necesario considerarlo un gasto a afrontar de cara al desarrollo.

3.6. Temporización

Para percibir de forma más visual la planificación temporal de las tareas se incluyen la *figura 3.2* con una tabla indicando los plazos de cada tarea y la *figura 3.3* con un diagrama de *Gantt* de dichas tareas.



Figura 3.1: Diagrama de estructura de descomposición de trabajo

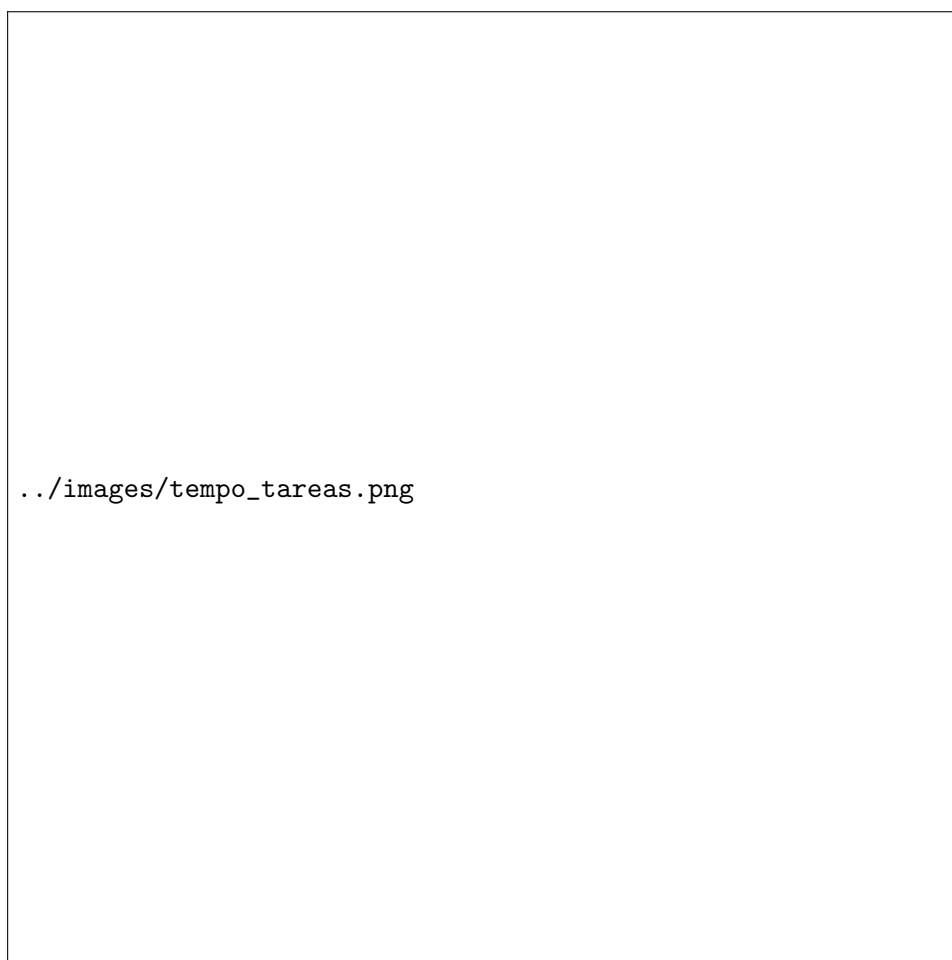
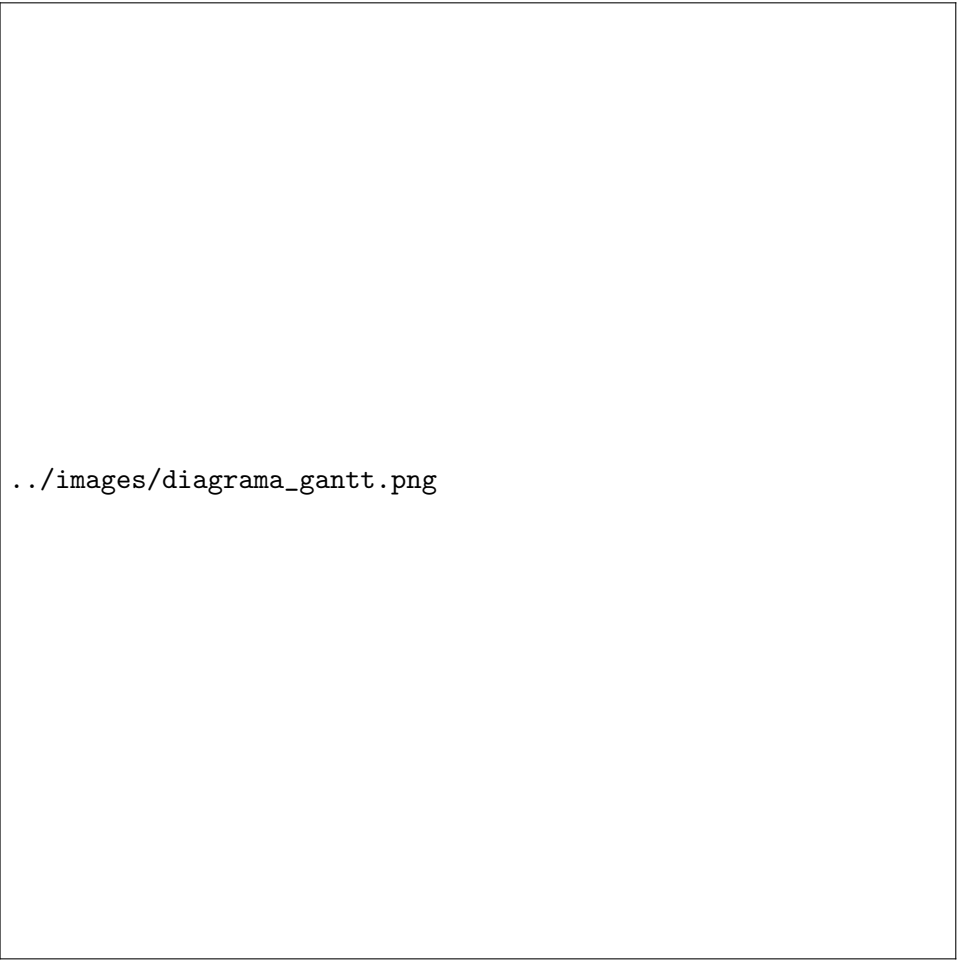


Figura 3.2: Temporización de las tareas



../images/diagrama_gantt.png

Figura 3.3: Diagrama de Gantt

Capítulo 4

Análisis

4.1. Análisis de requisitos

Todo software se desarrolla para cubrir una necesidad, por lo que en este apartado vamos a describir los requisitos que se estiman necesarios para cubrir los objetivos propuestos.

4.1.1. Descripción de los actores

Los actores implicados serán dos: el **desarrollador** y el **usuario**.

El **desarrollador** será el encargado de solucionar los problemas de visualización de los datos en el portal, además de portar el desarrollo actual del portal a un entorno de desarrollo continuo. También asumirá la administración del portal ya que este tipo de rol está muy integrado con las labores de despliegue en el desarrollo continuo.

El **usuario** de la aplicación será cualquier persona que tenga interés por conocer datos internos de la **Universidad de Granada** fácilmente. El usuario no pertenece a ningún público objetivo concreto, por lo que no se tiene que considerar que tenga una experiencia previa en navegador por sitios web.

Los dos tipos de actores descritos son la generalización de todo los tipos de personas que usarían el portal, sin embargo, podríamos hacer un análisis más extenso si nos basáramos en una metodología orientada al usuario como **persona**. Una persona sería la representación de usuarios que comparten objetivos y a los que se espera comportamientos similares. Un par de ejemplos:

- Alguien de la propia **Universidad de Granada** y de otra universidad que accedan al portal, ambos encajarían dentro del actor **usuario**, sin embargo eso no quiere decir que el objetivo de ambos sea el mismo, los usuarios de la propia universidad pueden querer consultar datos únicamente y los de otra universidad puede que estén más interesados en ver la estructura de las páginas del portal.
- Alguien que quiera hacer una despliegue de la aplicación, puede que no quiere hacerlo para una universidad sino para un ayuntamiento, volvemos a estar en el mismo caso, ambos entrarían en el tipo de actor **desarrollador**, pero sus intenciones e interés son muy distintos entre sí.

4.1.2. Requisitos funcionales

Los requisitos funcionales son las características que tiene que implementar el sistema para cubrir todas las necesidades de los distintos usuarios.

Al usuario lo único que le interesa es ver una página web estática con la información que desea consultar, para ello el desarrollador deberá hacer que sea posible que se generen siempre las tablas con los elementos de información.

Por otra parte, el desarrollador quiere integrar el sistema en un desarrollo continuo, por lo que añadirá tests unitarios, test de cobertura, integración continua, despliegue automático y provisionamiento con tal fin.

- **RF-1.** Administración portal:
 - **RF-1.1.** Automatizar inicio del servidor del portal.
- **RF-2.** Acceso información:
 - **RF-2.1.** Consultar información de *Administración*.
 - **RF-2.1.** Consultar información de *Docencia*.
 - **RF-2.1.** Consultar información de *Gestión e Investigación*.
 - **RF-2.1.** Consultar información de *Normativa Legal*.
- **RF-3.** Pruebas de software:
 - **RF-3.1.** Realizar tests unitarios.
 - **RF-3.2.** Realizar test de cobertura.
 - **RF-3.2.** Usar integración continua.
- **RF-4.** Configuración automática:
 - **RF-4.1.** Usar despliegue automático.
 - **RF-4.2.** Usar provisionamiento.

4.1.3. Requisitos no funcionales

Los requisitos no funcionales son las características propias del desarrollo, pero que no tienen que estar relacionadas con su funcionalidad.

- **RN-1.** Toda la programación del portal se hará en `Node.js` y los módulos que se usen deben ser instalables a través de su gestor de paquetes NPM.
- **RN-2.** El portal se iniciará y se detendrá mediante scripts lanzados con NPM.
- **RN-3.** Para iniciar la ejecución del portal es necesario que reciba el puerto de escucha del servidor y la dirección de acceso.
- **RN-4.** Todos los módulos se ejecutarán desde scripts lanzados con NPM.
- **RN-5.** Los tests unitarios se realizarán en base a comportamientos esperados y valores de estados. recibidos como contestación a las peticiones que se realicen al portal.
- **RN-6.** Los tests unitarios tienen que recibir las páginas del portal para ejecutarse.
- **RN-7.** El test de cobertura tiene que tener una automatización integrable con los tests unitarios.
- **RN-8.** La integración continua se ejecutará automáticamente con cada cambio que se haga en la programación del portal.
- **RN-9.** El despliegue automático se realizará mediante conexiones SSH.
- **RN-10.** Tanto para el despliegue automático como para el provisionamiento es necesario indicar el usuario que lo realiza y el destino en el que se realiza.

4.1.4. Requisitos de información

Los requisitos de información se refieren a la información que es necesaria almacenar en el sistema. La única información relevante que se va a almacenar son los datos descriptivos y de enlace de cada uno de los elementos del portal **OpenData UGR** que se van a mostrar en **UGR Transparente**.

- **RI-1.** Datos abiertos.
 - Información sobre cada uno de los elementos que se van a mostrar en el portal de transparencia como datos abiertos.
 - Contenido: nombre, categoría, conjunto de datos, enlace a **OpenData UGR**, enlace al recurso.

4.2. Modelos de casos de uso

Aunque ya se ha indicado que la parte funcional ya se encuentra implementada de forma previa a este proyecto, se van a incluir unos modelos de caso de uso simples para dar una visión más clara del funcionamiento general del portal UGR *Transparente*.

4.2.1. Descripción básica de actores

- **Ac-1.** Desarrollador.
 - Descripción: Encargado del desarrollo y administración del portal.
 - Características: Su trabajo está en el lado del servidor que genera la página, nunca trabaja desde el lado del cliente.
 - Relaciones: Ninguna.
 - Atributos: Ninguno.
 - Comentarios: Es el encargado de que la información se muestre en el portal y de añadir funcionalidades al portal.
- **Ac-2.** Usuario.
 - Descripción: Persona que usa el portal para consultar datos.
 - Características: Es el usuario común que accederá a la página.
 - Relaciones: Ninguna.
 - Atributos: Ninguno.
 - Comentarios: El usuario no es necesario que tenga ningún conocimiento previo al uso del portal, simplemente accederá y consultará los datos que sean de su interés.

4.2.2. Descripción casos de uso

- **CU-1.** Inicio automático del servidor del portal.
 - Actores: Desarrollador.
 - Tipo: Primario, esencial.
 - Referencias:
 - Precondición: El servidor esté detenido.
 - Postcondición: El portal será accesible públicamente.
 - Autor: Jaime Torres Benavente.
 - Versión: 1.0.
 - Propósito: Iniciar el servidor del portal UGR *Transparente*.
 - Resumen: Cuando se ejecuta el script de inicio de la aplicación, arranca el servidor y el portal será accesible desde Internet.

| Curso normal | | | |
|--------------|--|---------|--|
| Actor | | Sistema | |
| 1 | Desarrollador: da orden de que se inicie el servidor del portal. | | |
| | | 2a | Comprueba que el servidor está detenido y lo inicia para que el portal esté operativo. |

Tabla 4.1: Curso normal de CU-1. Inicio automático del servidor del portal

| Curso alterno | |
|---------------|--|
| 2b | Si el servidor está funcionando, no se ejecuta el script de inicio del servidor. |

Tabla 4.2: Curso alterno de CU-1. Inicio automático del servidor del portal

- **CU-2.** Consultar información de *Administración*.
 - Actores: Usuario.
 - Tipo: Primario, esencial.
 - Referencias:
 - Precondición: Existan archivos con los datos abiertos.
 - Postcondición:
 - Autor: Jaime Torres Benavente.
 - Versión: 1.0.
 - Propósito: El usuario consulta datos de administración en el portal UGR **Transparente**.
 - Resumen: El usuario que accede al portal de transparencia selecciona la sección de *Administración* y consulta la información de sus subsecciones.

| Curso normal | | | |
|--------------|---|---------|--|
| Actor | | Sistema | |
| 1 | Usuario: consulta la información de <i>Administración</i> publicada en el portal. | | |
| | | 2 | Se generan las tablas con todos los elementos de información de la sección <i>Administración</i> . |
| | | 3 | Se muestran en la página todos las tablas generadas con los elementos de información. |

Tabla 4.3: Curso normal de CU-2. Consultar información de Administración

■ **CU-3.** Consultar información de *Docencia*.

- Actores: Usuario.
- Tipo: Primario, esencial.
- Referencias:
- Precondición: Existan archivos con los datos abiertos.
- Postcondición:
- Autor: Jaime Torres Benavente.
- Versión: 1.0.
- Propósito: El usuario consulta datos de administración en el portal UGR **Transparente**.
- Resumen: El usuario que accede al portal de transparencia selecciona la sección de *Docencia* y consulta la información de sus subsecciones.

| Curso normal | | | |
|--------------|---|---------|--|
| Actor | | Sistema | |
| 1 | Usuario: consulta la información de <i>Docencia</i> publicada en el portal. | | |
| | | 2 | Se generan las tablas con todos los elementos de información de la sección <i>Docencia</i> . |
| | | 3 | Se muestran en la página todos las tablas generadas con los elementos de información. |

Tabla 4.4: Curso normal de CU-3. Consultar información de Docencia

■ **CU-4.** Consultar información de *Gestión e Investigación*.

- Actores: Usuario.
- Tipo: Primario, esencial.
- Referencias:
- Precondición: Existan archivos con los datos abiertos.
- Postcondición:
- Autor: Jaime Torres Benavente.
- Versión: 1.0.
- Propósito: El usuario consulta datos de administración en el portal UGR **Transparente**.
- Resumen: El usuario que accede al portal de transparencia selecciona la sección de *Gestión e Investigación* y consulta la información de sus subsecciones.

| Curso normal | | | |
|--------------|--|---------|---|
| Actor | | Sistema | |
| 1 | Usuario: consulta la información de <i>Gestión e Investigación</i> publicada en el portal. | | |
| | | 2 | Se generan las tablas con todos los elementos de información de la sección <i>Gestión e Investigación</i> . |
| | | 3 | Se muestran en la página todos las tablas generadas con los elementos de información. |

Tabla 4.5: Curso normal de CU-4. Consultar información de Gestión e Investigación

- **CU-5.** Consultar información de *Normativa Legal*.
 - Actores: Usuario.
 - Tipo: Primario, esencial.
 - Referencias:
 - Precondición: Existan archivos con los datos abiertos.
 - Postcondición:
 - Autor: Jaime Torres Benavente.
 - Versión: 1.0.
 - Propósito: El usuario consulta datos de administración en el portal UGR **Transparente**.
 - Resumen: El usuario que accede al portal de transparencia selecciona la sección de *Normativa Legal* y consulta la información de sus subsecciones.

| Curso normal | | | |
|--------------|--|---------|---|
| Actor | | Sistema | |
| 1 | Usuario: consulta la información de <i>Normativa Legal</i> publicada en el portal. | | |
| | | 2 | Se generan las tablas con todos los elementos de información de la sección <i>Normativa Legal</i> . |
| | | 3 | Se muestran en la página todos las tablas generadas con los elementos de información. |

Tabla 4.6: Curso normal de CU-5. Consultar información de Normativa Legal

■ **CU-6.** Realizar tests unitarios.

- Actores: Desarrollador.
- Tipo: Primario, esencial.
- Referencias:
- Precondición: Existan tests unitarios.
- Postcondición:
- Autor: Jaime Torres Benavente.
- Versión: 1.0.
- Propósito: Realizar tests unitarios para comprobar las funcionalidades de la aplicación.
- Resumen: Cada vez que se añadan nuevas páginas o funcionalidades al portal, se comprueba que funcionan correctamente.

| Curso normal | | | |
|--------------|--|---------|--|
| Actor | | Sistema | |
| 1 | Desarrollador: da orden de ejecutar los tests unitarios. | | |
| | | 2a | Comprueba que hay tests unitarios y los ejecuta. |

Tabla 4.7: Curso normal de CU-6. Realizar tests unitarios

| Curso alterno | |
|---------------|---|
| 2b | Si no hay tests unitarios creados, el sistema no hace nada. |

Tabla 4.8: Curso alterno de CU-6. Realizar tests unitarios

■ **CU-7.** Realizar tests de cobertura.

- Actores: Desarrollador.
- Tipo: Primario, esencial.
- Referencias: (CU-6.) Realizar tests unitarios.
- Precondición: Existan tests unitarios.
- Postcondición:
- Autor: Jaime Torres Benavente.
- Versión: 1.0.
- Propósito: Realizar test de cobertura para comprobar calidad de los tests unitarios.
- Resumen: Para comprobar si los tests unitarios cumplen correctamente con su función se analiza el porcentaje del código que está cubierto por los mismos.

| Curso normal | | | |
|--------------|--|---------|---|
| Actor | | Sistema | |
| 1 | Desarrollador: da orden de ejecutar test de cobertura. | | |
| | | 2a | Comprueba que hay tests unitarios y los ejecuta. |
| | | 3 | Se ejecuta el test de cobertura en base a los tests unitarios ejecutados. |

Tabla 4.9: Curso normal de CU-7. Realizar test de cobertura

| Curso alterno | |
|---------------|---|
| 2b | Si no hay tests unitarios creados, el sistema no hace nada. |

Tabla 4.10: Curso alterno de CU-7. Realizar test de cobertura

- **CU-8.** Usar integración continua.
 - Actores: Desarrollador.
 - Tipo: Primario, esencial.
 - Referencias: (CU-5.) Realizar tests unitarios.
 - Precondición: Existan test unitarios.
 - Postcondición: Se genera un informe con el resultado de los tests unitarios.
 - Autor: Jaime Torres Benavente.
 - Versión: 1.0.
 - Propósito: Comprobar si los cambios en la aplicación provocan errores.
 - Resumen: Cuando se efectúan cambios en la aplicación automáticamente se ejecutan los tests unitarios para comprobar si los cambios introducidos provocan conflictos en la plataforma.

| Curso normal | | | |
|--------------|---|---------|--|
| Actor | | Sistema | |
| 1 | Desarrollador: guardar cambios en el repositorio. | | |
| | | 2a | Comprueba que hay tests unitarios y comienza la verificación de la integración continua. |
| | | 3 | Ejecuta los tests unitarios para cada entorno definido. |

Tabla 4.11: Curso normal de CU-8. Usar integración continua

| Curso alterno | |
|---------------|---|
| 2b | Si no hay tests unitarios creados, el sistema no hace nada. |

Tabla 4.12: Curso alterno de CU-8. Usar integración continua

- **CU-.9** Usar despliegue automático.
 - Actores: Desarrollador.
 - Tipo: Primario, esencial.
 - Referencias: (CU-1.) Iniciar plataforma.
 - Precondición:
 - Postcondición: Los cambios introducidos son aplicados en la plataforma.
 - Autor: Jaime Torres Benavente.
 - Versión: 1.0.
 - Propósito: Aplicar automáticamente los cambios realizados a la aplicación.
 - Resumen: Para no tener que acceder manualmente al servidor de la plataforma y tener que desplegar los cambios introducidos, desde el entorno de desarrollo desplegamos automáticamente los cambios en el servidor.

| Curso normal | | | |
|--------------|---|---------|---|
| Actor | | Sistema | |
| 1 | Desarrollador: ordenar despliegue automático. | | |
| | | 2 | Conectar al servidor. |
| | | 3 | Crear copia de seguridad. |
| | | 4a | Comprobar si hay cambios en el repositorio de la plataforma y descargarlos en dicho caso. |
| | | 5 | Iniciar la plataforma con los cambios aplicados. |

Tabla 4.13: Curso normal de CU-9. Usar despliegue automático

| Curso alterno | |
|---------------|---|
| 4b | Si no hay cambios aplicables, se continua con el proceso. |

Tabla 4.14: Curso alterno de CU-9. Usar despliegue automático

- **CU-.10** Usar provisionamiento.
 - Actores: Desarrollador.
 - Tipo: Primario, esencial.
 - Referencias: (CU-1.) Iniciar plataforma.
 - Precondición:
 - Postcondición: El portal queda instalado en la infraestructura seleccionada.
 - Autor: Jaime Torres Benavente.
 - Versión: 1.0.
 - Propósito: Instalar automáticamente el portal en una infraestructura dada.
 - Resumen: Se instalarán automáticamente todos los elementos necesarios para poner en funcionamiento el portal en cualquier infraestructura que se indique.

| Curso normal | | | |
|--------------|---|---------|---|
| Actor | | Sistema | |
| 1 | Desarrollador: ordenar despliegue automático. | | |
| | | 2 | Conectar al servidor. |
| | | 3 | Comprobar aplicación necesaria. |
| | | 4a | Se comprueba si la aplicación necesaria está instalada, instalándola en caso de que no lo esté. |
| | | 5 | Descarga la plataforma desde el repositorio. |
| | | 6 | Instalar todas las dependencias de la plataforma. |
| | | 7 | Se establecen los parámetros de acceso al portal desde Internet. |
| | | 8 | Con todo preparado, se inicia la plataforma. |

Tabla 4.15: Curso normal de CU-10. Usar provisionamiento

| Curso alterno | |
|---------------|--|
| 4b | Si la aplicación está instalada, se continua con el proceso. |

Tabla 4.16: Curso alterno de CU-10. Usar provisionamiento

4.3. Diagrama de paquetes

Este diagrama representa la estructura lógica del sistema basado en las dependencias existentes entre sí. El paquete de **Configuración automática** depende del paquete **Administración portal** porque necesita de tareas de administración como es iniciar el servidor.



Figura 4.1: Diagrama de paquetes

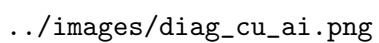
4.4. Diagramas de casos de uso

Los diagramas de casos de uso representan como los diferentes actores se relacionan con el sistema para usar sus funciones. Por ejemplo, en el primer caso de uso vemos como el **desarrollador** se relaciona con el sistema para

iniciar el servidor del portal, mientras, en el segundo vemos como el **usuario** se relaciona con el sistema para consultar la información de las diferentes secciones del portal.



Figura 4.2: Diagrama de casos de uso: paquete Administración portal




../images/diag_cu_ai.png

Figura 4.3: Diagrama de casos de uso: paquete Acceso información

En el tercer diagrama vemos como el **desarrollador** se relaciona con los procesos relacionadas con las pruebas (que a su vez son dependientes entre sí); y en el cuarto vemos como también el **desarrollador** se relaciona con el sistema para realizar las tareas de configuración automática, como son el despliegue automático y el provisionamiento.



Figura 4.4: Diagrama de casos de uso: paquete Pruebas de software



../images/diag_cu_ca.png

Figura 4.5: Diagrama de casos de uso: paquete Configuración automática

4.5. Diagramas de actividad

Los diagramas de actividad sirven para representar la descomposición de un proceso en las diferentes acciones de las que está compuesto. Las actividades de consultar algún tipo de información son procedimiento secuenciales en los que la ejecución es bastante simple; pero la actividad de iniciar el servidor del portal, realizar los test unitarios, realizar los test de cobertura, usar la integración continua tienen situaciones condicionales que son los que dan lugar a cursos alternos de la ejecución. Las actividades de despliegue automático y provisionamiento además tienen también puntos de sincronización que harán que el proceso siga el mismo cauce en su ejecución.



Figura 4.6: Diagrama de actividad CU-1. Inicio automático del servidor del portal



Figura 4.7: Diagrama de actividad CU-2. Consultar información de Administración



Figura 4.8: Diagrama de actividad CU-3. Consultar información de Docencia



Figura 4.9: Diagrama de actividad CU-4. Consultar información de Gestión e Investigación



Figura 4.10: Diagrama de actividad CU-5. Consultar información de Normativa Legal



Figura 4.11: Diagrama de actividad CU-6. Realizar tests unitarios



Figura 4.12: Diagrama de actividad CU-7. Realizar test de cobertura



Figura 4.13: Diagrama de actividad CU-8. Usar integración continua



Figura 4.14: Diagrama de actividad CU-9. Usar despliegue automático

4.6. Diagrama conceptual

En el diagrama conceptual podemos ver una representación de la estructura de la implementación. A excepción de la clase **test**, todas las clases son parte de la aplicación principal (**app**) por lo que tienen una relación de composición con la misma y no tienen sentido sin esta. Las clases de **test** y **app** tienen una relación de agrupación, porque el módulo test puede realizar las pruebas sobre cualquier módulo de aplicación que sea recibido.

4.7. Otros diagramas o interfaces

Gran parte de la implementación no va a ser de la aplicación principal en si misma, sino herramientas que se le van a agregar para obtener dife-



Figura 4.15: Diagrama de actividad CU-10. Usar provisionamiento

rentes funcionalidades que se usarán durante su desarrollo, de ahí que no se considere necesario el realizar diagramas de comunicación ni diagramas de secuencia.

En cuanto a la interfaz gráfica, no será necesaria porque todas las herramientas solo tienen modo de funcionamiento a través de terminal.



Figura 4.16: Diagrama conceptual

Capítulo 5

Diseño

5.1. Diseño general del portal

Para describir el diseño que se va a seguir en la programación del proyecto hay que tener en cuenta que se va a usar `Node.js`, que está basado en el lenguaje de programación `JavaScript` (que es un lenguaje orientado a objetos). Se usará un paradigma de programación orientado a objetos con diversas clases, que en este entorno de programación suelen ser referidos como módulos.

Existirá un módulo principal de la plataforma (`app.js`) que será la plataforma en si misma, que es el encargado de generar el servidor al que se realizarán las peticiones y visualizar cada unas de las páginas. Existirán también un módulo por cada una de las secciones del portal: *Administración* (`administración.js`), *Docencia* (`docencia.js`), *Gestión e Investigación* (`gestionInvestigacion.js`) y *Normativa Legal* (`normativaLegal.js`). Si comparamos este esquema con el de una aplicación más tradicional, podríamos considerar el módulo (`app.js`) como la clase principal y el resto de módulos como otras clases que tienen como función ser atributos compuestos de esa clase principal.

Cada uno de estos módulos de las diversas secciones, obtendrá la información para generar las páginas de sus subsecciones desde un archivo de datos *JSON*, organizados de la siguiente forma.

- UGR Transparente (`app.js`).
 - *Administración* (`administración.js`).
 - *Personal* (`personal.json`).
 - *Información Económica* (`infoEcononica.json`).
 - *Servicios* (`servicios.json`).
 - *Docencia*.
 - *Oferta y Demanda Académica* (`ofertaDemanda.json`).
 - *Claustro* (`clauastro.json`).
 - *Estudiantes* (`estudiantes.json`).
 - *Gestión e Investigación*.
 - *Misión* (`mission.json`).
 - *Plan Estratégico* (`.json`).
 - *Gobierno* (`gobierno.json`).
 - *Estadísticas* (`estadistica.json`).
 - *Normativa Legal*.
 - *Normativa Legal* (`normativaLegal.json`).

Cada vez que un usuario quiere consultar la información de una subsección, el portal de transparencia (UGR Transparente) hace una llamada al método de generación de la página de la sección elegida para su subsección determinada; ese método, procesará la plantilla *Jade* para la página pasándole el archivo *JSON* de datos y el resultado final será la página web que será visible en el portal.

De igual forma a lo mencionado de la aplicación del portal de transparencia, los test unitarios (`test.js`) y el despliegue automático (`flightplan.js`) son otros módulos que ejecutan dichas acciones, el test de cobertura se realiza automáticamente a partir de los test unitarios, la integración continua se realiza en base a la configuración del archivo `.travis.yml`, y finalmente, el provisionamiento sigue las tareas en el archivo de configuración `transparente.yml`. Todos estos módulos son independientes del funcionamiento del módulo principal, y simplemente llamaremos cuando queramos hacer uso de sus funcionalidades.

5.2. Diseño de una aplicación con desarrollo colaborativo

En varias ocasiones se ha indicado que este proyecto parte de un desarrollo colaborativo, esto es debido a que hoy en día es muy difícil concebir un proyecto de software libre fuera de un entorno colaborativo que permita publicar el código libremente en Internet donde sea accesible por cualquier persona, esta filosofía es la que sigue por ejemplo el desarrollo de **Linux** a principio de los años 90, el que se podría considerar como el más grande y relevante proyecto de software libre a nivel mundial. Además de abogar por la transparencia, este sistema de desarrollo hace que se más fácil encontrar errores durante el desarrollo ya que un mayor número de personas tienen acceso total al mismo. También hay que tener en cuenta como en todo proyecto que es imprescindible una herramienta de control de versiones, aún más en uno colaborativo donde es fácil que se produzcan conflictos en los archivos modificados por unos y otros desarrolladores.

La plataforma de desarrollo colaborativo y la herramienta de control de versiones a usar son **GitHub** y **Git** respectivamente. **GitHub** es una de las plataformas de desarrollo colaborativo más usadas en proyectos libres y usa **Git** como sistema de control de versiones para gestionar los proyectos que se almacenan en la plataforma. Estas son las herramientas que nos van a permitir que nuestro proyecto se convierta en un proyecto que se desarrollo de forma ágil, concretamente se va a seguir una metodología *DevOps*.

Una metodología de desarrollo *DevOps* consiste inicialmente en no hacer distinción entre el desarrollo del software y la administración del mismo, todo estará comunicado para que sea posible realizar entregas del software de forma frecuente asegurándose de que esas mismas entregas continuas no sea el origen de fallos futuros. La forma de asegurarse de que esos fallos no se producirán es dividir todo el desarrollo en fases que tengan que realizarse secuencialmente, controlando a cada fase que no se produzcan errores en la misma; como una cadena de montaje en la que podemos estar seguro de que el producto que llega al final está en perfectas condiciones, porque en caso contrario hubiera sido retirado durante el proceso.

En este proyecto, hemos considerado que las fases que nos permitirían asegurarnos que el producto que sale de la cadena de montaje en perfectas condiciones sean los test unitarios, la integración continua y el despliegue automático, todo esto apoyado en un sistema de control de versiones y una plataforma de desarrollo colaborativo abierto. Además, también se usará provisionamiento para facilitar la portabilidad del portal de una infraestructura a otra distinta.

5.3. Diseño de los tests unitarios y test de cobertura

Para cumplir la parte de testeo de la implementación de la metodología *DevOps*, a partir del desarrollo de este proyecto se pretende seguir un desarrollo guiado por pruebas (en inglés, *Test-Driven Development* o *TDD*). Esto consiste en escribir primero las pruebas que consideremos que la aplicación debe superar y luego desarrollar el código que realice la función que queremos cumplir, pero superando dichas pruebas.

Estas pruebas estarán basadas en la ejecución de tests unitarios que estarán diseñados para comprobar de forma automática que el código escrito cumple con el objetivo determinado; por lo que si una vez ejecutamos todos los tests unitarios no obtenemos que todos han tenido una ejecución exitosa, tenemos que revisar el código escrito para saber por qué fallan.

Haciendo esto nos aseguraremos que todos las funcionalidades que vayamos añadiendo a la implementación de la aplicación funcionarán correctamente; también nos asegura que si tenemos que hacer grandes modificaciones en el código, siempre que estas modificaciones sigan pasando las pruebas, no deberemos preocuparnos por estropear el funcionamiento de la aplicación.

El tipo de test unitario que vamos a usar es un test basado en el comportamiento, es decir, la forma de evaluar si un test se supera con éxito o fracaso es mediante la comprobación de la respuesta que da nuestra aplicación ante una solicitud determinada. Ejemplos:

- Para comprobar que un archivo con datos de configuración ha sido cargado, el archivo cargado no debe ser nulo.
- Para comprobar que en la información cargado se encuentra el nombre de la categoría, la categoría debe tener una propiedad llamada “nombre”.
- Para comprobar que las páginas del portal están accesibles, las respuesta a las solicitudes se espera que tenga el valor 200 (código de respuesta estándar para peticiones correctas en conexiones *HTTP*).

Pero el diseño de las pruebas no termina con escribir los tests unitarios que consideremos oportunos, también es necesario que esos tests pasen a su vez un test de cobertura. Un test de cobertura comprueba el porcentaje de código desarrollado y/o número de funciones de la aplicación que se están evaluando con los tests unitarios desarrollados, con esto se puede verificar la completitud de los tests unitarios; si todos nuestros tests unitarios se evalúan correctamente, pero solo están cubriendo un 30 % del total

de la aplicación, este resultado satisfactorio no nos puede asegurar que la funcionalidad completa de la aplicación se vaya a ejecutar sin problemas.

Los tests unitarios se van a desarrollar usando **Should** y **Supertest**, ambos módulos de **Node.js** para evaluar comportamientos en las peticiones. Estos tests unitarios a su vez van a ser evaluados por **Mocha**, un framework de **JavaScript** para testeo que se ejecuta tanto en aplicaciones **Node.js** como en navegadores web. Por último, todo esto pasará por las pruebas de cobertura de código de **Istanbul**, que generará un informe detallado especificando la cantidad de código y el número de funciones que están cubiertas y también los que no.

5.4. Diseño de la integración continua

La integración continua (en inglés, *continuous integration* o *CI*) consiste en comprobar cada vez que hacemos cambios en el código de la aplicación esto no genere problemas en su ejecución, ya que el cambio del comportamiento en una simple función puede cambiar en gran parte el comportamiento de la aplicación entera, y si otra función requiere del resultado de esta última, las consecuencias pueden ser desastrosas. Teniendo en cuenta que vamos a trabajar en un entorno de desarrollo colaborativo, es todavía más necesario que se asegure la integridad de la aplicación frente a los cambios.

Para que esto se pueda realizar, con cada cambio se activará automáticamente un proceso que verificará mediante la ejecución de los tests unitarios escritos que los cambios realizados no producen errores en la aplicación. Al igual que pasaba con los tests unitarios, la integración continua es una medida que nos permite incrementar la calidad del software producido porque cuanto más a menudo se hagan estas comprobaciones, mayor será la facilidad para detectar fallos en el propio software.

Realizaremos la integración continua a través de **Travis CI**, que es un sistema distribuido de integración continua libre que está integrado con **GitHub**. Este sistema además tiene como ventajas el soporte para numerosos lenguajes y la posibilidad de ejecutar las pruebas en distintas versiones del propio lenguaje, por lo que fácilmente podríamos probar nuestro desarrollo con diferentes configuraciones sin tener que realizar cambios en nuestro sistema local.

El procedimiento para realizar la integración continua consistirá en configurar el repositorio del proyecto en **GitHub** para que cada vez se realice un cambio en los archivos del repositorio, este iniciará un proceso por el cual se creará un *build* automático en un servidor virtual de **Travis CI** que se ha creado con tal finalidad y se desplegará en él la última versión con los

cambios que acabamos de realizar, para acto seguido ejecutar las pruebas que hayamos desarrollado y generar un informe sobre los resultados de las pruebas para cada una de las configuraciones que hayamos definido para la integración continua. Estos informes son almacenados en el propio sitio de **Travis**, recibiendo un aviso por correo en el caso de que las pruebas no hayan finalizado exitosamente en alguna de las configuraciones.

5.5. Diseño del despliegue automático

Una vez que tenemos la seguridad de que toda la aplicación funciona correctamente, solo nos queda desplegarla en nuestro servidor de producción; además, vamos a automatizar las tareas que deberíamos realizar manualmente para evitar posibles errores. En esto consiste el despliegue automático, hacer efectivos los cambios en nuestro software de nuestro software en una o varias plataformas objetivo sin que tengamos que realizar personalmente y paso a paso el proceso necesario para ello, todo es automatizado.

Antes de pasar a implementar el despliegue automático, hay que pensar que tareas es necesario o queremos que se realicen. Independientemente del tipo de aplicación, como medida preventiva se debe hacer una copia de seguridad del estado actual de la aplicación desplegada en el servidor. La ejecución del portal depende de que el servidor creado por **Express** esté en ejecución, por lo que antes de desplegar la nueva versión, se debería detener la ejecución de la versión anterior. Con todo esto hecho solo nos quedaría obtener los propios cambios realizamos, comprobar que todas las dependencias se siguen cumpliendo y establecer los parámetros para que el servidor sea accesible desde Internet; para finalizar la tarea, se iniciaría la ejecución de la aplicación.

Todas las tareas descritas, queremos que sean ejecutadas de forma secuencial una tras otra, de forma automática y necesitar la interacción con el usuario. Para cumplir esto usaremos **Flightplan**, un módulo de **Node.js** que nos permite coordinar de forma automática el despliegue de aplicaciones con tareas de administración del sistema.

5.6. Diseño del provisionamiento

Con toda la aplicación ya realizada, para su instalación inicial en un plataforma determinada primero tenemos que provisionarla con todos los recursos software necesarios. Encontraremos una mayor cantidad de referencias a este procedimiento por su término en inglés, *software provisioning*, ya que el término en español es una traducción literal que salvo por el acompañamiento del término “*software*”, puede referirse a abastecimientos de recursos necesarios en general no necesariamente relacionado con el campo de la informática.

Entre el software básico necesario hay que considerar para instalar la propia plataforma del desarrollo de la aplicación (**Node.js**) y la herramienta de control de versiones (**Git**). Con esto instalado, se clona el repositorio en **GitHub** del proyecto en un directorio local, y una vez instalada las dependencias de la aplicación y establecidos los parámetros de acceso, se arrancará el servidor para que el portal esté funcionando.

Todas estas tareas se harán de forma automática, para ello usaremos **Ansible**, una aplicación de software libre que nos permitirá configurar y administrar los ordenadores que indiquemos en la lista de hosts objetivo. Su funcionamiento básicamente se basa en conectarse al ordenador a configurar mediante una conexión *SSH* (por lo que deberemos indicarle con el usuario que tiene que realizarse la conexión) e irá ejecutando secuencialmente cada unas de las tareas del archivo *YAML* de su archivo de configuración (al que se hace referencia como *playbook*).

Capítulo 6

Implementación

El desarrollo del portal de transparencia fue iniciado por **Jaime Torres Benavente**, también becario anteriormente en la **Oficina de Software Libre de la Universidad de Granada** que se encargó de establecer las bases del proyecto. A su finalización del periodo de prácticas el estado del proyecto era el siguiente:

- **OpenData UGR**: utilizando la plataforma de código abierto **CKAN** para crear el portal de datos, se desarrolló **OpenData UGR** aplicándole un capa de personalización a la plataforma original, con la ventaja de proveer una API con la que crear, borrar y consultar todos los datos almacenados.
- **UGR Transparente**: el portal de transparencia funcionando bajo el funcionamiento conjunto de la plataforma de desarrollo **Node.js**, el framework web **Express** y el motor de plantillas **Jade**, que obtiene la información mediante consultas a una base de datos no relacional **MongoDB**.
- **Backend**: una base de datos que almacena la información referente a cada conjunto de datos que se muestra en el portal. Entre los datos almacenados, se encuentran las *URLs* para realizar peticiones a la API de nuestro **CKAN** (**OpenData UGR**) que nos permiten descargar la información contenida en sus tablas, bien en formato *CSV* o en formato *PDF*.
- **Frontend**: en primer lugar se convirtió una página con el formato de la página institucional de la **UGR** al lenguaje de plantillas **Jade** y se publicó en un repositorio de **GitHub**: <https://github.com/oslugr/PlantillaNeutraUGR>. Como el objetivo del portal de transparencia es el de visualizar y descargar los datos abiertos, las plantillas las *URLs* de **CKAN** para, mediante el uso de llamadas **AJAX**, descargar los datos de esas direcciones proporcionadas. Una vez procesados todos los datos

del archivo recibido, se muestran todas las tablas recibidas; teniendo además la opción de descargar o visualizar los datos de esas tablas. Esta visualización se hace abriendo una ventana modal en la que tras descargar el archivo *CSV* desde *CKAN* con *AJAX*, utilizamos *Google Charts* para generar las tablas con los datos recibidos.

Una vez comentado el trabajo previo existente, para explicar la implementación que he realizado se van a incluir pequeños fragmentos del código de los archivos escritos para la aplicación. Estos fragmentos solo contendrán las líneas más significativas, que además irán acompañados de una descripción sobre su significado y funcionamiento.

6.1. Uso de JSON como origen de datos

Inicialmente el origen de datos para las tablas de las páginas del portal de transparencia era una base de datos *NoSQL* montada sobre un sistema *MongoDB*. Aunque funcionaba, había problemas relacionados con el funcionamiento asíncrono de *Node.js*; por dicho funcionamiento *asíncrono* el proceso de generar la página del portal y el proceso de acceder a la información de la base de datos eran independientes en su ejecución aunque dependientes en su funcionamiento, por lo que a no ser que ambos procesos estuvieran perfectamente sincronizados esto derivaría en la aparición de problemas.

Como no se consiguió que se sincronizaran los *callbacks* de ambos procesos eventualmente se producía uno de las situaciones cuando se intentaba cargar una página:

- Si la página se intentaba cargar antes de que se hubiera podido acceder a la base de datos, las tablas con datos a visualizar no existían, por lo que se producía un error interno con el código 500.
- Si la página se intentaba cargar antes de que se hubieran podido recuperar todos los datos de la base de datos las tablas de datos se generaban vacías, por lo que las páginas que se cargaban se cargaban en blanco.
- Solo si la llamada a la base de datos se resolvía antes de que se intentara mostrar la información, la página se mostraba correctamente.

Después de replantear la situación, se consideró que como todos los datos iban a estar finalmente almacenados en la plataforma **OpenData UGR**, la solución más recomendable era eliminar la base de datos y usar en su lugar archivos *JSON* que hicieran de índices de enlaces hacía los conjuntos de datos **OpenData UGR** desde el portal de transparencia. La ventaja de usar archivos *JSON* es la gran flexibilidad que proporcionan a la hora de realizar estructuras de datos; aprovechando esto se diseñaron estos archivos con los siguientes campos (ejemplo en el *fragmento de código 6.1*):

- **nombre:** es la subcategoría en el portal de transparencia.
- **plantilla:** el archivo de plantilla a partir del que se generan las páginas del portal de transparencia.
- **contenido:** conjunto con la información de cada una de las tablas que muestran en las páginas del portal de transparencia.
 - **encabezado:** nombre de la tabla.
 - **link:** salto a la posición de la página donde empieza la tabla.
 - **texto:** descripción de los datos que contiene la tabla.
- **datos:** conjunto con los elementos que componen cada una de las tablas.
 - **dataset:** tabla a la que pertenece el elemento.
 - **id_dataset:** conjunto de datos en **OpenData UGR** al que pertenece el elemento.
 - **nombre:** descripción del elemento que visualizará en la tabla.
 - **vista:** valor que indica si el elemento se puede previsualizar desde el propio portal de transparencia (en caso de ser 1) o si solo se puede visualizar accediendo a su enlace a **OpenData UGR** (en caso de ser 0).
 - **url:** dirección al elemento como recurso dentro de **OpenData UGR** para poder ser visualizado.
 - **descarga:** dirección de descarga directa del elemento almacenado como recurso en **OpenData UGR**.

```
1 {
2   "nombre": "Personal",
3   "plantilla": "personal",
4   "contenido": [
5     {
6       "encabezado": "Informaci n Salarial 2015",
7       "link": "informacion-salarial-2015",
8       "texto": "Informaci n relativa a la oferta..."
9     },
10
11   ],
12   "datos": [
13     {
14       "dataset": "Informaci n Salarial 2015",
15       "id_dataset": "informacion-salarial-2015",
16       "nombre": "An lisis total plantilla: g nero",
17       "vista": 1,
18       "url": "51d53138-0408-4257-9909-57acea137a58",
19       "descarga": "985a8ele-734b-432a-ac65-a7da..."
20     },
21
22   ]
23 }
```

Fragmento de código 6.1: Archivo JSON con información de personal

Para que estos archivos puedan ser utilizados desde la aplicación, tenemos que convertir los archivos *JSON* en objetos *JSON*, esto lo podemos hacer fácilmente usando dos sencillos métodos como vemos en el *fragmento de código 6.2*. Ambos métodos pertenecen al módulo `cargar.js` de la carpeta **lib**:

- `readFileSync`: es un método del módulo `Fs` (módulo encargado de realizar las operaciones de entrada/salida en `Node.js`) que nos permite leer de forma síncrona los archivos que reciba como argumento.
- `JSON.parse`: es un método del objeto *JSON* que nos permite convertir el texto que reciba como argumento en un objeto *JSON* (siempre que este tenga una formato compatible con *JSON*).

Es importante aclarar que en `Node.js` los módulos, independientemente de si son creados por nosotros o no, se importan para ser usados dentro de módulo con la palabra clave *“require”* y se exportan para ser utilizables por otros módulos con la palabra clave *“export”*.

```
1 var fs = require("fs");
2
3 var cargar = function (archivo){
4     var config = null;
5
6     try{
7         config = JSON.parse(fs.readFileSync(archivo));
8     }
9     catch(e){
10         console.log("Error: no existe el archivo " + archivo);
11     }
12
13     return config;
14 };
15
16 module.exports = cargar;
```

Fragmento de código 6.2: Módulo para cargar archivos JSON

Tenemos toda la información como objetos *JSON*, así que ya podemos hacer uso de ella desde nuestra aplicación. Desde la misma aplicación principal lo primero será importar los módulos que nos van a permitir crear la infraestructura web lógica para nuestro portal, estos son los módulos **Express** y **Http**.

El siguiente paso será importar los módulos de cada una de las secciones del portal; en la línea 4 del *fragmento de código 6.3* podemos ver como ejemplo como importamos el módulo de la sección *Administración* que se correspondería con el archivo **administracion.js** de la carpeta **routes**, la carpeta en la que se encuentran todos los módulos de cada una de las secciones.

A continuación, como vemos en la línea 6, importamos el módulo que hemos descrito antes para la lectura de archivos *JSON* y procedemos a cargar todos los archivos *JSON* de origen de datos que se encuentran en la carpeta **config**, que se corresponden con la configuración del servidor del portal y los elementos de información de cada una de las secciones.

Otro de los aspectos esenciales en el código de la aplicación principal es crear las rutas para que las páginas del portal de transparencia sean accesibles desde Internet. En la línea 15 vemos como indicar que la ruta de una página de subsección se vincule con la función del módulo que va a generar esa página, en este caso la página de información de *Personal* (**personal.html**) se corresponde al método **personal** del módulo **administracion.js**.

Ya solo nos quedaría crear el servidor. Con la línea 8 creamos una instancia del módulo **Express**, ahora mediante el módulo **Http** creamos el servidor sobre esa instancia e indicamos las opciones de accesibilidad necesarias, en este caso el puerto de escucha del servidor (**port**) y la dirección de acceso al portal (**ip**). Finalmente exportamos el módulo de la aplicación principal para más adelante poder realizarle los test los tests unitarios.

```
1 var express = require('express');
2 var http = require('http');
3
4 var administracion = require(__dirname+'/routes/administracion')
5   ;
6 var cargar = require(__dirname+'/lib/cargar');
7
8 var app = express();
9
10 config = cargar(__dirname+'/config/config.json');
11 module.exports.config = config;
12
13 module.exports.personal = cargar(__dirname+'/config/personal.
14   json');
15 app.get('/personal.html', administracion.personal);
16
17 http.createServer(app).listen(app.get('port'), app.get('ip'),
18   function(){
19     console.log('Express server listening on ' + app.get('ip') + '
20       :' + app.get('port'));
21   });
22 module.exports = app;
```

Fragmento de código 6.3: Módulo principal de la aplicación

La aplicación principal ya está descrita, así que ahora vamos a pasar a los módulos que van a generar las páginas del portal. La estructura de estos módulos es igual en todos los casos, primero se importa el módulo de la aplicación principal para acceder a los archivos JSON de los datos cargados y seguidamente se exporta la función que va a generar cada una de las páginas de las subsecciones del portal (esta es la misma función a la que se llamaba en la línea 15 de la aplicación principal). Cada una de estas funciones contendrá la llamada a una función **render** que será la encargada de pasarle toda la información a la plantilla **Jade** a partir de la que, finalmente, se generará la página web que será accesible en el portal.

```
1 var conf = require('./app');
2
3 exports.personal = function(req, res){
4   var personal = conf.personal;
5
6   res.render(personal.plantilla, {
7     servidor: conf.config.servidor,
8     seccion: personal.nombre,
9     contenido: personal.contenido,
10    datos: personal.datos,
11  });
12 };
```

Fragmento de código 6.4: Archivo administracion.js

Para arrancar y detener la aplicación se usarán dos scripts del archivo package.json que nos permitirán ejecutar estas tareas con NPM. En el caso del script de arranque es necesario que indiquemos los siguientes parámetros:

- El puerto de escucha del servidor (variable **PORT**). Es el puerto del que el servidor estará escuchando las peticiones, en el *fragmento de código 6.5* se ha especificado el puerto 3000 por ser una configuración local (podría ser cualquier otro puerto libre), pero para que el portal sea accesible públicamente se deberá configurar para el puerto 80.
- La dirección a través de la que se accederá al portal (variable **IP**). Al igual que en el caso del puerto, la dirección IP indicada (127.0.0.1) es para acceso local únicamente; para un acceso público sería necesario indicar una dirección IP o una URL que sea accesible desde Internet.

Con los parámetros especificados, el servidor comenzará a correr ejecutado mediante el módulo **Forever** que hará que la aplicación se ejecute de forma ininterrumpida al ejecutar ‘**npm start**’.

Para detener el servidor no tenemos que indicar nada, simplemente ejecutar el script de parada; este se encargará de buscar todos los procesos abiertos por la aplicación principal y mandarles la señal de terminación. Se ejecuta con ‘**npm run-script kill**’ por no estar reconocido como uno de los scripts por defecto de NPM.

```
1  "scripts": {  
2    "start": "PORT=3000 IP=127.0.0.1 forever start -l /var/log/  
        forever.log -a -o /var/log/out.log -e /var/log/err.log ./  
        app.js",  
3    "kill": "ps aux | grep 'app.js' | grep -v grep | awk '{print  
        \\\"sudo kill -9 \\\" $2}' | sh"  
4  }
```

Fragmento de código 6.5: Scripts de inicio y detención

6.2. Tests unitarios y de cobertura

Los tests unitarios que se han implementados cumplen con la función de probar la aplicación siendo evaluada en función de comportamientos deseados. Para ello se usarán los módulos **Should** y **Supertest**, el primero evaluará que los archivos *JSON* tengan todos los elementos necesarios para generar las páginas del portal, mientras que el segundo evaluará que las páginas del portal que se generan sean accesibles sin problemas. Todos los tests se encuentran en el archivo **test.js** del directorio **test**.

En el caso de los archivos *JSON* los comportamientos que tenemos que comprobar son muy simples:

- En el caso de cargar un archivo *JSON* con la función **cargar** del módulo **cargar**. Para comprobar que el archivo se ha cargado correctamente, se verifica que el archivo no sea nulo como se comprueba en la línea 10 del *fragmento de código 6.7*.
- Con el archivo ya cargado, ahora comprobamos que tiene cada uno de los campos que necesitamos para generar las páginas web; por ejemplo, en la línea 17 comprobamos que el archivo de configuración tengo el campo **nombre**.

Para comprobar que las páginas son accesibles, realizamos una solicitud de la misma a la que el servidor responderá con un código numérico, si ese código tiene el valor **200** significará que la petición se ha resuelto correctamente, por lo que significará que la página está accesible. Podemos ver como se hace esto observando las líneas 28 a 36.

Hemos descrito las comprobaciones que realizan los tests, nos falta por explicar como se evalúan dichas comprobaciones, los tests unitarios son ejecutados por Mocha. La ejecución de las pruebas se divide en dos procesos, una descripción que indica qué se está evaluando y la propia evaluación de un test unitario que nos informa si la aplicación supera el test o no; la descripción se realiza con la orden **describe** mientras que la evaluación se realiza con **it**. Un ejemplo de todo esto se puede ver en el *fragmento de código 6.7*.

```
1 var should = require("should"),
2 request = require("supertest");
3
4 describe('Test de carga y formato de JSONs', function(){
5   describe('Archivo de configuracion', function(){
6     var config = cargar(__dirname+"/../config/config.json");
7
8     describe('Carga de archivo', function(){
9       it('Cargado', function(){
10         config.should.not.be.null;
11       });
12     });
13
14     describe('Formato de archivo', function(){
15       describe('Campos obligatorios', function(){
16         it('nombre', function(){
17           config.should.have.property("nombre");
18         });
19       });
20     });
21   });
22 });
23
24
25 describe('Prueba de acceso', function(){
26   ..each(acceso.elemento, function(valor) {
27     it(valor.nombre, function(done){
28       request(app)
29         .get(valor.ruta)
30         .expect(200)
31         .end(function(err, res){
32           if (err){
33             throw err;
34           }
35           done();
36         });
37     });
38   });
39 });
```

Fragmento de código 6.6: Módulo de tests unitarios

Al igual que los scripts de inicio y parada de la aplicación principal, los


test se ejecutarán mediante un script de NPM. Se ha comentado el uso de **Mocha** para pasar los tests unitarios, pero no se ha comentado nada de los tests de cobertura; esto se debe a que el módulo elegido para realizar los tests de cobertura, **Istanbul**, está integrado con **Mocha** para poder realizar automáticamente los tests de cobertura a los tests unitarios que se ejecuten con este último. Ambos se ejecutarán con ‘`npm test`’.

```
1 "scripts": {  
2   "test": "istanbul cover _mocha ./test --recursive"  
3 }
```

Fragmento de código 6.7: Scripts de test

6.3. Integración continua

Haber elegido realizar la integración continua con **Travis CI** hace que incorporarla a nuestra aplicación sea un procedimiento realmente sencillo. Lo primero que tenemos que hacer es darnos de alta en la página de la plataforma (<http://travis-ci.org/>), para ello podemos usar nuestra propia cuenta de **GitHub**. Una vez dentro, solo tendremos que activar la integración continua para nuestro repositorio.



../images/activar_travis.png

Figura 6.1: Activación de la integración continua

La configuración por defecto de **Travis** es que se compruebe la integración continua con cada “*push*” o “*pull request*” que se haga a nuestro repositorio, aunque esto se puede cambiar en función de nuestras preferencias.



Figura 6.2: Disparador de integración continua

Para terminar, solo nos queda crear el archivo de configuración `.travis.yml` que será el que **Travis** siga para comprobar la integración continua. Aspectos básicos de este archivo son indicar en `language` el lenguaje en el que está escrita la aplicación y con el propio nombre del lenguaje, indicar también las versiones del mismo para las que se va a comprobar la aplicación. Para el portal hemos decidido que se comprueba la integración en las versiones estables más recientes **Node.js** que son las *0.10*, *0.11* y *0.12*, además de *iojs*, que es un derivado del propio **Node.js**.

```
1 # language setting
2 language: node_js
3
4 # version numbers, testing against two versions of node
5 node_js:
6 - "0.12"
7 - "0.11"
8 - "0.10"
9 - "iojs"
```

Fragmento de código 6.8: Archivo de configuración de Travis CI

6.4. Despliegue automático

Antes de poder realizar el despliegue automático tenemos que tener conexión mediante **SSH** al servidor, además es necesario que nuestro archivo de

clave pública esté copiado también el servidor. Si no tenemos un archivo de clave pública podemos generarlo con el comando “`ssh-keygen -t rsa`”, que por defecto usa un cifrado *RSA* de 128 bits. Ahora copiamos este archivo al servidor con el comando “`ssh-copy-id USUARIO@transparente.ugr.es`”, donde usuario es un usuario que pueda acceder mediante SSH al servidor.

Para realizar el despliegue tenemos que escribir un nuevo módulo con el nombre `flightplan.js`. Este módulo contendrá todo el procedimiento que seguirá `Flightplan` para realizar el despliegue de la aplicación en el servidor. Lo más importante a tener en cuenta en el contenido de este archivo es lo siguiente:

- **plan.target:** podemos configurar el despliegue para uno o varios objetivos, por lo que tenemos que indicar un nombre para diferenciar el destino de nuestro despliegue.
 - **host:** la dirección del servidor en el que vamos a desplegar la aplicación es necesario para que `Flightplan` sepa dónde hacerlo.
 - **username:** el usuario que ejecutará las tareas en el despliegue, hay que tener en cuenta que este usuario debe tener los permisos necesarios para ejecutar esas tareas. Por razones de seguridad/comodidad no definimos ningún usuario concreto, sino que lo introduciremos como argumento en la ejecución.
 - **agent:** será el método que usaremos para conectarnos al servidor, en nuestro caso será una conexión `SSH`.
- **plan.remote:** cada una de las tareas que se ejecutarán de forma remota para realizar el despliegue se introducen en esta función.
 - **remote.sudo:** estos serán los comandos que tienen que ser ejecutados con permisos de administración.
 - **remote.exec:** serán los comandos que se ejecutarán con el usuario que hemos lanzado el despliegue.
 - **remote.with:** los comandos que se ejecutan en bloque después de ejecutar un comando previo.
 - **remote.log:** estas serán las salidas que se mostrarán por pantalla y que usaremos para informar sobre el curso del proceso.

En el siguiente fragmento de código se puede ver como se realiza el despliegue del portal en el servidor de `UGR Transparente`. Las acciones a realizar son muy simples, además de que cada una está acompañada de una salida por pantalla que describe brevemente lo que realiza.

```
1 var plan = require('flightplan');
2
3 plan.target('transparente', {
4   host: 'transparente.ugr.es',
5   username: process.env.USER,
6   agent: process.env.SSH_AUTH_SOCK
7 });
8
9 plan.remote(function(remote) {
10   remote.log('Creando copia de seguridad...');
11   remote.sudo('cp -Rf ugr-transparente-servidor ugr-transparente
12     -servidor.bak', {user: process.env.USER});
13
14   remote.with('cd ugr-transparente-servidor', function() {
15     remote.log('Deteniendo el servidor...');
16     remote.exec('sudo npm run-script kill');
17     remote.log('Restableciendo parámetros de acceso...');
18     remote.exec('sed "s/IP=transparente.ugr.es/IP=127.0.0.1/" -i
19       package.json');
20     remote.exec('sed "s/PORT=80/PORT=3000/" -i package.json');
21     remote.log('Obteniendo cambios...');
22     remote.exec('git pull');
23     remote.log('Instalando dependencias...');
24     remote.exec('sudo npm install');
25     remote.log('Cambiano parámetros de acceso...');
26     remote.exec('sed "s/IP=127.0.0.1/IP=transparente.ugr.es/" -i
27       package.json');
28     remote.exec('sed "s/PORT=3000/PORT=80/" -i package.json');
29     remote.log('Arrancando el servidor...');
30     remote.exec('sudo npm start');
31   });
32 });
```

Fragmento de código 6.9: Módulo de despliegue automático

Al finalizar el despliegue el portal estará operativo como lo estaría normalmente, hemos podido actualizarlo sin tener que acceder y aplicar la actualización manualmente. Para ejecutar el despliegue automático también se usa un script de NPM que se ejecuta mediante “`USER=USUARIO npm run-script deploy`”, donde **USUARIO** es el usuario con acceso SSH al servidor y permisos para ejecutar todos los comandos.

```
1 "scripts": {
2   "deploy": "fly transparente"
3 }
```

Fragmento de código 6.10: Scripts de despliegue automático

6.5. Provisionamiento

Al igual que para el despliegue automático, para el provisionamiento necesitamos poder conectarnos al servidor mediante SSH. **Ansible** necesita que le indiquemos la dirección de nuestro servidor en un formato específico y mediante un archivo que se conoce como **ansible.hosts**. Un ejemplo de esta configuración se puede ver en el siguiente fragmento de código.

```
1 [ transparente ]
2 transparente.ugr.es
```

Fragmento de código 6.11: Archivo de hosts de Ansible

El archivo de configuración de **Ansible** (al que se suele referir como “*playbook*”) es un archivo con formato *YAML* que contiene todas las ordenes que se ejecutarán en proceso de provisionar nuestro servidor. El esquema general del archivo es el siguiente:

- **hosts**: el nombre con el que nos referimos a la máquina de la que hemos indicado la dirección de acceso en el archivo **ansible.host** al que se le realizará el provisionamiento.
- **sudo**: indica que los comandos que se ejecuten lo harán con permiso de administrador.
- **remote_user**: es el usuario con el que nos conectaremos al equipo, el que tiene que tener acceso mediante SSH al servidor.
- **tasks**: la lista de tareas que **Ansible** ejecutará de forma secuencial en el ordenador remoto.
 - **name**: Descripción que se mostrará por pantalla de la tarea que se está llevando a cabo en ese momento.
 - **“comando”**: es la orden que se ejecutará, podemos encontrar ordenes *especiales* reconocidas por **Ansible** y las órdenes comunes. Podremos diferenciar ambos tipos de órdenes porque las comunes son las que empiezan por la palabra **command**, mientras que las especiales suelen comenzar con alguna palabra relacionada con con el comando en cuestión, además de que suelen tener una mayor cantidad de opciones de ejecución.

En el siguiente fragmento de código se pueden ver los pasos a seguir para realizar el provisionamiento de un equipo, la descripción de las acciones que realizan vuelven ser bastante simples como se la descripción **name** de cada tarea.

```
1  ---
2  - hosts: transparente
3    sudo: yes
4    remote_user: "{{user}}"
5    tasks:
6      - name: Añadiendo repositorio para instalar Node.js...
7        apt_repository: repo='ppa:chris-lea/node.js '
8
9      - name: Actualizando lista de paquetes...
10       apt: update_cache=yes
11
12      - name: Instalando git...
13        apt: name=git state=present
14
15      - name: Instalando Node.js...
16        apt: name=nodejs state=present
17
18      - name: Clonando repositorio con la aplicacion...
19        git: repo=https://github.com/oslugr/ugr-transparente-
20              servidor.git
21              dest=/home/"{{user}}"/ugr-transparente-servidor
22              version=master
23
24      - name: Cambiando propietario del directorio de la
25        aplicacion...
26        file: path=/home/"{{user}}"/ugr-transparente-servidor
27              owner="{{user}}" group="{{user}}" state=directory
28              recurse=yes
29
30      - name: Instalando las dependencias de la aplicacion...
31        npm: path=/home/"{{user}}"/ugr-transparente-servidor
32
33      - name: Cambiando parametros de acceso (1/2)...
34        command: sed "s/IP=127.0.0.1/IP=transparente.ugr.es/" -i
35                  /home/"{{user}}"/ugr-transparente-servidor/
36                  package.json
37
38      - name: Cambiando parametros de acceso (2/2)...
39        command: sed "s/PORT=3000/PORT=80/" -i
40                  /home/"{{user}}"/ugr-transparente-servidor/
41                  package.json
42
43      - name: Arrancando el servidor...
44        command: chdir=ugr-transparente-servidor npm start
```

Fragmento de código 6.12: Playbook de Ansible

Este es la única funcionalidad que no se ejecuta como un script de NPM debido a que no es un módulo de Node.js, sino una aplicación externa. Para ejecutarla, desde introducimos el comando:

```
1 ANSIBLE_HOSTS=provisioning/ansible_hosts ansible-playbook provisioning  
  /transparente.yml --extra-vars 'user=USUARIO'
```

Fragmento de código 6.13: Línea de comando de Ansible

Estando dividido en tres partes:

- **ANSIBLE_HOSTS=provisioning/ansible_hosts**: le pasamos a Ansible como parámetro la localización del `ansible_hosts`.
- **ansible-playbook provisioning/transparente.yml**: ejecutamos el *playbook* `transparente.yml` con Ansible.
- **--extra-vars 'user=USUARIO'**: le pasamos a Ansible como variable el nombre del usuario con el que se va a conectar al ordenador.

6.6. Repositorio con el desarrollo

Todo la implementación descrita en este capítulo se puede encontrar en el contenido del repositorio del portal de transparencia: <https://github.com/oslugr/ugr-transparente-servidor>.

Además, esta aplicación está funcionando y es accesible desde la siguiente dirección: <http://transparente.ugr.es/>

Capítulo 7

Pruebas

En el capítulo anterior se describía como se había realizado toda la implementación del proyecto, por lo que ya solo queda probar que todo funciona correctamente.

7.1. Pruebas unitarias

Para ejecutar las pruebas unitarias vamos a ejecutar con **Mocha** a través de **Istanbul**, realizando las pruebas unitarias y generando el resultado de la prueba de cobertura en un misma ejecución. Cuando ejecutemos **npm test** comenzarán a comprobarse una por una todas las comprobaciones en el archivo **tests.js**; las llamadas al método **describe** son las líneas que aparecen como título en blanco mientras que las comprobaciones que se hacían con el método **it** son las líneas en gris, que aparecen con un *check* verde si el valor esperado es el correcto ó, en caso de error, con un número rojo que indique el error de forma enumerada.

Una vez que todas las comprobaciones hayan sido realizadas nos aparecerá un resumen con los resultados de los test pasados, indicando cuantos se han ejecutado correctamente (**passing**) y cuantos han producido un error (**failing**); si alguna de las tareas asíncronas no devolviera el mensaje **done** estas quedarían como comprobaciones pendientes (**pending**), pero esto es algo que no se produce en nuestro caso.



Figura 7.1: Ejecución de los test unitarios con Mocha (1/2)

En la siguiente imagen vemos que todas los tests se han pasado correctamente (**148 passing**) en un tiempo de **3 segundos**. Además, como hemos dicho que **Mocha** es ejecutado a través de **Istanbul**, también obtenemos el mensaje de que el test de cobertura se ha generado dentro de la carpeta **coverage** y un resumen de la cobertura del código que realizan nuestros test.



Figura 7.2: Ejecución de los test unitarios con Mocha (2/2)

Si alguno de los test fallara (por ejemplo, uno de los archivos de origen de datos no existe) en el resumen además del número de test fallados, se mostrará una descripción de los mismos. Es el caso de la siguiente imagen, el archivo `personal.json` no existe por lo que entre los 11 errores que se han producido vemos que en la comprobación de la existencia del archivo se produce un aserción debido a que el resultado esperado por **Mocha** es **true** y sin embargo recibe **false** en su lugar, relacionado con esto nos encontramos el siguiente error y es que como no se ha podido cargar el archivo no se han podido comprobar las propiedades que deberían tener ese archivo (la propiedad que espera **should** es nula).



Figura 7.3: Error en tests unitarios

7.2. Prueba de cobertura

Como veíamos al finalizar los test unitarios, el resultado del test de cobertura se almacena en la carpeta **coverage**, dentro de dicha carpeta encontraremos un archivo *HTML* con el mismo resumen que aparecía al finalizar la ejecución de los tests unitarios, pero en este caso permitiéndonos acceder a una información más completa.

Por ejemplo, vamos a comprobar más detalladamente la cobertura en el código de la aplicación principal de la plataforma, por lo que seleccionamos **ugr-transparente-servidor** y después el archivo de nuestra aplicación `app.js`. En el resumen todo el código de esta parte de la aplicación aparecía con una cobertura del **100 %** por eso todas las líneas del código aparecen en verde.



Figura 7.4: Resumen de resultados de test de cobertura

En el caso de que alguna sentencia no estuviera cubierta por los test unitarios, esta aparecería en rojo; esto es lo que sucede por ejemplo en uno de los módulos desarrollado: cuando se captura una excepción espera que sea lanzada a un nivel superior sin embargo solo se muestra un mensaje por pantalla, **Istanbul** aprecia esto como un fallo de cobertura y por eso destaca esa línea en rojo.



Figura 7.5: Resultado de cobertura de la aplicación principal

7.3. Integración continua

Tenemos los test unitarios hecho y hemos comprobado que funcionan, por lo que teniendo la configuración de **Travis CI** realizada, cada vez que se haga un *push* al repositorio se generará un *build* en sus servidores para comprobar que los nuevos cambios introducidos no producen errores en la aplicación. Esta comprobación se hará en cada una de las versiones de nuestro lenguaje que hayamos definido, obteniendo un resumen al final de su ejecución que nos indicará si todo se ha ejecutado correctamente; dicho resultado es accesible a través de su página web (<https://travis-ci.org/oslugr/ugr-transparente-servidor>). En la imagen siguiente tenemos el resultado de un *build* en el que no se ha producido ningún error, por lo que todos pruebas de las diferentes versiones están marcadas con un *check* verde.



Figura 7.6: Resultado de cobertura de un módulo desarrollado

Si seleccionamos en concreto alguna de las entradas, obtendremos la salida del despliegue de la aplicación y la posterior ejecución de los test.

La salida de la ejecución de los tests es exactamente igual a la salida que hemos visto que se produce se ejecutamos los tests de forma local llamando a `npm test` (que es lo hace Travis igualmente).

7.4. Despliegue automático

También vamos a comprobar que el despliegue automático funciona correctamente. Simplemente ejecutamos el script `deploy` pasándole como parámetro el usuario con el que nos conectaremos al servidor. En la imagen siguiente vemos el resultado de la ejecución, aunque no en este momento no había nuevos cambios que aplicar al servidor, si podemos ver como hace la copia de



Figura 7.7: Build en Travis CI generado con éxito

seguridad inicial y ejecuta el resto de órdenes necesarias para que la aplicación del portal quede en funcionamiento.



Figura 7.8: Salida del build en Travis CI (despliegue)

7.5. Provisionamiento

Como el sitio de **UGR Transparente** está funcionando sin problemas en este momento, para probar el provisionamiento crearemos una máquina virtual con **Vagrant** con el mismo sistema operativo que se encuentra en el servidor de la plataforma (**Ubuntu 14.04**) para realizarle el provisionamiento con el **playbook** escrito para **Ansible**.

Las órdenes a introducir para crear la máquina virtual son las que aparecen en el siguiente fragmento de código.

```
1 vagrant box add ubuntu/trusty64
2 vagrant init ubuntu/trusty64
3 vagrant up
```



Figura 7.9: Salida del build en Travis CI (resultados tests)

```
4 | vagrant ssh
```

Fragmento de código 7.1: Órdenes para crear máquina virtual Vagrant



Figura 7.10: Ejecución de despliegue automático

También tenemos que crear un archivo **Vagrantfile** en el que es importante establecer la dirección de acceso a la máquina y como provisionador **Ansible**, indicando también el *playbook* que se va a utilizar.

```
1 # -*- mode: ruby -*-
2 # vi: set ft=ruby :
3
4 VAGRANTFILE_API_VERSION = "2"
5
6 Vagrant.configure(VAGRANTFILE_API_VERSION) do |config|
7     config.vm.box = "ubuntu/trusty64"
8     config.vm.network :private_network, ip:"192.168.2.50"
9
10    config.vm.provision "ansible" do |ansible|
11        ansible.playbook = "transparente.yml"
12    end
13 end
```

Fragmento de código 7.2: Archivo Vagrantfile

Lo siguiente será cambiar en el archivo `ansible_hosts` la dirección del portal por la dirección de nuestra máquina virtual (comando `sed`), declarar como variable de entorno archivo `ansible_hosts` (comando `export`), y finalmente con **Vagrant**, recargar la configuración del **Vagrantfile** (`vagrant reload`) y ordenar el provisionamiento de la máquina virtual (`vagrant provision`). En el siguiente fragmento de código se listan todos estos comandos y en la imagen que le sigue se puede comprobar la ejecución, obteniendo como resultado un provisionamiento completamente exitoso.

```
1 sed "s/transparente.ugr.es/192.168.2.50/" -i ansible_hosts
2 export ANSIBLE_HOSTS=ansible_hosts
3 vagrant reload
4 vagrant provision
```

Fragmento de código 7.3: Órdenes para provisionar máquina Vagrant

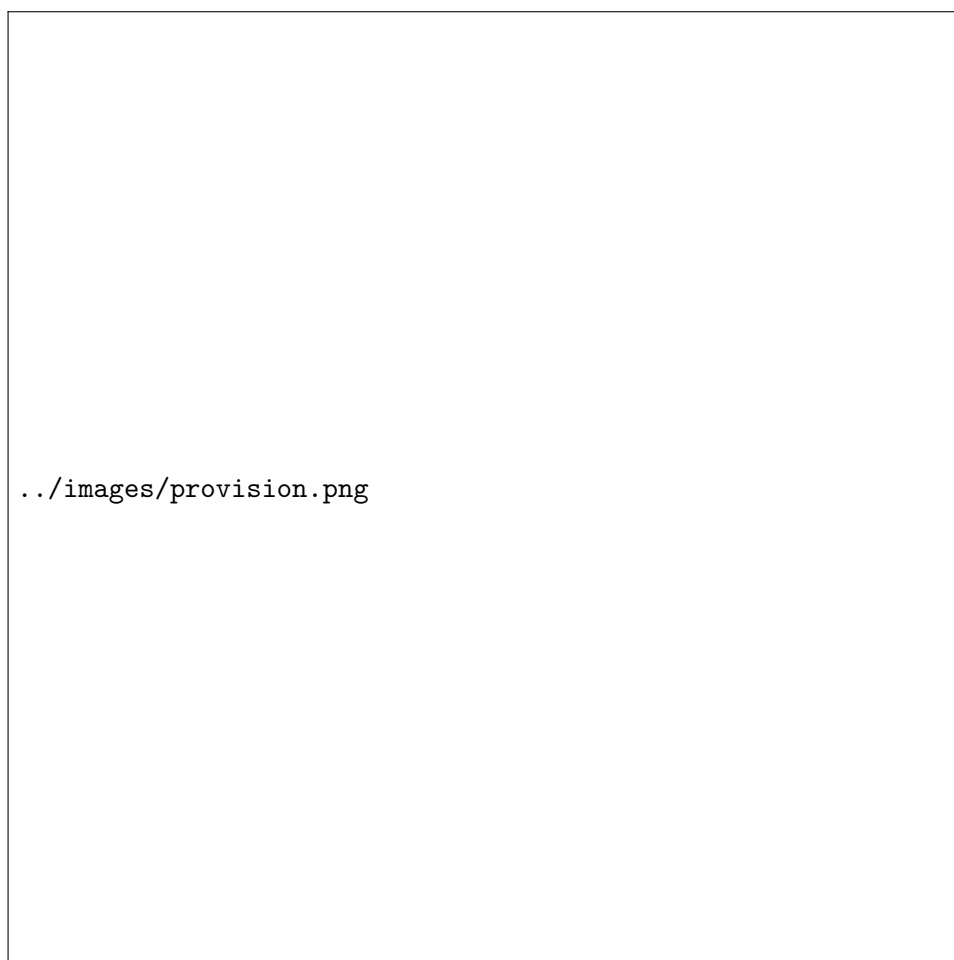


Figura 7.11: Prueba de provisionamiento con Ansible

7.6. Prueba de carga

Una vez que ya hemos realizado las pruebas de software que hemos considerado oportunas, también tenemos que pasar pruebas a la infraestructura como tal para analizar su rendimiento. En este tipo de pruebas entra en juego tanto el aspecto software como hardware con el objetivo de obtener una medición del rendimiento de nuestra aplicación.

7.6.1. Métricas y parámetros que afectan al rendimiento

Para comparar las prestaciones de la aplicación debemos tener en cuenta los siguientes criterios:

- El objetivo de esta prueba es medir las prestaciones del servidor generado por **Express** para dar servicio a la aplicación del portal de transparencia bajo unas condiciones que nos aporte un análisis neutro del rendimiento del mismo.
- La herramienta que se usará para realizar estas mediciones es **ApacheBench**.
- Los parámetros que se considerarán los parámetros usados en la herramienta: el número de peticiones que se realizan al servidor y el nivel de concurrencia con el que se realizan las peticiones.
- Los valores de estos parámetros irán modificándose para tener unos resultados más completos ante las diferentes cargas de trabajo que soportará el servidor.

El hardware y el software del sistema serán los siguientes:

- **Hardware:**
 - Procesador: Intel Pentium Dual CPU E2180 @ 2.00GHz
 - Placa base: MSI MS-7255
 - Chipset: VIA P4M900
 - Memoria: 3GB (2+1 DIMM DDR2)
 - Disco: Maxtor 6Y160P0 160GB 7200RPM
 - Tarjeta gráfica: ATI Radeon X300SE 325MHz 128MB
 - Red: Realtek RTL8100C 100Mbps
- **Software:**
 - Sistema operativo: Ubuntu 14.04.1 LTS i686 GNU/Linux
 - Kernel: Linux 3.13.0-35-generic
 - Sistema de archivos: ext4

7.6.2. Técnicas de evaluación, carga de trabajo y diseño de experimentos

Para evaluar el rendimiento de la aplicación vamos a realizar benchmark hacia la aplicación en ejecución para poder realizar una evaluación sobre su comportamiento bajo diferentes cargas de trabajo. El programa con el que vamos a hacer las pruebas es el ya mencionado **ApacheBench**, que nos permitirá realizar de forma sencilla pruebas de rendimiento a cualquier servidor, sea cual sea el lenguaje en el que esté realizado.

Según la información de registros de acceso del servidor en los últimos **6 meses** el número de peticiones de páginas del portal ha sido de **2.388 peticiones**, lo que sería aproximadamente **13 peticiones/día**. Para realizar los diferentes tests se realizará un **número de conexiones** variables al servidor (**30, 50 y 100**) con diferentes **niveles de concurrencia** en función del total de conexiones (**25 %, 50 % y 75 %**). Los números de conexiones para las pruebas han sido elegidos para evaluar como se comportaría el servidor ante un gran aumento de actividad en el mismo en línea con el nivel de conexiones que se producen en la actualidad.

De entre las peticiones al portal, la página que ha recibido un mayor número de peticiones es la página de **Personal**, por lo que el experimento a realizar consistirá en realizar peticiones de la página de **Personal** a la aplicación. Este experimento nos permitirá comprobar como se comportará la aplicación en diferentes situaciones con diferentes niveles de carga y la repercusión en su rendimiento ante estas diferentes pruebas, viendo por ejemplo si fuera necesario buscar una forma de balancear la carga del servidor.

En total el experimento constará de 9 pruebas, y a su vez cada una de estas pruebas se repetirá 10 veces consecutivas para así asegurarnos que los resultados son legítimos y no producto de sucesos fortuitos. Los resultados mostrados serán la media de esas 10 ejecuciones y su desviación estándar, que nos permitirá saber como de válidos podemos considerar los valores medios obtenidos. Estos valores promedios se introducirán en un gráfico para comparar los cambios que se producen con diferentes números de conexiones y mismos porcentaje de concurrencia. Las variables respuestas a tener en cuenta para el estudio serán:

- Tiempo de ejecución.
- Solicitudes por segundo.
- Tiempo por solicitud concurrente.
- Velocidad de transferencia.

El único factor a considerar para el experimento será, como hemos dicho, las respuestas del servidor de la aplicación desarrollada del portal de transparencia UGR **Transparente**. Cuando se disponga de los resultados de todas las pruebas, se procederá a analizar e interpretar los resultados.

7.6.3. Presentación de los resultados

Enumeración de las pruebas a realizar en el experimento:

- **Prueba 1:** 30 solicitudes, concurrencia del 25 % (8).
- **Prueba 2:** 30 solicitudes, concurrencia del 50 % (15).
- **Prueba 3:** 30 solicitudes, concurrencia del 75 % (23).
- **Prueba 4:** 50 solicitudes, concurrencia del 25 % (13).
- **Prueba 5:** 50 solicitudes, concurrencia del 50 % (25).
- **Prueba 6:** 50 solicitudes, concurrencia del 75 % (38).
- **Prueba 7:** 100 solicitudes, concurrencia del 25 % (25).
- **Prueba 8:** 100 solicitudes, concurrencia del 50 % (50).
- **Prueba 9:** 100 solicitudes, concurrencia del 75 % (75).

Tiempo de ejecución

| Tiempo de ejecución | | | |
|---------------------|------------------|------------|------------|
| N.º conexiones | Concurrencia (%) | Tiempo (s) | Desviación |
| 30 | 25 | 8,823 | 0,199 |
| 30 | 50 | 7,552 | 0,214 |
| 30 | 75 | 6,827 | 0,970 |
| 50 | 25 | 10,215 | 0,248 |
| 50 | 50 | 12,506 | 0,299 |
| 50 | 75 | 13,959 | 0,217 |
| 100 | 25 | 26,966 | 1,781 |
| 100 | 50 | 22,850 | 3,845 |
| 100 | 75 | 24,473 | 3,845 |

Tabla 7.1: Resultados de tiempo de ejecución



Figura 7.12: Gráfico de tiempo de ejecución

Solicitudes por segundo

| Solicitudes por segundo | | | |
|-------------------------|------------------|---------------|------------|
| N.º de conexiones | Concurrencia (%) | Solicitudes/s | Desviación |
| 30 | 25 | 3,402 | 0,078 |
| 30 | 50 | 3,976 | 0,113 |
| 30 | 75 | 4,470 | 0,531 |
| 50 | 25 | 4,898 | 0,118 |
| 50 | 50 | 4,000 | 0,095 |
| 50 | 75 | 3,583 | 0,056 |
| 100 | 25 | 3,728 | 0,292 |
| 100 | 50 | 4,496 | 0,708 |
| 100 | 75 | 4,197 | 0,703 |

Tabla 7.2: Resultados de solicitudes por segundo



Figura 7.13: Gráfico de solicitudes por segundo

Tiempo por solicitud concurrente

| Tiempo por solicitud concurrente | | | |
|----------------------------------|------------------|-------------|------------|
| N.º de conexiones | Concurrencia (%) | Tiempo (ms) | Desviación |
| 30 | 25 | 294,087 | 6,645 |
| 30 | 50 | 251,733 | 7,129 |
| 30 | 75 | 227,563 | 32,347 |
| 50 | 25 | 204,296 | 4,959 |
| 50 | 50 | 250,128 | 5,990 |
| 50 | 75 | 279,178 | 4,346 |
| 100 | 25 | 269,658 | 17,811 |
| 100 | 50 | 228,497 | 38,451 |
| 100 | 75 | 244,729 | 38,449 |

Tabla 7.3: Resultados de tiempo por solicitud concurrente



Figura 7.14: Gráfico de tiempo por solicitud concurrente

Velocidad de transparencia

| Velocidad de transferencia | | | |
|----------------------------|------------------|------------------|------------|
| N.º de conexiones | Concurrencia (%) | Velocidad (KB/s) | Desviación |
| 30 | 25 | 747,735 | 17,116 |
| 30 | 50 | 873,789 | 24,798 |
| 30 | 75 | 982,389 | 116,778 |
| 50 | 25 | 645,866 | 15,498 |
| 50 | 50 | 527,515 | 12,561 |
| 50 | 75 | 472,469 | 7,339 |
| 100 | 25 | 245,783 | 19,227 |
| 100 | 50 | 296,416 | 46,684 |
| 100 | 75 | 276,707 | 46,384 |

Tabla 7.4: Resultados de velocidad de transferencia



Figura 7.15: Gráfico de velocidad de transferencia

7.6.4. Análisis e interpretación de los resultados

Tiempo de ejecución

- Para un número de 30 conexiones (poco más de la media de conexiones diarias al portal) la concurrencia no afecta al tiempo de respuesta de la aplicación.
- Cuando el número de conexiones aumenta hasta 50, vemos como en este el nivel de concurrencia sí empieza a afectar al tiempo de respuesta de la aplicación de forma negativa.
- En la última prueba con un número de conexiones bastante más elevado, vemos en los tiempos de ejecución que el comportamiento es más aleatorio. Si nos fijamos en las desviaciones éstas han crecido bastante en comparación con las pruebas anteriores, por lo que aparentemente la aplicación empieza a soportar cierta sobrecarga.

Solicitudes por segundo

- En relación directa con los resultados descritos en el apartado anterior vemos que para 30 conexiones, al igual que el tiempo de respuesta bajaba según el nivel de concurrencia, ahora el número de solicitudes respondidas por segundo aumentan; el comportamiento que cabría esperar.
- En el caso de 50 conexiones, de igual forma, cuando el nivel de concurrencia sube, el número de solicitudes respondidas baja.
- Para la prueba de 100 conexiones volvemos a tener una situación similar a la anterior, donde aunque en esta caso las desviaciones no son tan altas, no se puede concretar una tendencia en el rendimiento de la aplicación.

Tiempo por solicitud concurrente

- Se produce la misma situación, para 30 conexiones el tiempo necesario para resolver una solicitud recurrente va disminuyendo según aumenta la concurrencia en las conexiones. A destacar que en esta prueba el valor de la desviación crece en gran cantidad, por lo que aunque se puede encontrar una tendencia general de mejor en el rendimiento los tiempos para resolver las solicitudes no son tan homogéneos.
- En el caso de las 50 conexiones también se produce la misma situación a las pruebas anteriores: según aumenta la concurrencia aumenta el tiempo necesario para responder una solicitud concurrente, lo que indica que el rendimiento ha bajado.
- De igual forma, para 100 conexiones volvemos a encontrar sin un patrón claro de comportamiento, pero si podemos destacar que las desviaciones son enormes.

Velocidad de transferencia

- Mismo patrón para la prueba con 30 conexiones. Habíamos visto que según aumenta la concurrencia, disminuye el tiempo de ejecución y aumentan el número de solicitudes por segundo respondidas o lo que es lo mismo, aumenta el rendimiento; así que como podríamos intuir y con estos datos comprobar, también aumenta la cantidad de información transferida. Nos encontramos en algunos casos con desviaciones enormes, pero eso podemos atribuirlo a la propia naturaleza inestable de las conexiones de red.

- Como se ha venido produciendo, en el caso de la prueba con 50 conexiones el rendimiento ha ido bajando según ha ido aumentando el porcentaje de concurrencia; esto también se produce en este caso: según aumentaba la concurrencia, disminuía la velocidad de transferencia.
- Por último, en la prueba con 100 conexiones esta es la prueba que ha tenido resultados más igualados (y además muy bajos en comparación) entre los diferentes niveles de concurrencia, lo que nos vuelve a hacer suponer que el servidor está sobrecargado.

7.6.5. Conclusiones sobre los resultados

Una vez analizados los resultados de las pruebas realizadas la conclusión a la que podríamos llegar es que la aplicación puede aguantar perfectamente la carga de trabajo que tiene en la actualidad (alrededor de 13 peticiones al día), y que tampoco no habría ningún problema en que esta se duplicara como hemos visto en los resultados de las pruebas de 30 conexiones. El problema aparecería si este volumen de visitas siguiera creciendo, ya que como hemos visto en las pruebas de 50 conexiones, en cuanto empieza a aumentar la concurrencia de las solicitudes el rendimiento de la aplicación cae, llegando a un punto de las pruebas de 100 conexiones en el la aplicación comienza a estar sobrecargada.

Viendo el volumen de visitas actual, es difícil que a corto plazo se llegarán a producir este tipo de problemas de saturación, sin embargo, en futuro se deberían aplicar técnicas de balanceo de carga sobre la aplicación usando algún otro módulo de `Node.js` como puede ser `node-http-proxy`, cambiar la estructura de la aplicación para que esta no tenga problemas a unos niveles altos de carga de trabajo, o incluso cambiar el sistema operativo por otro que pudiera dar un mayor rendimiento en entornos de servidores.

7.7. Acceso web

El resultado final es que el portal de transparencia sigue siendo accesible, y aunque visualmente no se aprecia ningún cambio, por debajo su funcionamiento y su desarrollo han cambiado completamente a como se ha descrito a lo largo de todo el documento.

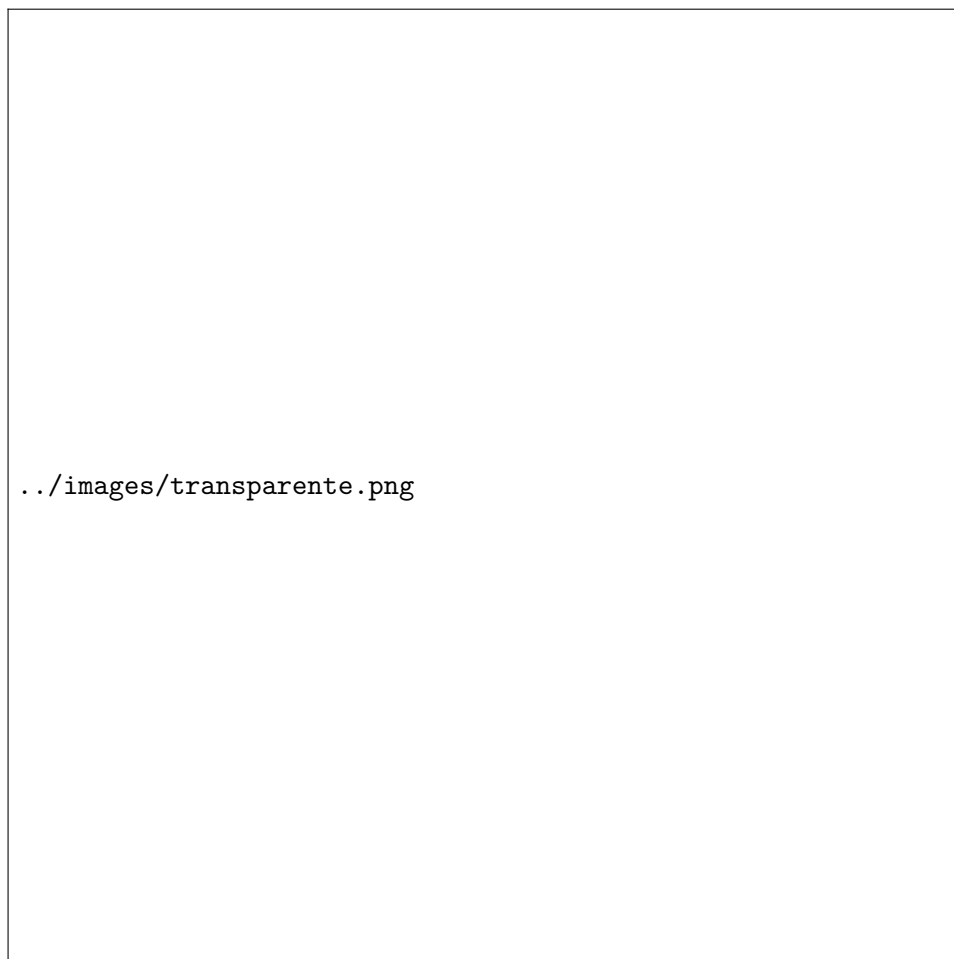


Figura 7.16: Visualización del portal de transparencia

Capítulo 8

Conclusiones y trabajos futuros

Después de realizar este proyecto la principal conclusión a la que se puede llegar es que si bien un desarrollado menos *instrumentado* puede ser muy rápido, el dividir el desarrollo en diferentes etapas (con herramientas que a su vez tienen sus propios sistemas de control) harán del desarrollo una tarea más fácil y robusta.

Este era uno de los principales problemas del estado inicial del portal, al no encontrarse por ejemplo implementados ningún tipo de tests unitarios se producían varios errores por causas desconocidas que eran difíciles de situar en el código de la aplicación.

Uno de los principales problemas, es que para esta metodología de desarrollo las fases de planificación y análisis pueden ser más difíciles de plantear, ya que se basa en un desarrollo continuo en el que todo elemento a desarrollar se decide según la necesidad y generalmente con un tiempo de antelación bastante corto.

Como ventaja tenemos que hay un gran cantidad de herramientas para gestionar las diferentes etapas, pudiendo elegir las que consideremos precisas en un determinado momento ya sea por restricciones del desarrollo o por simple comodidad. Por ejemplo: los test unitarios se podrían haber pasado con **Unit JS**, la integración continua con **Jenkins** y el provisionamiento con **Chef**; así aunque los procedimientos fueran distintos, el resultado hubiera sido el mismo.

En cuanto a trabajos futuros relacionados con el proyecto algunas posibilidades serían:

- Realizar un instalación personalizada de **CKAN** en un versión actual de **Ubuntu** (actualmente solo existe una instalación oficial para la versión 12.04) para poder desarrollar plugins propios con el fin de realizar tareas que se puedan considerar interesantes; principalmente interesa que los datos se pudieran obtener directamente desde el propio **CKAN** y no se tuvieran que introducir manualmente mediante archivos *JSON*. Este era uno de los objetivos del proyecto, pero por su dificultad para realizarlo en el periodo de tiempo establecido se determinó que fuera un objetivo secundario; finalmente no ha podido ser cumplido por dicho motivo, por lo que queda como una posible extensión del trabajo inicial.
- Sería conveniente hacer que en general que el portal sea más dinámico porque actualmente todas las páginas son estáticas, si eso se mantiene así en unos cuantos años cuando la cantidad de datos disponibles aumente considerablemente, la navegabilidad por el portal empeorará considerablemente. Sería necesario implementar un forma de que se puedan seleccionar para visualizar solo los datos que se deseen.
- Analizando los resultados de las pruebas de carga observamos que cuando el número de peticiones aumentaba en gran cantidad con respecto a la carga de trabajo actual, o aunque las peticiones no aumentaran tanto, pero si aumentase el nivel de concurrencia la aplicación se saturaba. Esto debería ser solucionado mediante cambios en la estructura de la aplicación o uso de balanceo de carga.

Glosario de términos

Backend: es el motor de una aplicación, se encarga de realizar las funciones en segundo plano que se encargan de que la aplicación funcione.

Balanceo de carga: técnica de configuración de servidores que permite que la carga de trabajo total se reparte entre varios de ellos para que no disminuya el rendimiento general de la infraestructura.

Build: versión compilada de un programa lista para ser ejecutada.

CKAN: plataforma de código abierto para el almacenamiento de datos que permite que estos sean publicados y compartidos fácilmente.

SSH (Secure SHell): protocolo que permite conectarse a máquinas remotas mediante conexiones seguras de red.

Dataset: conjunto de datos que representa las características de un modelo de información.

Datos abiertos: información de interés general que es publicada de forma que sea fácilmente accesible por cualquier persona o institución interesada.

Desarrollo colaborativo: metodología de desarrollo de software en el que todo el código está disponible públicamente, permitiendo que cualquier persona pueda colaborar en el proyecto. **GitHub** es una de las plataformas de desarrollo colaborativo más utilizadas.

Despliegue automático: proceso que permite que una aplicación sea instalada en una infraestructura objetivo sin tener que realizar local y manualmente todos los pasos necesarios para efectuar la instalación.

DevOps (DEvelopment and OPerationS): metodología de desarrollo ágil en la que todo el progreso es secuencial sin diferenciar entre el desarrollo y la administración del propio software.

Frontend: es la interfaz de la aplicación, es la parte de la aplicación que el usuario utiliza para comunicarse con la misma.

HTML (HyperText Markup Language): lenguaje de marcado que se utiliza para la realización de páginas web.

Integración continua: proceso que consiste en realizar las pruebas diseñadas para una aplicación cada vez que se realizan cambios en la misma, todo esto con el fin de encontrar lo más rápidamente posiblemente posibles errores producidos en la actualización.

JavaScript: lenguaje de programación orientado a objetos interpretado que se utiliza principalmente para cargar programas desde el lado del cliente en los navegadores web.

JSON (JavaScript Object Notation): formato de texto plano usado para el intercambio de información, independientemente del lenguaje de programación.

LaTeX: sistema de composición de documentos que permite crear textos en diferentes formatos (artículos, cartas, libros, informes...) obteniendo una alta calidad en los documentos generados.

Módulo: fragmento de un programa desarrollado para realizar una tarea específica.

MongoDB: sistema de base de datos NoSQL de código abierto orientado a documentos.

Node.js: entorno de programación asíncrono orientado a eventos con funcionamiento asíncrono basado en JavaScript utilizado principal para ejecutar programas desde el lado del servidor.

NoSQL: sistema de gestión de base de datos que usan diversas estructuras para organizar los datos (documentos, grafos, pares clave/valor, orientación a objetos...) cuya principal ventaja es el gran rendimiento en la realización de tareas de obtención y almacenamiento de datos.

NPM: sistema de gestión de paquetes usado por `Node.js`.

Portal de transparencia: sitio web cuya función es la publicación de datos abiertos.

Programación asíncrona: paradigma de programación en el que las operaciones pueden ejecutarse de forma independiente a la secuencia en la que son llamadas.

Provisionamiento: proceso que se encarga de preparar una infraestructura con todos los recursos software necesarios para poder desplegar una aplicación en ella.

Pull: acción consistente en obtener los cambios de un proyecto desde su repositorio de `GitHub`.

Pull request: acción consistente en fusionar los cambios realizados en un repositorio desde otro repositorio que hubiera hecho una copia del proyecto con el propio repositorio original.

Push: acción consiste en publicar los cambios hechos en un proyecto a su repositorio de `GitHub`.

RSA: sistema criptográfico de clave pública usado para la seguridad de transferencia de datos.

Sistema de control de versiones: aplicación que permite gestionar los cambios que se producen durante el desarrollo de un proyecto pudiendo así llevar un histórico de los mismos, ver las diferencias introducidas ó deshacer dichos cambios, entre otras operaciones. `Git` es un ejemplo de sistema de control de versiones y uno de los más usados en la actualidad.

Software libre: software cuya licencia permite que este sea usado, copiado, modificado y distribuido libremente según el tipo de licencia que adopte.

TDD (Test-Driven Development): metodología de desarrollo de software en el que durante la fase inicial se desarrollan las pruebas que el software debe superar y posteriormente se desarrolla el software para que pase dichas pruebas.

Test de cobertura: comprobación de los resultados obtenidos por los tests unitarios que permiten conocer el porcentaje total del código del software que esta cubierto por algún tipo de prueba.

Tests unitarios: prueba de cada una de las funcionalidades implementadas en una aplicación destinadas a comprobar que la funcionalidad se desempeña correctamente.

URL (Uniform Resource Locator): nombre y con un formato estándar que permite acceder a un recurso de forma inequívoca.

YAML (YAML Ain't Another Markup Language): formato de serialización de datos legible por humanos cuyo principal uso es el intercambio de información, aunque también es usado como formato de archivos de configuración.

Anexo. Realización del trabajo de fin de grado con licencia libre y formato abierto

Cuando empecé a trabajar en el proyecto ya había un trabajo previo, entre sus particularidades estaba que era un proyecto cuyo software estaba siendo directa y totalmente liberado en `GitHub` bajo una licencia libre (concretamente **GNU General Public License v3**) según iba siendo desarrollado.

Partiendo de esta base mi tutor me propuso que realizara toda la documentación del proyecto bajo esta misma metodología. Al principio se me hacía raro concebir el desarrollo de un documento en un sitio generalmente orientado al desarrollo de software, aunque seguramente el sistema de control de versiones sería algo a lo que podría sacarle mucha ventaja.

Ya que el proyecto se estaba desarrollando en una licencia libre decidí que la documentación debía seguir también totalmente ese camino, por lo que el uso de una licencia **Creative Commons** era la que mejor se adaptaba a mi objetivo, además, permite que este trabajo fluya en ese mismo camino del software libre: puedes leerlo, puedes copiarlo, puedes modificarlo y finalmente, puedes seguir distribuyéndolo; lo único que tienes que hacer es reconocer mi autoría original.

El siguiente paso natural en la evolución de este proyecto es que la documentación tenía también que ser realizada en un formato abierto, y hablando de documentación, el formato técnico por excelencia es \LaTeX . Este sistema de composición de textos puede ser bastante complicado de usar al principio, sin embargo una vez acostumbrado llegado a ser mucho más cómodo de usar que los típicos procesadores de texto, sobretodo debido a que no es necesario preocuparse de revisar continuamente que los estilos o el formato se han estropeado; una vez que los estilos y los formatos están definidos tenemos

la seguridad de estos no se van a “estropear”, por lo que solo deberemos centrarnos en el contenido del documento que es lo realmente importante.

Según iba desarrollando la documentación me acostumbré a plantear las tareas y objetivos como si de un desarrollo de software cualquier se tratara, así podría aprovechar las ventajas que me proporcionaba **GitHub** en este ámbito. Organizarme las tareas a base de “*issues*” a los que le asignaba una etiqueta según la finalidad de la misma, pudiendo diferenciar entre tareas para añadir un nuevo contenido o corregir contenido en el que hubiera algún error. Este mismo sistema también me servía para estar en contacto con mi tutor y que pudiera seguir el desarrollo del mismo, pudiendo comentarme en cualquier momento las observaciones que considerara pertinentes, pero no solo él, al ser una plataforma pública cualquier persona podría interesada podría hacer comentarios o sugerencias para mejorar el proyecto, haciendo que aumente considerablemente la riqueza de contenidos que a este proyecto podrían ser añadidos.

Si hubiera elegido seguir una metodología más tradicional para desarrollar este documento, en diversos momentos su avance me hubiera sido más rápido y fácil, pero sin lugar a dudas no hubiera sido una experiencia tan enriquecedora en la que habría aprendido tanto como he aprendido siguiendo esta forma de trabajar.

Bibliografía

Artículos sobre las metodologías y procesos utilizados en el proyecto:

- [1] “Continuous Delivery 101: Automated Deployments”. Martin Etmajer, 18/11/2014. <http://apmblog.dynatrace.com/2014/11/18/continuous-delivery-101-automated-deployments/>
- [2] “¿Qué es DevOps?”. Patricio Bruna, 23/10/2013. <http://blog.itlinux.cl/blog/2013/10/23/que-es-devops/>
- [3] “The Beginner’s Guide to Unit Testing: What Is Unit Testing?”. Tom McFarlin, 19/06/2012. <http://code.tutsplus.com/articles/the-beginners-guide-to-unit-testing-what-is-unit-testing--wp-25728>
- [4] “Travis-CI: What, Why, How”. Sayanee Basu, 18/09/2013. <http://code.tutsplus.com/tutorials/travis-ci-what-why-how--net-34771>
- [5] “What Is DevOps?”. Carlos Gomez, 07/12/2011. <http://theagileadmin.com/what-is-devops/>
- [6] “Tools for Unit Testing and Quality Assurance in Node JS”. Adam Duncan, 29/04/2014. <http://www.clock.co.uk/blog/tools-for-unit-testing-and-quality-assurance-in-node-js>
- [7] “A Closer Look At Personas: What They Are And How They Work (Part 1)”. Shlomo Goltz, 06/08/2014. <http://www.smashingmagazine.com/blog/2014/08/06/a-closer-look-at-personas-part-1/>
- [8] “Persona (user experience)”. Wikipedia, última edición: 26/05/2015. [https://en.wikipedia.org/wiki/Persona_\(user_experience\)](https://en.wikipedia.org/wiki/Persona_(user_experience))
- [9] “Testing your frontend JavaScript code using mocha, chai, and sinon”. Nicolas Perriault, 23/07/2013. <https://nicolas.perriault.net/code/2013/testing-frontend-javascript-code-using-mocha-chai-and-sinon/>

Páginas de consulta sobre licencias y APIs del software utilizado

- [10] Creative Commons. <http://creativecommons.org/licenses/>
- [11] CKAN. <http://docs.ckan.org/en/ckan-2.2/api.html>
- [12] Flightplan. <https://www.npmjs.com/package/flightplan>
- [13] Forever. <https://github.com/foreverjs/forever>
- [14] Istanbul. <https://github.com/gotwarlost/istanbul>
- [15] Jade. <http://jade-lang.com/api/>
- [16] Wikibooks (LaTeX). <https://en.wikibooks.org/wiki/LaTeX>
- [17] Mocha. <http://mochajs.org/>
- [18] Node.js. <https://nodejs.org/api/>
- [19] NPM. <https://docs.npmjs.com/>
- [20] Should.js. <http://shouldjs.github.io/>
- [21] Supertest. <https://github.com/visionmedia/supertest>
- [22] TeXdoc (LaTeX). <http://www.texdoc.net/>
- [23] Travis CI. <http://docs.travis-ci.com/>
- [24] Underscore.js. <http://underscorejs.org/>

Otro material

- Diversas consultas puntuales al sitio Stack OverFlow.
- Material docente de las asignaturas **Fundamentos de Ingeniería del Software**, **Ingeniería de Servidores e Infraestructura Virtual** impartidas en **Grado en Ingeniería Informática** en la **Universidad de Granada**.

