Trabajo de Fin de Grado

Grado en Ingeniería Informática

Un portal de transparencia para datos libres

Autor
Germán Martínez Maldonado

Tutor
Juan Julián Merelo Guervós

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

—

Granada, 17 de junio de 2015

Un portal de transparencia para datos libres

Germán Martínez Maldonado

Resumen

Palabras clave: software libre, transparencia, datos abiertos, sistema de control de versiones, aprovisionamiento, tests, integración continua, despliegue automático

Se pretende desarrollar una plataforma para la transparencia que sea desplegable en un infraestructura física o virtual basándose en el trabajo realizado en el portal UGR Transparente, respaldada por los datos abiertos publicados en la plataforma OpenData UGR.

Este desarrollo tendrá como objetivo que el resultado sea una plataforma totalmente basada en el software libre que se pudiera adaptar con facilidad a otro organismo, teniendo en cuenta funcionalidades y requisitos obligatorios, además de aspectos de accesibilidad y escalabilidad.

La herramientas básicas a usar serán un sistema de control de versiones y un sistema de desarrollo colaborativo que albergue el proyecto, aque además use dicho sistema de control de versiones. También se quiere que la plataforma se pueda desarrollarr y administrar simultáneamente, por lo que para convertir todo esto es un proceso más ágil e ininterrumpido se usarán otras herramientas que permitan lo siguiente:

- Realizar aprovisionamiento de las infraestructuras.
- Validación mediante tests unitarios.
- Comprobación de conflictos mediante integración continua.
- Actualizaciones mediante despliegue automático.

A transparency portal for open data

Germán Martínez Maldonado

Extended abstract

Keywords: free software, transparency, opendata, version control system, provisioning, tests, continuous integration, automated deployment

It is intended to develop a platform for transparency that is deployable on a physical or virtual infrastructure based on work done on the site UGR Transparente, backed by open data published in the Open Data UGR.

This development aims that the result is a platform completely based on free software that could be easily adapted to another organization, taking into consideration features and mandatory requirements, as well as issues of accessibility and scalability. The platform which presents the data will develop in Node.js, which is a programming environment that operates at runtime based on the Google's V8 JavaScript engine; also will use Express for web application development and Jade to generate HTML files based on templates. Moreover, the platform that contains the data is based on CKAN, a open-source data portal platform developed by The Open Knowledge Foundation, a not-for-profit organisation that promotes creating and sharing knowledge freely.

Being free software, this development is not limited to the working group that started the work, but rather is focused on that anyone can make their contribution to the project; therefore, for all this can be handled is necessary a version control system, in this case Git will be used, which is practically a standard in this area. In addition, it will also be used a recognized and open collaborative based development platform as is GitHub, which is also integrated with Git, providing ease in the development and distribution of work, because any who access the project repository can freely copy it for its open license.

An important point to facilitate system administration is the provisioning. Provisioning a machine, as the name suggests, is to provide to the machine all the resources needed for its performance, in our case we refer to all the software necessary for that the platform developed works properly in such infrastructure. In the case of transparency portal, as the entire project is hosted on GitHub, we can set up a utility like Ansible to download the project and then install on the machine all the necessary software so that the platform can function properly. In contrast, such as the open data portal

is not a own development, their provisioning will consist of installing and customizing CKAN for the University of Granada.

During the development of any software application, new features are introduced to the software gradually, so must be ensured that the new features do not compromise the overall stability of the project. For this purpose the unit tests are written, which could be considered as small programs within the system that handles check by assertions and behavior patterns that all elements operate as they should. To verify that our unit tests are sufficient also need to pass a coverage test that tells us that all the functionality of our platform are properly validated by the corresponding unit test. There are many libraries that allow both actions, but for the facility of work between them, unit test will be passed with Mocha and coverage test will be with Istanbul.

Once we have the tests written, we don't need concern ourselves with run them manually, we have the option of running in a external platform every time we make changes and directly get the results, this is continuous integration. Continuous integration will be made with Travis CI and the operation is very simple: every time we make a change in GitHub, Travis CI download the latest version of the code, builds it and passes the tests we have written, to finish by returning the results of the tests that will make us know whether changes have produced a conflict in the system.

The last thing to keep in mind is about deploying the changes automatically on our server, can be a tedious procedure having to manually access to the infrastructure of our platform every time that we want to apply the new changes we have made in the application, so we'll use the tool Flightplan for automatic deployment; only with specify our server as the target we can set a series of tasks that will make that automatically for the updates we're making become effective on the main machine.

By using these tools we are getting an application in which development and management are closely related and automated, which makes the application much more reliable and easy recovery in case of problems.

Yo, **Germán Martínez Maldonado**, 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 (*Un portal de transparencia para datos libres*) en la biblioteca del centro para que pueda ser consultada por las personas que lo deseen.

Fdo: Germán Martínez Maldonado

Granada, a 17 de junio de 2015

D. **Juan Julián Merelo Guervós**, Profesor del Departamento de Arquitectura y Tecnología de Computadores de la Universidad de Granada.

Informa:

Que el presente trabajo, titulado ***Un portal de transparencia para datos libres***, ha sido realizado bajo su supervisión por **Germán Martínez Maldonado**, 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 17 de junio de 2015.

El tutor:

Juan Julián Merelo Guervós

Agradecimientos

[Poner aquí agradecimientos]

Índice general

1. Introducción **1**

2. Objetivos **3**
2.1. Alcance de los objetivos 4
2.2. Interdependencia de los objetivos 4

3. Planificación **5**
3.1. Fases y entregas 5
3.1.1. Fases 5
3.1.2. Lista de entregas 5
3.2. Estructura de Descomposición del Trabajo 6
3.3. Lista de actividades 7
3.4. Recursos humanos 8
3.5. Presupuesto 8
3.6. Temporización 8

4. Análisis **11**
4.1. Análisis de requisitos 11
4.1.1. Descripción de los actores 11
4.1.2. Requisitos Funcionales 11
4.1.3. Requisitos no Funcionales 12
4.1.4. Requisitos de Información 13
4.2. Modelos de casos de uso 13
4.2.1. Descripción básica de actores 14
4.2.2. Descripción casos de uso 14
4.3. Diagrama de paquetes 24
4.4. Diagramas de casos de uso 24
4.5. Diagramas de actividad 26
4.6. Diagrama conceptual 35

5. Diseño **37**
5.1. Diseño general del portal 37
5.2. Diseño de una aplicación con desarrollo colaborativo 39
5.3. Diseño de los tests unitarios y test de cobertura 40

5.4. Diseño de la integración continua 41
5.5. Diseño del despliegue automático 42
5.6. Diseño del aprovisionamiento 43

6. Implementacion **45**
6.1. Uso de JSON como origen de datos 45
6.2. Tests unitarios y de cobertura 49
6.3. Integración continua 50
6.4. Despliegue automático 50
6.5. Aprovisionamiento 51

Índice de figuras

3.1. Diagrama de Estructura de Descomposición de Trabajo . . . 9
3.2. Temporización de las tareas 10
3.3. Diagrama de Gantt 10

4.1. Diagrama de paquetes 24
4.2. Diagrama de casos de uso: paquete Administración portal . . 24
4.3. Diagrama de casos de uso: paquete Acceso información 25
4.4. Diagrama de casos de uso: paquete Pruebas de software . . . 25
4.5. Diagrama de casos de uso: paquete Configuración automática 26
4.6. Diagrama de actividad CU-1. Inicio automático del servidor del portal 26
4.7. Diagrama de actividad CU-2. Consultar información de Administración 27
4.8. Diagrama de actividad CU-3. Consultar información de Docencia 28
4.9. Diagrama de actividad CU-4. Consultar información de Gestión e Investigación 29
4.10. Diagrama de actividad CU-5. Consultar información de Normativa Legal 30
4.11. Diagrama de actividad CU-6. Realizar tests unitarios 31
4.12. Diagrama de actividad CU-7. Realizar test de cobertura . . . 31
4.13. Diagrama de actividad CU-8. Usar integración continua . . . 32
4.14. Diagrama de actividad CU-9. Usar despliegue automático . . 33
4.15. Diagrama de actividad CU-10. Usar aprovisionamiento 34
4.16. Diagrama conceptual 35

Índice de tablas

4.1. Curso normal de CU-1. Inicio automático del servidor del portal 15
4.2. Curso alterno de CU-1. Inicio automático del servidor del portal 15
4.3. Curso normal de CU-2. Consultar información de Administración 16
4.4. Curso normal de CU-3. Consultar información de Docencia . 17
4.5. Curso normal de CU-4. Consultar información de Gestión e Investigación 18
4.6. Curso normal de CU-5. Consultar información de Normativa Legal 19
4.7. Curso normal de CU-6. Realizar tests unitarios 19
4.8. Curso alterno de CU-6. Realizar tests unitarios 20
4.9. Curso normal de CU-7. Realizar test de cobertura 20
4.10. Curso alterno de CU-7. Realizar test de cobertura 20
4.11. Curso normal de CU-8. Usar integración continua 21
4.12. Curso alterno de CU-8. Usar integración continua 21
4.13. Curso normal de CU-9. Usar despliegue automático 22
4.14. Curso alterno de CU-9. Usar despliegue automático 22
4.15. Curso normal de CU-10. Usar aprovisionamiento 23
4.16. Curso alterno de CU-10. Usar aprovisionamiento 24

Índice de fragmentos de código

6.1. Archivo `JSON` con informacion de personal 46
6.2. Archivo `cargar.js` 47
6.3. Archivo `app.js` 48
6.4. Archivo `administracion.js` 48
6.5. Scripts de inicio y detención 48
6.6. Archivo `test.js` 49
6.7. Scripts de test 50
6.8. Archivo `JSON` con informacion de personal 50
6.9. Archivo `test.js` 50
6.10. Scripts de despliegue automático 51
6.11. Archivo de hosts 51
6.12. `Playbook` de `Ansible` 51

Capítulo 1

Introducción

En la actualidad, nuestro país al igual que muchos otros se rige por lo que se conoce como **"democracia"**. Si buscamos el significado de esta palabra, encontraremos que su definición es:

> Sistema político que defiende la soberanía del pueblo y el derecho del pueblo a elegir y controlar a sus gobernantes.
> (`http://www.oxforddictionaries.com/es/definicion/espanol/democracia`)

Aunque *"el derecho del pueblo a elegir"* es lo importante, para que esa elección pueda ser coherente, primero será el propio pueblo el que deberá tener la obligación a conocer lo que elige, y la única forma de conocer lo que se elige es a través de la transparencia. Por este motivo, el pueblo debe exigir transparencia en el funcionamiento de las instituciones públicas, y esas instituciones públicas siempre no tengan nada que esconder debería facilitar de buena fe (y no porque haya una ley que les obligue) todo la información que el ciudadano le requiera.

Desde que se aprobó la Ley de Transparencia en el año 2013, es obligatorio que todas las entidades públicas garanticen la transparencia en su actividad mediante la publicación de la información de la misma y faciliten el derecho de los ciudadanos al acceso de dicha información. Sin embargo, en el caso de las universidad públicas en particular, año y medio después de la aprobación de esta ley, podemos ver que son pocas las instituciones que tienen un portal de transparencia que realmente facilite datos útiles y cuya accesibilidad sea la adecuada para el acceso de todo tipo de población.

Estas inquietudes fueron las que hicieron en un primer momento que se comenzara con el desarrollo del Portal de Transparencia de la Universidad de Granada a principios del año pasado, cuyo desarrollo le fue encargado a la Oficina de Software Libre de la propia universidad y aproximadamente a los 7 meses de comenzar su desarrollo fue presentada una primera versión.

En febrero de este año entré como becario en la Oficina de Software Libre y me integré en el equipo de desarrollo del portal, siendo mis mayores contribuciones cambiar el origen de datos de la página para eliminar un error que se producía frecuentemente durante la navegación por el parte, y además, implantar una metodología de desarrollo DevOps. El funcionamiento de todos estos conceptos será ampliado en el capítulo 5 (Diseño) y en el capítulo 6 (Implementación), donde se explicará con más detalle en que consisten y como han sido integrados en el proyecto.

Además, antes de pasar a detalles tan técnicos, en los capítulos previos se explicarán otros aspectos del proyecto como son:

- En el capítulo 2 (Objetivos), detallar de forma algo más concreta los objetivos determinados que se quieren cumplir con este proyecto.
- En el capítulo 3 (Planificación), la planificación y desarrollo de cada una de las fases del proyecto .
- En el capítulo 4 (Análisis), especificar todos los requisitos que se necesitan cubrir con el software a desarrollar, así como describir cómo esos requisitos tienen que tomar forma en el desarrollo.

Una vez estén realizadas tanto la parte descriptiva como la parte de desarrollo, quedarán solo por añadir los apartados finales; el capítulo 7 (Pruebas) en el que se explicarán las diferentes pruebas a las que ha sido sometido el software desarrollado para comprobar su correcto funcionamiento, y el capítulo 8 (Conclusiones y trabajos futuros) en el que expondrán los conocimientos y cuestiones sacados del desarrollo del proyecto, además de todo el trabajo que podría quedar pendiente para ampliar el portal.

[REFERENCIAR TAMBIÉN APÉNDICES]

Capítulo 2

Objetivos

El objetivo de este proyecto es el de obtener un portal de transparencia para la Universidad de Granada basado en el software libre que además permita una metodología de desarrollo en el que continuamente se puedan añadir nuevas funcionalidades a la vez que se van testeando, para después de una fácil integración culminar con un despliegue automático. Este portal no almacenará los propios datos abiertos, si no que hará de presentación de los datos que estarán contenidos en otra plataforma destinada únicamente a dicho fin (OpenData UGR).

Un resumen de los principales objetivos a alcanzar son:

- **OBJ-1.** Promover la idea de que los desarrollos bajo software libre en general y en la administración pública en particular ayudan a generar confianza y demuestran compromiso con la transparencia.
- **OBJ-2.** Solucionar los problemas de generación de las tablas con los elementos de información.
- **OBJ-3.** Implementar en el portal UGR Transparente una metodología de desarrollo continuo que se base en el uso de tests unitarios para cada una de las funcionalidades que se vayan añadiendo, test de cobertura que evalúen los test unitarios, integración continua ante cambios introducidos, despliegue automático de las actualizaciones que se vayan produciendo y aprovisionamiento software para la infraestructura.

Además como objetivo secundario tendremos:

- **OBJ-4.** Estudiar la posibilidad de hacer una instalación totalmente personalizada del portal de datos de código abierto CKAN para los datos abiertos de la Universidad de Granada, además de un aprovisionamiento que permitiera que esta instalación se pudiera realizar automáticamente en una infraestructura.

Destacar en los aspectos formativos previos más utilizados para el desarrollo del proyecto los conocimientos sobre infraestructuras virtuales para el tema de gestión de configuraciones, ingeniería de software para el análisis del proyecto e ingeniería de servidores para la realización de pruebas desde el aspecto hardware.

2.1. Alcance de los objetivos

La aplicación resultante visualmente será idéntica que la versión actual de UGR Transparente, pero internamente habrá cambiado totalmente la metodología de desarrollo, pasando de un desarrollo sin control a uno totalmente controlado mediante pruebas unitarias e integración continua. Además los cambios ya no se tendrá que aplicar manualmente en el servidor, sino que se hará uso de un despliegue automático para tal fin.

Esta aplicación se usará de base en la Universidad de Granada, pero además el objetivo del desarrollo es que se pueda exportar para su uso en cualquier organización sin demasiada dificultad, de ahí que se vaya a estudiar también la posibilidad de personalización de CKAN, ya que esto permitiría generar una gran plataforma de liberación de datos totalmente personalizable según el ámbito y las necesidades.

2.2. Interdependencia de los objetivos

Todos los objetivos son independientes entre sí, pero el primer objetivo (**OBJ-1**) es el principal motivador de este proyecto, por lo que aún sin representar el desarrollo de ningún trabajo en concreto es el que va a escudar y avalar el desarrollo de los otros. En aspectos más relacionados con la realización del proyecto, el segundo objetivo (**OBJ-2**) es el que se solucionará de forma inmediata ya que impide el correcto funcionamiento del portal. El resto de objetivos serán tratados en mayor medida durante los capítulos de implementación y pruebas.

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 aprovisionamiento.
  - *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.

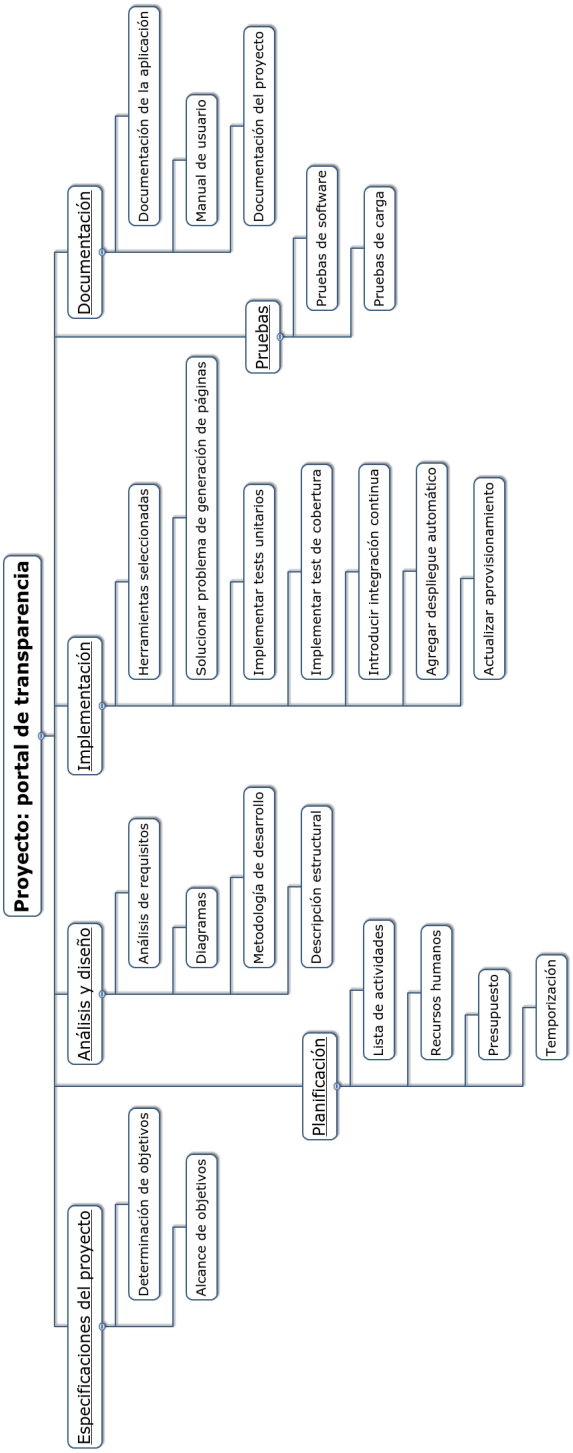

Figura 3.1: Diagrama de Estructura de Descomposición de Trabajo

Nombre	Fecha de inicio	Fecha de fin
Especificaciones del proyecto	16/02/15	18/02/15
Planificación	19/02/15	26/02/15
Análisis y diseño	27/02/15	16/03/15
Implementación	17/03/15	27/04/15
Pruebas	28/04/15	11/05/15
Documentación	12/05/15	25/05/15

Figura 3.2: Temporización de las tareas

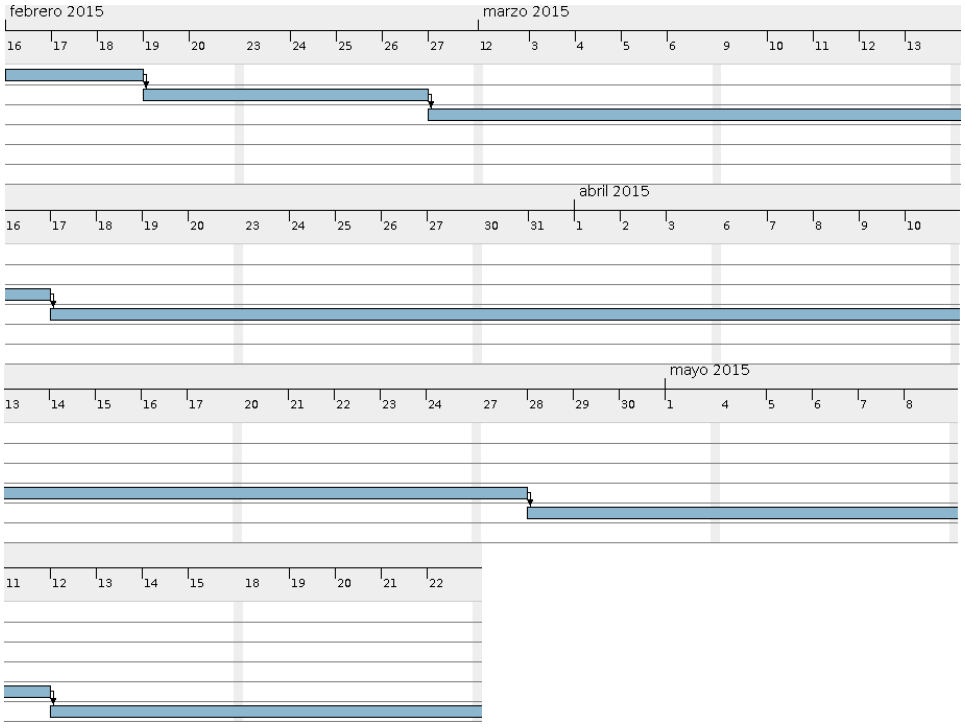

Figura 3.3: Diagrama de Gantt

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.

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 aprovisionamiento 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 aprovisionamiento.

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 lanzamos 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 aprovisionamiento 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 un 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 interes.

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: Germán Martínez Maldonado
  - 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: Germán Martínez Maldonado
  - 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 "Administraciónz consulta la información de sus subsecciones.

Curso normal			
Actor		Sistema	
1	Usuario: consulta la información de Administración publicada en el portal.		
		2	Se generan las tablas con todos los elementos de información de la sección Administración.
		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: Germán Martínez Maldonado
  - 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 "Docenciaz consulta la información de sus subsecciones.

Curso normal			
Actor		**Sistema**	
1	Usuario: consulta la información de Docencia publicada en el portal.		
		2	Se generan las tablas con todos los elementos de información de la sección Docencia.
		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: Germán Martínez Maldonado
  - 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 Investigaciónz consulta la información de sus subsecciones.

Curso normal			
Actor		**Sistema**	
1	Usuario: consulta la información de Gestión e Investigación publicada en el portal.		
		2	Se generan las tablas con todos los elementos de información de la sección Gestión e Investigación.
		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: Germán Martínez Maldonado
  - 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 Legalz consulta la información de sus subsecciones.

Curso normal			
Actor		**Sistema**	
1	Usuario: consulta la información de Normativa Legal publicada en el portal.		
		2	Se generan las tablas con todos los elementos de información de la sección Normativa Legal.
		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: Germán Martínez Maldonado
  - 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: Germán Martínez Maldonado
  - 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: Germán Martínez Maldonado
  - 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: Germán Martínez Maldonado
  - 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 aprovisionamiento
  - Actores: Desarrollador
  - Tipo: Primario, Esencial
  - Referencias: (CU-1.) Iniciar plataforma
  - Precondición:
  - Postcondición: El portal queda instalado en la infraestructura seleccionada.
  - Autor: Germán Martínez Maldonado
  - 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 aprovisionamiento

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

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

4.3. Diagrama de paquetes

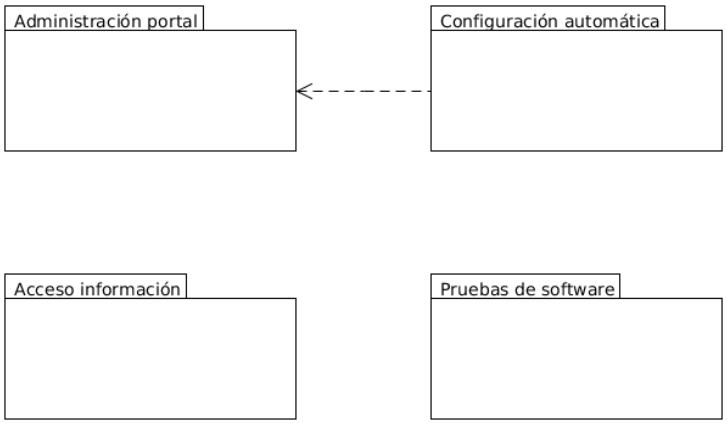

Figura 4.1: Diagrama de paquetes

4.4. Diagramas de casos de uso

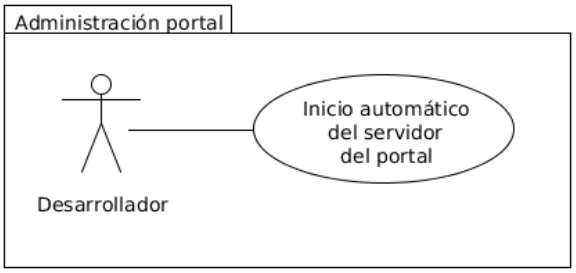

Figura 4.2: Diagrama de casos de uso: paquete Administración portal

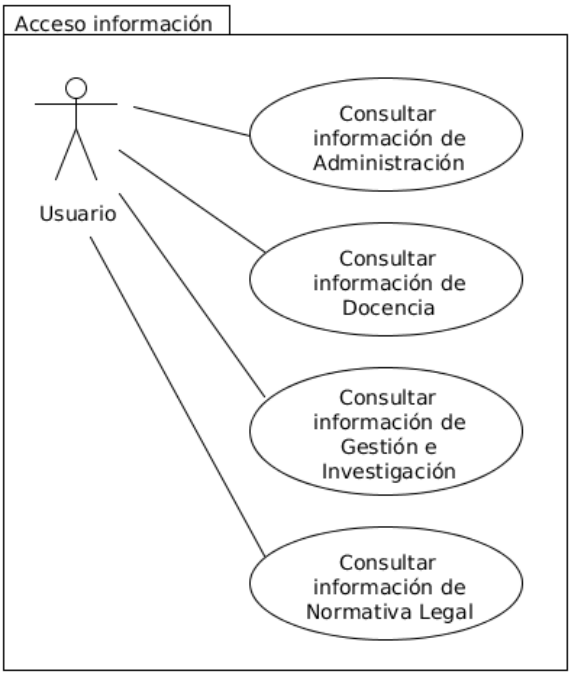

Figura 4.3: Diagrama de casos de uso: paquete Acceso información

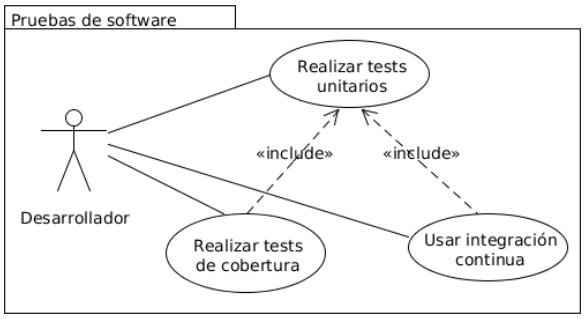

Figura 4.4: Diagrama de casos de uso: paquete Pruebas de software

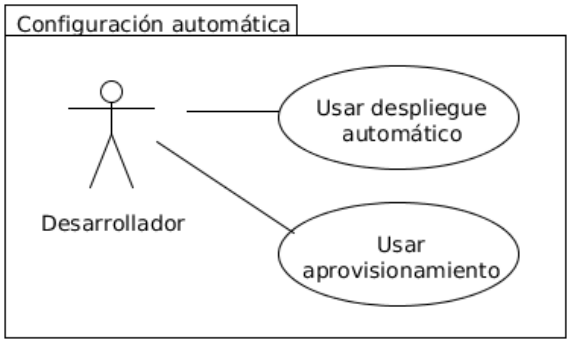

Figura 4.5: Diagrama de casos de uso: paquete Configuración automática

4.5. Diagramas de actividad

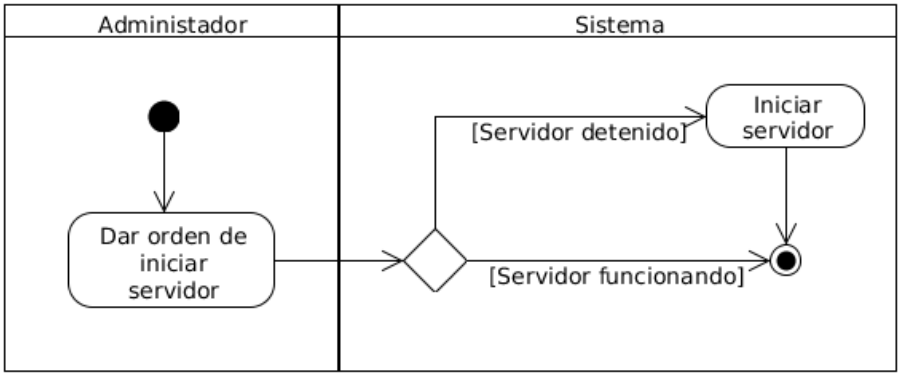

Figura 4.6: Diagrama de actividad CU-1. Inicio automático del servidor del portal

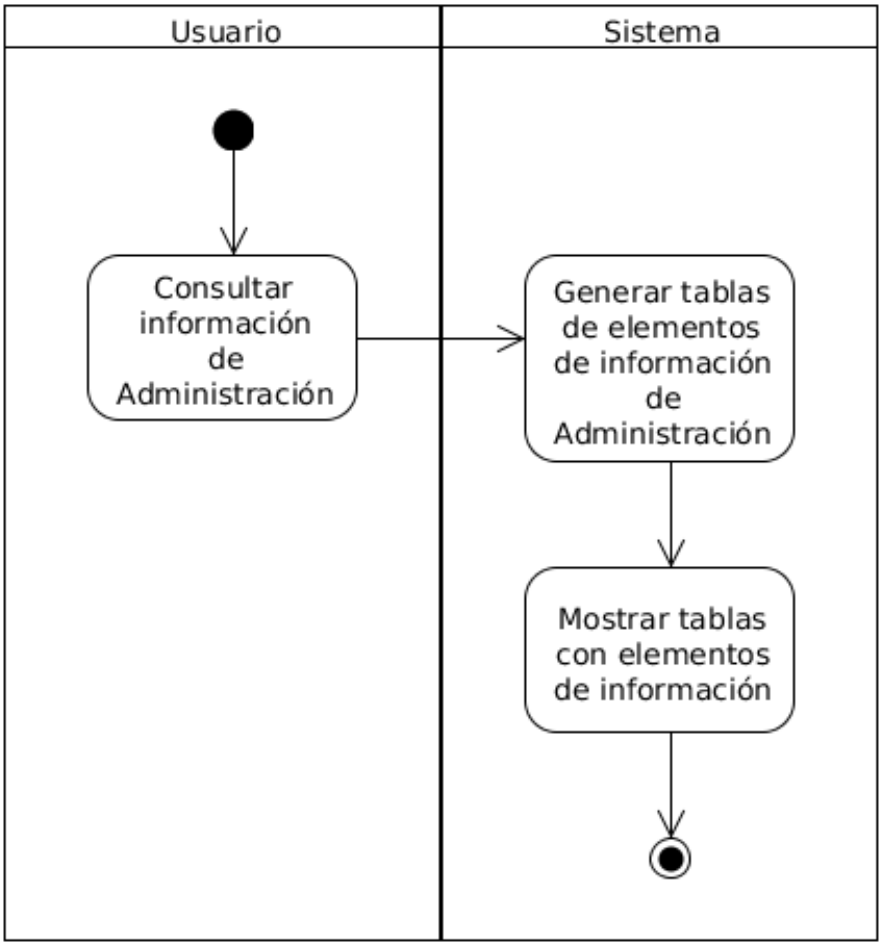

Figura 4.7: Diagrama de actividad CU-2. Consultar información de Administración

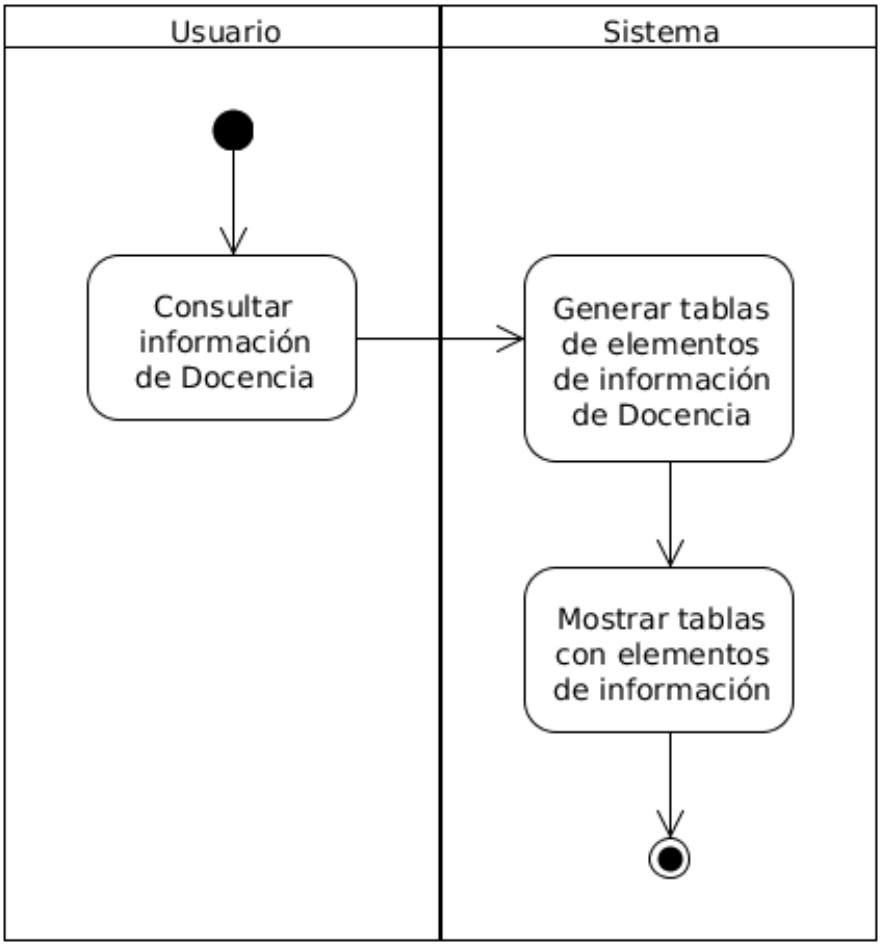

Figura 4.8: Diagrama de actividad CU-3. Consultar información de Docencia

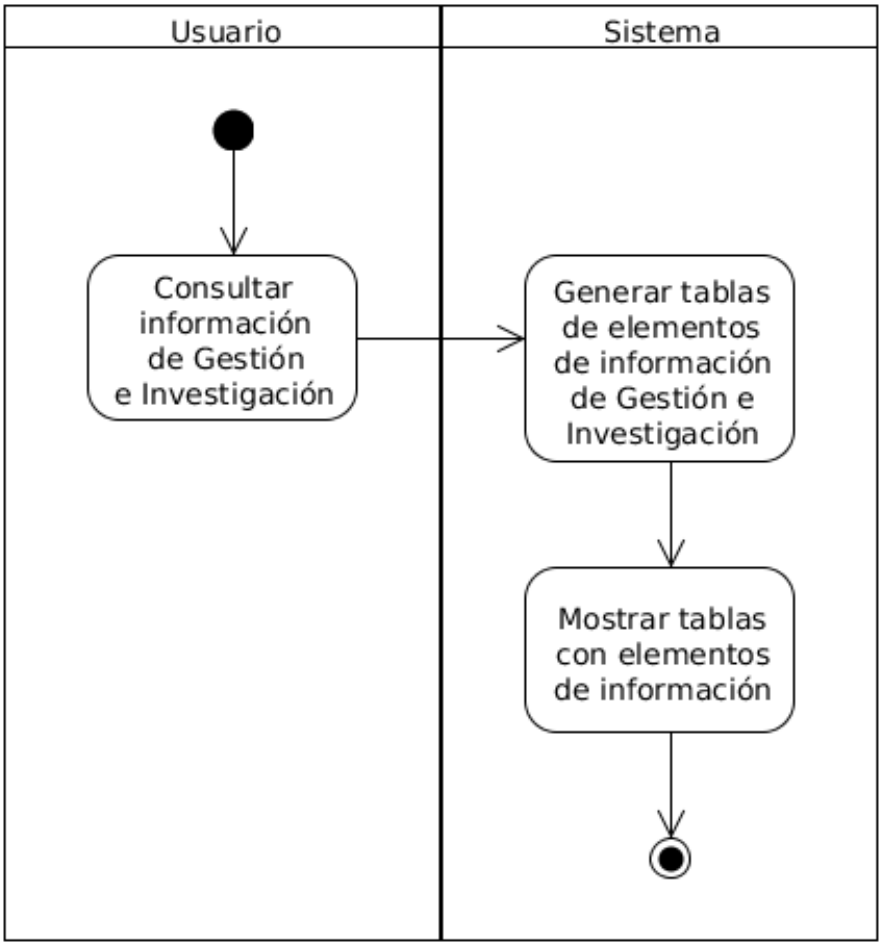

Figura 4.9: Diagrama de actividad CU-4. Consultar información de Gestión e Investigación

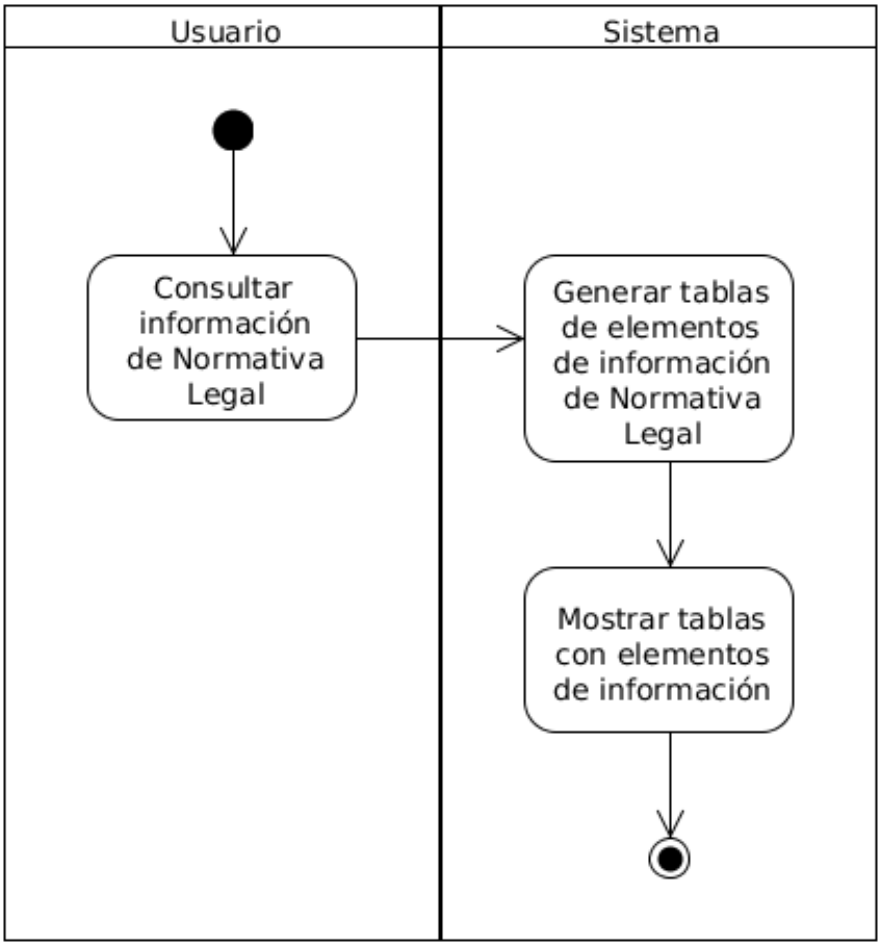

Figura 4.10: Diagrama de actividad CU-5. Consultar información de Normativa Legal

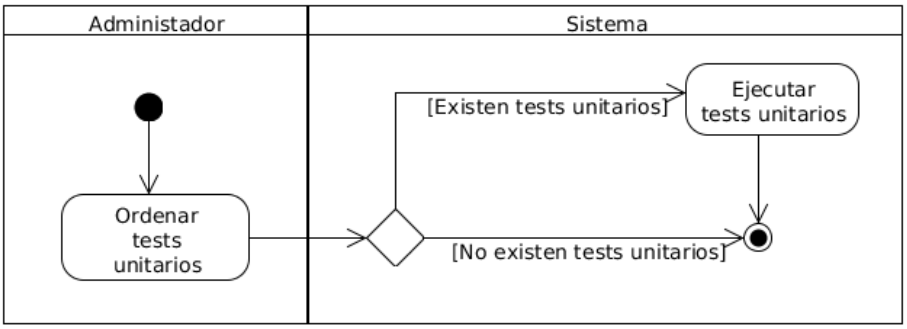

Figura 4.11: Diagrama de actividad CU-6. Realizar tests unitarios

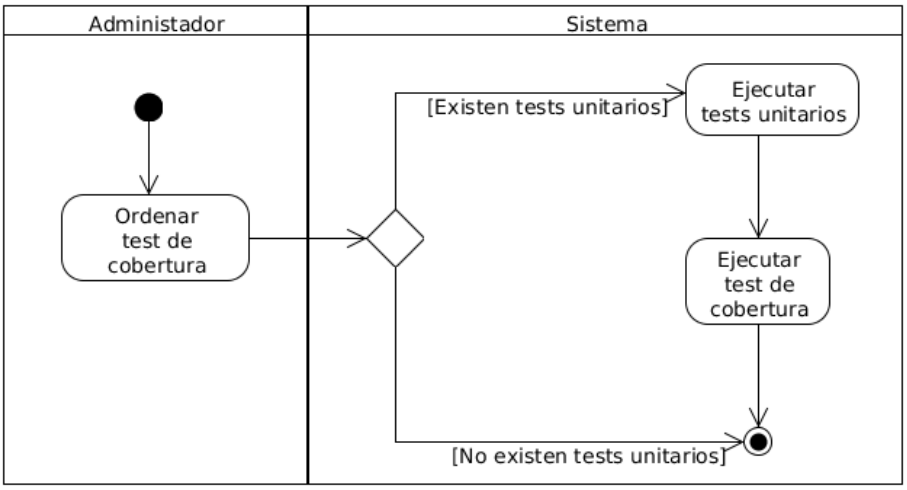

Figura 4.12: Diagrama de actividad CU-7. Realizar test de cobertura

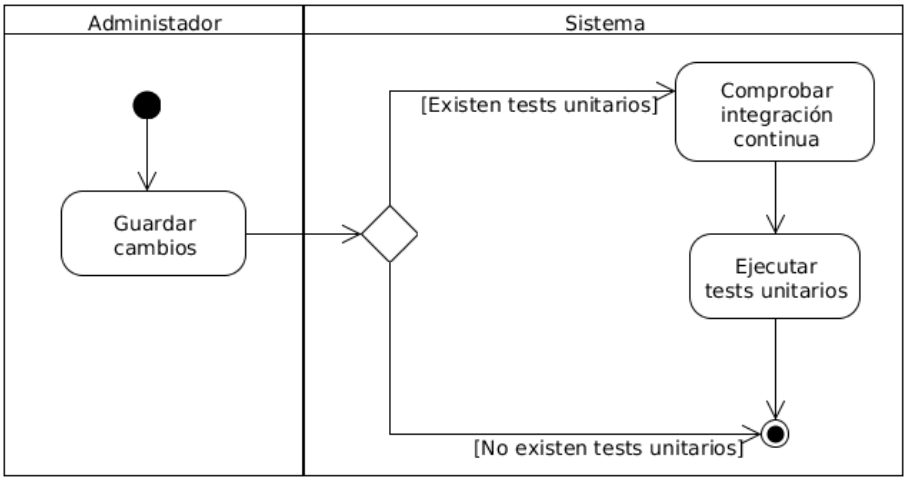

Figura 4.13: Diagrama de actividad CU-8. Usar integración continua

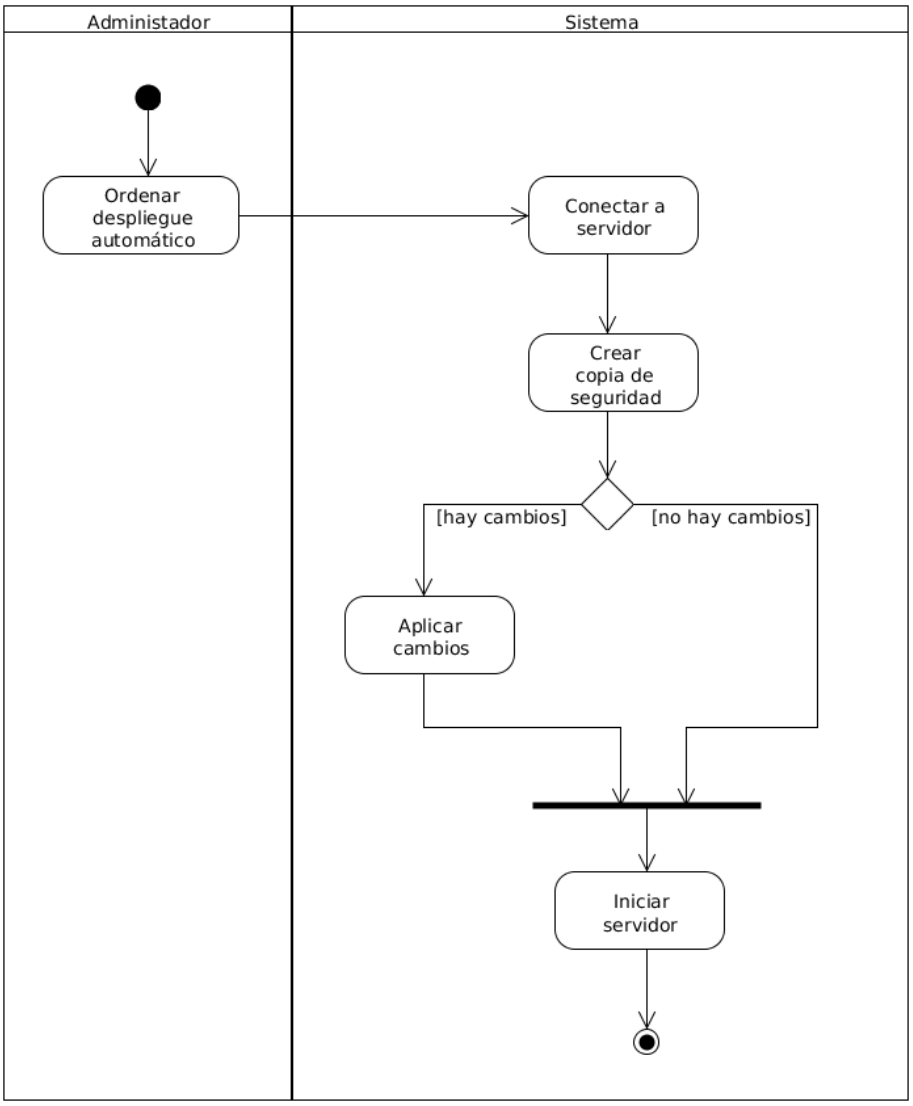

Figura 4.14: Diagrama de actividad CU-9. Usar despliegue automático

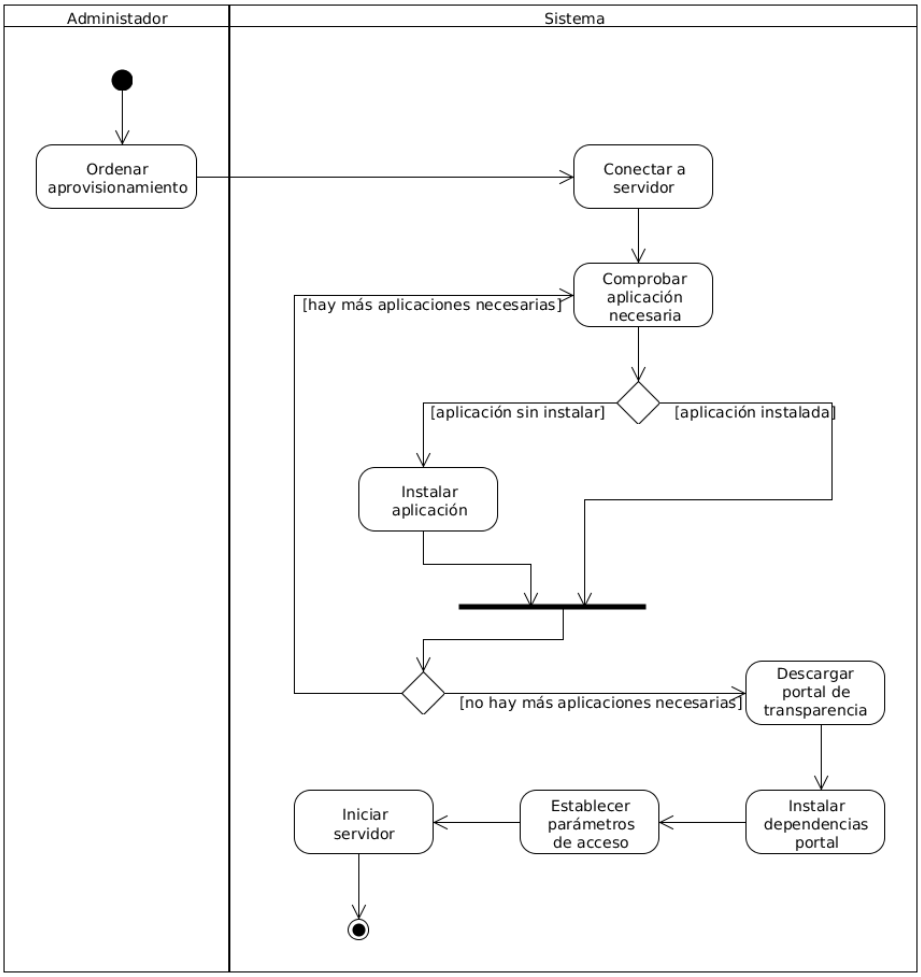

Figura 4.15: Diagrama de actividad CU-10. Usar aprovisionamiento

4.6. Diagrama conceptual

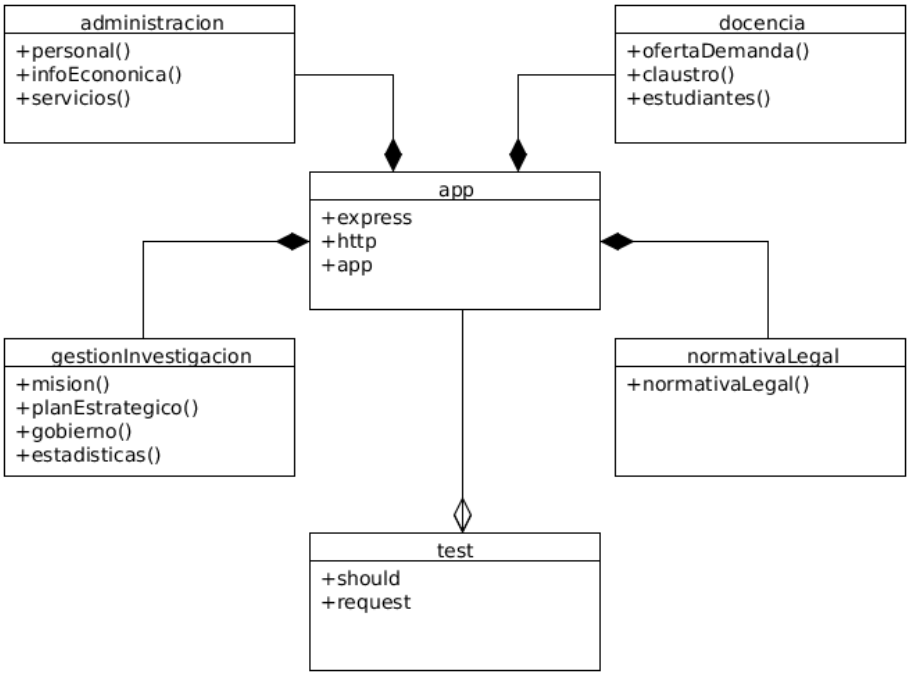

Figura 4.16: Diagrama conceptual

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 (`gestionInvestigación.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.

- Portal de Transparencia (`app.js`)
  - Administración (`administración.js`)
    - Personal (`personal.json`)
    - Información Económica (`infoEcononica.json`)
    - Servicios (`servicios.json`)
  - Docencia
    - Oferta y Demanda Académica (`ofertaDemanda.json`)
    - Claustro (`claustro.json`)
    - Estudiantes (`estudiantes.json`)
  - Gestión e Investigación
    - Misión (`mision.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 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 aprovisionamiento 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. Estás 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 aprovisionamiento 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 de automática que el código escrito cumple con el objetivo determinado; por lo que si una vez ejecutamos todos los tests unitarios, si 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 informa 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 cambios 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 aprovisionamiento

Con toda la aplicación ya realizada, para su instalación inicial en un plataforma determinada primero tenemos que aprovisionarla con todos los recursos software necesarios. Encontraremos una mayor cantidad de referencias a este procedimiento por su término en ingles, software provisioning, ya que el termino en español es una traducción literal que salvo por el acompañamiento del termino *"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

Implementacion

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 inicialmente 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.
- cotenido: 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":"Informacion Salarial 2015",
7        "link":"informacion-salarial-2015",
8        "texto":"Informacion relativa a la oferta..."
9      },
10
11   ],
12   "datos":[
```

```
13      {
14        "dataset":"Informacion Salarial 2015",
15        "id_dataset":"informacion-salarial-2015",
16        "nombre":"Analisis total plantilla: genero",
17        "vista":1,
18        "url":"51d53138-0408-4257-9909-57acea137a58",
19        "descarga":"985a8e1e-734b-432a-ac65-a7da..."
20      },
21
22    ]
23  }
```

Fragmento de código 6.1: Archivo `JSON` con informacion 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:

- 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).

```
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: Archivo `cargar.js`

Como ya tenemos toda la información como objetos JSON, ya podemos hacer uso de ella desde nuestra aplicación.

```
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  config = cargar(__dirname+'/config/config.json');
9  module.exports.config = config;
10
11 module.exports.personal = cargar(__dirname+'/config/personal.
       json');
12
13 app.get('/personal.html',administracion.personal);
14 app.get('/archivos/personal', function(req, res) {
15   res.send(cargar(__dirname+'/config/personal.json'));
16 });
17
18 http.createServer(app).listen(app.get('port'), app.get('ip'),
       function(){
19   console.log('Express server listening on ' + app.get('ip') + '
         :' + app.get('port'));
20 });
21
22 module.exports = app;
```

Fragmento de código 6.3: Archivo `app.js`

```
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`

```
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

```
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: Archivo `test.js`

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

Fragmento de código 6.7: Scripts de test

6.3. Integración continua

```
# language setting
language: node_js

# version numbers, testing against two versions of node
node_js:
- "0.12"
- "0.11"
- "0.10"
- "iojs"
```

Fragmento de código 6.8: Archivo `JSON` con informacion de personal

6.4. Despliegue automático

```
var plan = require('flightplan');

plan.target('transparente', {
  host: 'transparente.ugr.es',
  username: process.env.USER,
  agent: process.env.SSH_AUTH_SOCK
});

plan.remote(function(remote) {
  remote.log('Creando copia de seguridad...');
  remote.sudo('cp -Rf ugr-transparente-servidor ugr-transparente
      -servidor.bak', {user: process.env.USER});

  remote.with('cd ugr-transparente-servidor',function() {
    remote.log('Deteniendo el servidor...');
    remote.exec('sudo npm run-script kill');
    remote.log('Restableciendo parametros de acceso...');
    remote.exec('sed "s/IP=transparente.ugr.es/IP=127.0.0.1/" -i
        package.json');
    remote.exec('sed "s/PORT=80/PORT=3000/" -i package.json');
    remote.log('Obteniendo cambios...');
    remote.exec('git pull');
    remote.log('Instalando dependencias...');
    remote.exec('sudo npm install');
```

```
23      remote.log('Cambiando parametros de acceso...');
24      remote.exec('sed "s/IP=127.0.0.1/IP=transparente.ugr.es/" -i
            package.json');
25      remote.exec('sed "s/PORT=3000/PORT=80/" -i package.json');
26      remote.log('Arrancando el servidor...');
27      remote.exec('sudo npm start');
28    });
29  });
```

Fragmento de código 6.9: Archivo `test.js`

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

Fragmento de código 6.10: Scripts de despliegue automático

6.5. Aprovisionamiento

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

Fragmento de código 6.11: Archivo de hosts

```
1  ---
2  - hosts: transparente
3    sudo: yes
4    remote_user: "{{user}}"
5    tasks:
6      - name: A adiendo 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-
             servidor.git
20         dest=/home/"{{user}}"/ugr-transparente-servidor
21         version=master
22
```

```
23      - name: Cambiando propietario del directorio de la
            aplicacion...
24        file: path=/home/"{{user}}"/ugr-transparente-servidor
25              owner="{{user}}" group="{{user}}" state=directory
                    recurse=yes
26
27      - name: Instalando las dependencias de la aplicacion...
28        npm: path=/home/"{{user}}"/ugr-transparente-servidor
29
30      - name: Cambiando parametros de acceso (1/2)...
31        command: sed "s/IP=127.0.0.1/IP=transparente.ugr.es/" -i
32                 /home/"{{user}}"/ugr-transparente-servidor/
                       package.json
33
34      - name: Cambiando parametros de acceso (2/2)...
35        command: sed "s/PORT=3000/PORT=80/" -i
36                 /home/"{{user}}"/ugr-transparente-servidor/
                       package.json
37
38      - name: Arrancando el servidor...
39        command: chdir=ugr-transparente-servidor npm start
```

Fragmento de código 6.12: `Playbook` de `Ansible`