Facultad

de

Ciencias

APLICACIÓN WEB PARA LA GESTIÓN CURRICULAR DE LA EMPRESA

(Web application for curriculum management in the company)

Trabajo de Fin de Grado

para acceder al

GRADO EN INGENIERÍA INFORMÁTICA

Autor: Jorge Andrés de las Cuevas

Director: Rafael Menéndez de Llano Rozas

Co-Director: José Miguel Prellezo Gutiérrez

Junio - 2015

CONTENIDO

Agradecimientos VII

Resumen IX

Abstract XI

1 Introducción y objetivos 1

1.1 Introducción al tema 1

1.2 Objetivos 1

1.3 Estructura de la memoria 2

2 Material y métodos utilizados 3

2.1 Estado del arte 3

2.2 Tecnologías 4

2.2.1 Groovy 4

2.2.2 Groovy on Grails 4

2.2.3 Maven 5

2.2.4 HTML5 + CSS3 + JavaScript 5

2.2.5 Ajax 5

2.2.6 Bootstrap 6

2.2.7 JUnit 6

2.2.8 Spock 6

2.2.9 MySQL 7

2.2.10 Hibernate 7

2.2.11 LDAP 7

2.3 Herramientas 8

2.3.1 GGTS (Groovy/Grails Tool Suite) 8

2.3.2 Selenium 8

2.3.3 Tomcat 9

2.3.4 MySQL Workbench 9
2.3.5 Subversion 9
2.3.6 Putty 10
2.3.7 WinSCP 10
2.3.8 JXPlorer 10
2.3.9 Dia 11
2.3.10 Microsoft Office 11
2.3.11 Método *Kanban* 11
2.4 Framework de desarrollo: Groovy on Grails 12
2.4.1 Configuration Over Convention (COC) 12
2.4.2 Don´t Repeat Yourself (DRY). 13
2.4.3 ¿Por qué Grails? 13
2.4.4 Plug-ins utilizados 14
2.5 Metodología utilizada 15
2.6 Testing del Software 17
2.6.1 Test Unitarios 17
2.6.2 Test de Integración 18
2.6.3 Test Funcionales 19
2.6.4 Test de Aceptación 19
2.6.5 Test de Regresión 19
2.7 Test Driven Development (TDD) 20
2.7.1 Ciclo de desarrollo 20
2.7.2 Los test también son documentación 21
2.7.3 Ventajas 21
2.7.4 Desventajas 22
2.8 Principio de pareto 22
3 Desarrollo del proyecto 24

3.1 Análisis del sistema 24
3.1.1 Identificación de Usuarios 24
3.1.2 Especificación de casos de uso 25
3.1.3 Especificación de requisitos 26
3.1.4 Requisitos Funcionales 27
3.1.5 Requisitos No Funcionales 31
3.2 Diseño arquitectónico o lógico 32
3.2.1 Modelo Vista Controlador (MVC) 33
3.3 Implementación 35
3.3.1 Modelo 35
3.3.2 Test 36
3.3.3 Vista 39
3.3.4 Controlador 41
4 Evaluación en producción 43
4.1 Puesta en producción 43
4.2 Pruebas de aceptación 43
4.3 Pruebas de rendimiento 46
5 Conclusiones y trabajos futuros 47
5.1 Conclusiones 47
5.2 Trabajos futuros 48
5.2.1 Creación de modelos de entrevista 48
5.2.2 Integración con el Gestor de Proyectos. 49
5.2.3 Formación complementaria del candidato 49
5.2.4 Internacionalización 49
5.2.5 Mantenimiento de la aplicación 49
6 Referencias 50
Anexo I 52

Anexo II 53

ÍNDICE DE FIGURAS

Ilustración 1. Interfaz aplicación eProwin 3
Ilustración 2. Logotipo de Groovy 4
Ilustración 3. Logotipo de Grails 4
Ilustración 4. Logotipo de Maven 5
Ilustración 5. Logotipo de tecnologías web 5
Ilustración 6. Logotipo de Bootstrap 6
Ilustración 7. Logotipo de JUnit 6
Ilustración 8. MySQL 7
Ilustración 9. Logotipo de Hibernate 7
Ilustración 10. Interfaz Groovy/Grails Tool Suite 8
Ilustración 11. Logotipo de Selenium 8
Ilustración 12. Logotipo de Tomcat 9
Ilustración 13. Logotipo de MySQL Workbench 9
Ilustración 14. Logotipo de Subversion 9
Ilustración 15. Logotipo de Putty 10
Ilustración 16. Logotipo de WinSCP 10
Ilustración 17. Logotipo de JXPlorer 10
Ilustración 18. Logotipo de Dia 11
Ilustración 19. Logotipo de Microsoft Office 11
Ilustración 20. Demostración del Método Kanban 11
Ilustración 21. Comparativa Spring y Grails 14
Ilustración 22. Diagrama de flujo habitual de TDD 16
Ilustración 23. Pirámide de tipos de test 17
Ilustración 24. Esquema TDD 20
Ilustración 25. Fases de TDD 21

Ilustración 26. Diagrama de Casos de Uso del Sistema..... 26
Ilustración 27. Formato Requisitos del Sistema 27
Ilustración 28. Relación Casos de uso y Requisitos..... 27
Ilustración 29. Esquema arquitectura MVC 33
Ilustración 30. Diagrama de Despliegue del Sistema 34
Ilustración 31. Diagrama de Componentes del Sistema..... 34
Ilustración 32. Ejemplo de mapeo mediante GORM 35
Ilustración 33. Mapa Web de la Aplicación..... 40
Ilustración 34. Controladores del Sistema..... 41

AGRADECIMIENTOS

A mi familia, por todo el ánimo y preocupación que han mostrado por mí.

A todos los profesores de la carrera, porque cada uno puso su granito de arena para que hoy pueda estar aquí.

A mis compañeros de clase, por el mutuo apoyo y las inolvidables experiencias de estos últimos años.

A mis amigos, por su gran apoyo y confianza en mí, y a todas aquellas personas que han pasado por mi vida en estos años y han influido en mí. Ya que todas, en mayor o menor medida, tienen un hueco en mi memoria.

Un agradecimiento especial a la empresa CIC Consulting Informático, por darme la posibilidad de realizar el presente proyecto, poniendo a mi disposición los medios y experiencia que dispone.

RESUMEN

El presente documento del Proyecto de Fin de Grado (PFG) “Aplicación web para la gestión curricular de la empresa CIC Consulting informático de Cantabria” describe las distintas fases asociadas al análisis, diseño, implementación y pruebas de una herramienta de gestión de currículums y empleados para la empresa *CIC Consulting Informático de Cantabria*.

En una empresa en pleno crecimiento, en la que se ofrecen tanto nuevos puestos de empleo como cursos de formación, los currículums llegan diariamente. Con una media de 10 currículums recibidos a la semana, y 40 al mes, podemos darnos cuenta de la importancia que tiene guardar y organizar la información debidamente.

Esta herramienta, destinada especialmente al departamento de recursos humanos, permite informatizar los currículums y perfiles de todos los candidatos que llegan a la empresa, así como de los ya empleados en la misma. De esta forma se agiliza la búsqueda de personal para nuevos puestos de empleo, cursos de formación o proyectos.

La aplicación se ha implementado con el framework de desarrollo *Groovy on Grails*, mediante la técnica *Test Driven Development* (*TDD*), la cual asegura que se cumplen los requisitos acordados con el cliente.

Debido a la gran cantidad de dispositivos actuales, la aplicación se ha desarrollado en base a una estructura web, la cual permite generar vistas adaptables a cualquier dispositivo (ordenador, tablet, smartphone, etc). Para ello, se ha implementado una arquitectura Modelo Vista Controlador (MVC), que separa el software en tres capas. Esto otorga una arquitectura definida, homogénea y ampliamente aceptada en base a la experiencia, mejorando la velocidad de desarrollo y facilitando la tarea de mantenimiento creando un software más modular.

Palabras clave: Aplicación web, Curriculums, Test Driven Development, Modelo Vista Controlador, Grails

ABSTRACT

This document about Degree´s Final Project (DFP) "Web application for curriculum management in the company CIC Consulting Informatico de Cantabria" desribes the different phases associated to the analysis, design, implementation and testing of a curriculum and employee management tool for the company CIC Consulting Informático de Cantabria..

The average of curriculums that receives this company is about 10 each week. This means, that they have 40 new resumes each month. So we come to realize the importance of saving and organizing this information properly.

This tool, destinated especially to the human resources department, allows to save the curriculum and profiles of all candidates who come to the company, as well as the information about the employees. This speeds up the search of staff for new jobs, formation courses or projects.

This application has been implemented with the framework Groovy on Grails, through the Test Driven Development (TDD), which guarantees that all accorded requirements with the client are met.

Due to the large amount of current devices, this application has been developed based on a web architecture, which allows to generate adaptable views for any device (computer, tablet, smartphone, etc.). For this, a Model View Controller (MVC) architecture has been implemented, that divides the software in 3 layers, in order to have a defined, homogeneous and widely accepted architecture, improving the development speed and easing the maintenance by making a more modular software.

Keywords: Web application, curriculums, Test Driven Development, Model View Controler, Grails

1 INTRODUCCIÓN Y OBJETIVOS

En este primer apartado se introducirá al lector en la materia del proyecto, metodología de desarrollo, objetivos que se pretenden conseguir y medios técnicos a utilizar. Así mismo, se expondrá la estructura que sigue este documento.

1.1 INTRODUCCIÓN AL TEMA

Una de las tareas primordiales para una empresa consiste en seleccionar personal para su plantilla, que facilite el desarrollo de sus actividades de la forma más competente posible. Normalmente se busca personal para un puesto de trabajo específico y se requieren unas capacidades o habilidades concretas. Es trabajo del departamento de recursos humanos, en adelante *RRHH*, estudiar cada currículum que les entregan y seleccionar los mejores candidatos para cada puesto en función de sus habilidades y experiencias.

A menudo estos currículums se amontonan y se deben leer hojas y hojas de candidatos que no interesan o se dejan de leer currículums de otros que podrían estar muy cualificados para la empresa.

En otras ocasiones, se puede llegar a entrevistar a un candidato en más de una ocasión, sin disponer de los datos de otras entrevistas al ser realizadas por evaluadores diferentes.

Una búsqueda de un candidato entre 50 posibles, implica leer un mínimo de 100 hojas de currículums, hacer entrevistas a los que puedan interesar y después seleccionar los más aptos. Mientras que un sistema de información permitiría hacer una búsqueda con los parámetros requeridos, obteniendo un resultado de los candidatos aptos ordenados por sus habilidades, agilizando el proceso de selección, ahorrando tiempo, recursos y, por lo tanto, dinero.

Otro asunto en los procesos de selección es que se requiere un trabajo colaborativo inter-departamental, ya que pueden entrar en juego competencias de distintos perfiles.

Con esta aplicación se pretende que el trabajo del departamento de *RRHH* sea mucho más eficaz tanto en el tiempo empleado en la búsqueda de personal, como en los resultados obtenidos en la selección de candidatos.

1.2 OBJETIVOS

El objetivo principal que se pretende conseguir con este Trabajo de Fin de Grado es el desarrollo del sistema de información “Aplicación web para la gestión curricular de la empresa CIC Consulting informático”, aportando valor añadido al departamento de *RRHH* de la empresa.

Para la realización de este proyecto, se definen de manera más específica los siguientes sub-objetivos:

1. Almacenamiento de toda la información relevante de los candidatos y sus currículums.
2. Guardar los datos de los candidatos presentados a cada proceso de selección y la información generada en sus entrevistas.
3. Implementación de un potente buscador de candidatos por habilidades, titulaciones y experiencia.
4. Construcción de una interfaz de la aplicación accesible y funcional desde distintos dispositivos.
5. Conectar con el directorio activo de la empresa para autenticar a los usuarios del sistema de información.
6. Funcionalidad para mostrar estadísticas con la información recibida en la aplicación.

Además, se cumplen otros objetivos académicos, como poner en práctica el método de desarrollo *Test-Driven Development* (*TDD*), el cual basa la construcción del software en las pruebas y la refactorización continua.

También se ha estudiado el estado actual de los diferentes frameworks de desarrollo web y sus arquitecturas software.

1.3 ESTRUCTURA DE LA MEMORIA

La distribución de los diferentes apartados de esta memoria del trabajo de fin de grado es la siguiente:

- Primero, en la sección 2, se presenta el estado del arte en el momento de realización del proyecto, así como los medios físicos, herramientas y tecnologías utilizadas. Se expone también la metodología utilizada.
- El apartado 3 es el importante de esta memoria. En él se explica cómo ha sido el progreso de producción del software, explicando cada fase de desarrollo. Empieza identificando los distintos usuarios, siguiendo con un análisis de las funcionalidades y obteniendo los requisitos del sistema. Finalmente, se presenta el modelo arquitectónico escogido y su implementación.
- En la sección 4, se evalúa la puesta en producción de la aplicación.
- En el apartado 5, se analizan los resultados tanto personales como profesionales del proyecto. Además, se exponen algunas mejoras que podrían ser implementadas en el futuro.
- Por último, en la sección 6 podemos encontrar toda la bibliografía.

2 MATERIAL Y MÉTODOS UTILIZADOS

En este capítulo se trata en primer lugar el estado del arte, donde se describen otras herramientas consultadas como base del trabajo. A continuación se detallan las tecnologías y herramientas software utilizadas. Por último, se detalla la metodología escogida para llevar a cabo el proyecto.

2.1 ESTADO DEL ARTE

En una empresa en pleno crecimiento como es *CIC*, que dispone de más de 200 empleados y recibe de media 10 currículums semanales, la gestión de la información interna empieza a ser una de las actividades clave para el departamento de *RRHH*.

Hasta la implementación del presente proyecto, los empleados encargados de gestionar esta información almacenaban los currículums en papel, anotando datos de entrevistas en documentos Excel y otros métodos rudimentarios.

Con todo esto, la empresa decide implantar una nueva aplicación que organice mejor la información, de forma que sea más accesible y apropiada.

Al buscar aplicaciones de estas características nos encontramos con algunos *CRM* (*Customer Relationship Management*) que implementan módulos que resuelven algunas necesidades de la empresa.

La herramienta que más se ajusta a las funcionalidades requeridas es *eProwin [1].*

Ilustración 1. Interfaz aplicación eProwin

eProwin es un software de gestión de currículums online, que permite gestionar las ofertas de trabajo y las candidaturas.

Esta herramienta da la posibilidad a las empresas de publicar ofertas de empleo y gestionar los currículums que reciben para realizar una gestión completa y centralizada. Además permite a los candidatos crear sus propios currículums online para enviárselo directamente a las empresas.

Con todo esto, podemos deducir que se trata de un portal de empleo, con ciertas funcionalidades adicionales de gestión por parte de la empresa.

CIC buscaba una herramienta más optimizada a sus necesidades y productos, almacenando otro tipo de información, como la de los empleados que ya se encuentran en la empresa o un sistema de alertas que avise cuando, por ejemplo, un candidato termine sus estudios.

Con esta búsqueda inicial, se decidió que lo mejor era desarrollar una nueva aplicación completamente personalizada a los flujos de trabajo de la empresa y que además, pudiera estar integrada con otras aplicaciones de gestión interna.

2.2 TECNOLOGÍAS

Para el desarrollo de este trabajo de fin de grado han sido necesarias varias tecnologías, que se exponen a continuación.

2.2.1 Groovy

Groovy [2] es un lenguaje de programación dinámico, orientado a objetos e implementado sobre la plataforma *Java*. Tiene una curva de aprendizaje corta, sintaxis compacta, soporte para tipado dinámico y pruebas unitarias. Al estar desarrollado sobre *Java*, comparte muchas características con éste y puede acceder a sus librerías.

Ilustración 2. Logotipo de Groovy

Grails se basa en este lenguaje para el desarrollo de software, por lo que se ha utilizado la versión *v2.3.7* de Groovy para su programación.

2.2.2 Groovy on Grails

El framework de desarrollo utilizado ha sido *Groovy on Grails [3]* en su versión *v2.5.0*. *Grails* aglutina todas las características que un sistema de desarrollo de aplicaciones web requiere, siguiendo el patrón *MVC*. En el punto *2.4* se explicarán en detalle sus características y porqué ha sido escogido en detrimento de otros frameworks.

Ilustración 3. Logotipo de Grails

2.2.3 Maven

Maven [4] utiliza un fichero *POM* (*Project Object Model*) para describir las dependencias del software en construcción con otros módulos y componentes externos, de forma que su compilación, empaquetado y despliegue sea mucho más rápida y fácil.

Ilustración 4. Logotipo de Maven

La gestión de dependencias en *Groovy* es automática, siendo *Maven* quien trabaja por debajo para descargar todos los *.jar* necesarios.

2.2.4 HTML5 + CSS3 + JavaScript

La mayoría de aplicaciones web modernas están basadas en estas tres tecnologías *[5]*:

Ilustración 5. Logotipo de tecnologías web

HyperText Markup Language, o *HTML*, es un lenguaje de etiquetas con el que se maquetan las páginas web, definiendo la estructura y el contenido a visualizar. En este proyecto se ha utilizado la versión 5, soportada ya por la mayoría de los navegadores.

Cascading Style Sheet, o *CSS*, es el lenguaje encargado de dar estilo visual a los contenidos de la página web. Se ha utilizado la versión 3, que otorga grandes funcionalidades para potenciar el aspecto visual.

La tercera, *JavaScript*, es un lenguaje de programación interpretado utilizado para dotar a las webs de dinamismo, esto es, que los objetos o contenidos de la web puedan cambiar dinámicamente sin necesidad de recargar la página por completo. El propio navegador web es capaz de interpretarlo, ya que no es necesaria compilación alguna.

El conjunto de estas 3 tecnologías, se usa para construir la parte de la Vista del patrón MVC que explicaremos en la sección *3.2.1*.

2.2.5 Ajax

Asynchronous JavaScript And XML [5], o *Ajax*, no es realmente una tecnología, sino como dice su nombre, es la mezcla de *JavaScript* y *XML* para conseguir intercambiar datos entre el cliente y el servidor de forma asíncrona.

Su uso proporciona una experiencia de usuario más cercana a las aplicaciones de escritorio, evitando los recargos de página tradicionales en las aplicaciones web.

Se ha utilizado para actualizar datos de una pantalla, o para cambiar la vista de un conjunto de datos y poder modificarlos.

2.2.6 Bootstrap

Bootstrap [6] es un framework de *Twitter* que utiliza las tecnologías descritas en el punto *2.3.4* para desarrollar aplicaciones web de forma rápida y fácil.

Dispone de una serie de temas prediseñados para este tipo de webs y con un diseño responsive. Esto significa que una misma página cambia de tamaño y estructura dependiendo del ancho del dispositivo con el que lo visualicemos.

Ilustración 6. Logotipo de Bootstrap

Para agilizar el proceso de creación de la interfaz gráfica y que se adapte a todo tipo de dispositivos se ha utilizado esta herramienta, simplificando la tarea de maquetación en la fase de desarrollo.

2.2.7 JUnit

JUnit [7] es un framework que nos permite desarrollar test para aplicaciones Java, ejecutándolos de forma controlada, para comprobar que su funcionamiento es correcto.

El entorno de desarrollo utilizado, que se explicará en la sección de Herramientas *2.3.1*, cuenta con un plugin que facilita la creación de estos test, presentando también los resultados de los mismos de forma gráfica.

Ilustración 7. Logotipo de JUnit

Se ha utilizado en su versión *4* para realizar los test funcionales, ya que es uno de los lenguajes para test más utilizados en la plataforma Java.

2.2.8 Spock

Spock [8] es una herramienta que funciona sobre *JUnit* para crear test unitarios y de integración en aplicaciones desarrolladas con *Groovy*.

Dispone de casi todas las funcionalidades de JUnit, y otras añadidas, como crear *Mocks*, que sirven para simular el comportamiento de los módulos del sistema.

Su verdadero poder está en la simplicidad y claridad del código de las pruebas, centrando la atención en los datos a probar más que en el propio código.

Se ha utilizado la versión *v0.7* para la realización de los test unitarios y de integración.

2.2.9 MySQL

MySQL [9] es un sistema de gestión de base de datos relacional, multihilo y multiusuario. Para el acceso a las bases de datos se ha utilizado el lenguaje *SQL*, a través de un framework de acceso a datos que proporciona *Grails*.

Ilustración 8. MySQL

Para el almacenamiento de toda la información de la aplicación se ha utilizado una base de datos *MySQL v5.5*.

2.2.10 Hibernate

Hibernate [10] es una herramienta de *Mapeo Objeto-Relacional* (*ORM*) para la plataforma *Java*, que facilita el mapeo de objetos de aplicación a una base de datos relacional.

Ilustración 9. Logotipo de Hibernate

Grails utiliza esta tecnología para liberar al programador de la tarea de interacción con las *clases DAO* que proveen información a la capa de negocio, integrándose directamente con las entidades de dominio utilizadas. En este proyecto se ha utilizado la versión *v4* de *Hibernate*.

2.2.11 LDAP

Lightweight Directory Access Protocol, o LDAP, es un protocolo que permite acceder a un servicio de directorio activo.

Estos directorios se pueden encontrar en uno o varios servidores, guardando información ordenada en forma de árbol acerca de usuarios, equipos o grupos. *CIC* utiliza un repositorio basado en el *Microsoft Active Directory.*

Se ha utilizado para acceder a los datos del personal de la empresa y poder autenticar así a los usuarios de la aplicación con su usuario y contraseña que utilizan para otras aplicaciones. Además, se han creado 3 grupos de usuarios en el directorio, de manera que podamos diferenciar si el usuario es un empleado, personal de *RRHH* o administrador.

2.3 HERRAMIENTAS

A continuación se describen las principales herramientas software utilizadas.

2.3.1 GGTS (Groovy/Grails Tool Suite)

GGTS [11] es un entorno de desarrollo open source, actualmente propiedad de *Pivotal Software*. Está basado en *Eclipse*, proporcionando una serie de funcionalidades (perspectiva propia, plugins, soporte para *Groovy*, etc.) para trabajar con el framework de *Groovy on Grails*.

Para el desarrollo del proyecto se ha utilizado la versión *v3.6.4* de la herramienta.

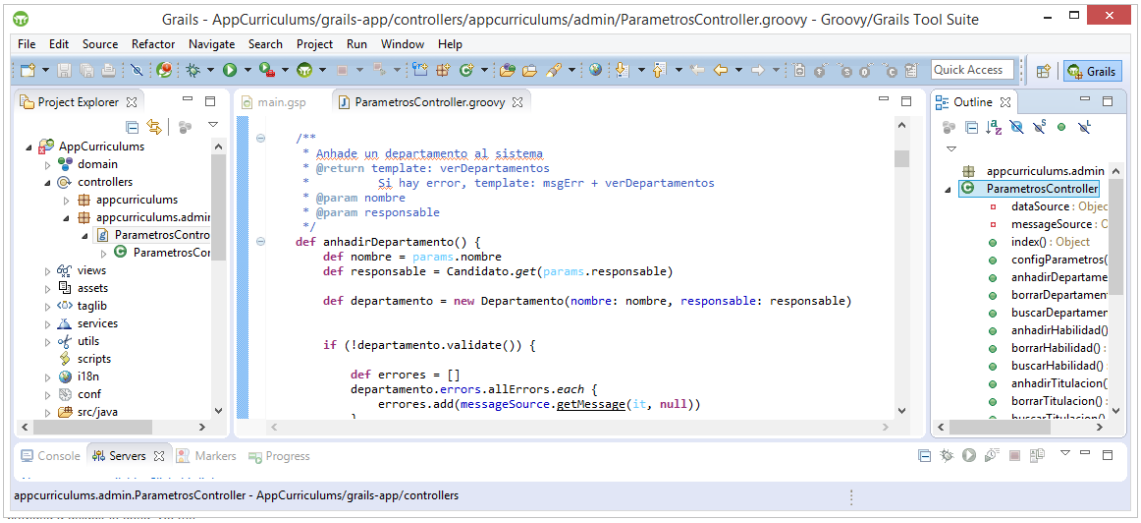


Ilustración 10. Interfaz Groovy/Grails Tool Suite

2.3.2 Selenium

Selenium *[12]* es una herramienta que permite, mediante un driver para el navegador, grabar y reproducir "acciones" en una página web, de forma que puedan servir como test funcionales.

Ilustración 11. Logotipo de Selenium

Entre todos los lenguajes que soporta *Selenium*, se ha elegido escribir los test en *JUnit*, ya que es soportado por el framework de *Grails* sin necesidad de añadir nuevos plugins y es uno de los más utilizados por los desarrolladores. Para la grabación de comandos en la interfaz web se ha utilizado la extensión *Selenium IDE* para el explorador *Firefox*.

2.3.3 Tomcat

Apache Tomcat [13] es un contenedor de *servlets* y *JSPs*. Está basado en el lenguaje *Java*, por lo que puede funcionar en cualquier *Máquina Virtual de Java* (*JVM*).

Para desplegar una aplicación sólo necesitamos generar el archivo *.war*, incluirlo en la carpeta correspondiente del *Tomcat* y poner en marcha el servidor de aplicaciones.

Ilustración 12. Logotipo de Tomcat

Se ha utilizado la versión *v8,* tanto en el entorno de desarrollo como en el de producción, para la puesta en marcha de la aplicación web.

2.3.4 MySQL Workbench

Para la gestión de la base de datos en el entorno de desarrollo se ha utilizado la herramienta *MySQL workbench [14]*, la cual proporciona facilidades para la creación, administración y mantenimiento de bases de datos *MySQL*. Se ha utilizado la versión *v6.3* de la herramienta.

Ilustración 13. Logotipo de MySQL Workbench

2.3.5 Subversion

Subversion [15], o *SVN*, es un Sistema de Gestión de la Configuración (SCM) open source. Con ella podemos tener, en un repositorio, las distintas versiones de nuestro proyecto. Además, nos aporta otro tipo de funcionalidades como ver diferencias en el código entre una versión u otra o trabajar varias personas en un mismo proyecto.

Ilustración 14. Logotipo de Subversion

Para el acceso al repositorio se ha utilizado el cliente de subversión *TortoiseSVN [16]* en su versión *v1.7.*

2.3.6 Putty

Putty [19] es un cliente SSH para el acceso a servidores remotos mediante comandos.

Ilustración 15. Logotipo de Putty

Se ha utilizado la versión *v0.64* para gestionar de forma remota todos los componentes desplegados en el servidor, principalmente el gestor de bases de datos *MySQL* y el contenedor de aplicaciones web *Apache Tomcat*.

2.3.7 WinSCP

WinSCP [20] es una aplicación de software libre, que hace de cliente *SCP* y *SFTP*, para acceder al directorio de archivos del servidor y poder intercambiar ficheros de forma segura.

Ilustración 16. Logotipo de WinSCP

Se ha utilizado la versión *v5.7.3* para subir todos los archivos necesarios al servidor de producción.

2.3.8 JXPlorer

JXPlorer [21] es una herramienta para la gestión de directorios mediante *LDAP, que* permite ver y modificar los datos del mismo.

Ilustración 17. Logotipo de JXPlorer

Para la gestión del directorio activo de usuarios de la aplicación se ha utilizado la versión *v3.3.1* de esta herramienta.

2.3.9 Dia

Dia [17] es una aplicación que permite crear diagramas de muchos ámbitos de forma fácil y rápida. Se ha utilizado la versión *v0.97.2* para la creación de diagramas entidad-relación, casos de uso, *UML*, etc.

Ilustración 18. Logotipo de Dia

2.3.10 Microsoft Office

La *Suite Office de Microsoft [18]* permite crear todo tipo de documentos. Se ha utilizado para escribir la presente memoria, así como documentación, estadísticas y ejemplos de uso. Se ha utilizado la versión *2013* del *pack office*.

Ilustración 19. Logotipo de Microsoft Office

2.3.11 Método *Kanban*

El *método kanban* define un proceso de organización de la producción que permite tener claro en todo momento el estado de cada tarea. Su procedimiento se basa en colocar *tarjetas*, “ban” en japonés, de forma *visual*, “tarjeta” en japonés, en un tablero con regiones para las tareas por hacer, en proceso y hechas.

Existen también otras variantes con tipos de urgencia o más estados, pero en este caso se ha optado por la versión más simple. El resultado del tablero es el siguiente:

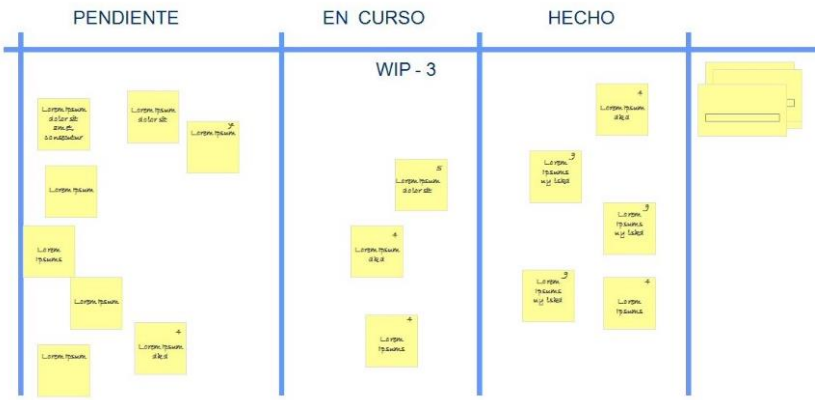


Ilustración 20. Demostración del Método Kanban

El proceso de utilización es:

- Definir el flujo de trabajo del proyecto, con las tareas que se han de realizar.
- Escribir cada tarea que va surgiendo en un post-it, con una pequeña descripción, estimación de horas y prioridad.
- Definir las un número máximo de tareas en curso. De esta forma siempre procuraremos acabar una tarea antes de empezar otra.
- Empezar las tareas con mayor urgencia, y tener siempre a mano el tablero para poder hacer un seguimiento del estado del proyecto.

2.4 FRAMEWORK DE DESARROLLO: GROOVY ON GRAILS

Como ya hemos comentado, el framework de desarrollo utilizado ha sido *Groovy on Grails*. Este framework para la construcción de aplicaciones web, crea un marco de trabajo altamente productivo para cada una de las capas de la aplicación. Está desarrollado sobre Java, por lo que puede desplegarse en cualquier *JVM* (*Java Virtual Machine*). Se presenta como un framework fácil de aprender, de alto rendimiento y con potentes funcionalidades, ya que se integra con muchas tecnologías.

Groovy on Grails surge gracias a la iniciativa del proyecto *Ruby on Rails (RoR)*, por lo que comparte características con éste, como la convención sobre la configuración que explicaremos en el punto 2.4.1.

Al estar construido sobre el entorno de *Java*, se pueden utilizar la mayoría de sus librerías. Funciona también sobre otras tecnologías ya probadas y muy utilizadas, como son *Hibernate* y *Spring*, haciendo su configuración muy sencilla para el programador.

La curva de aprendizaje y el tiempo de puesta en marcha de un proyecto son muy pequeñas, por lo que es especialmente recomendable para grupos de desarrollo que estén empezando con frameworks de aplicaciones web.

Dispone también de un generador de documentación similar al *javadoc*. Con esto, podemos generar una web *html* o un *pdf* con toda la documentación del proyecto en base a los comentarios del código.

Grails se apoya principalmente en dos paradigmas de la programación, que se explican a continuación: la *convención sobre la configuración* y el *DRY*.

2.4.1 Configuration Over Convention (COC).

La *Convención Sobre Configuración* busca agilizar el proceso de configuración de una aplicación, estandarizando el mismo. De esta forma el programador sólo debe definir los aspectos no convencionales del software.

Grails se encarga de generar automáticamente los archivos de configuración de la aplicación, tomando algunas reglas como estándar. Por ejemplo, las clases cuyo nombre acabe en *Controller*, serán tomadas como controladores, etc.

También podemos cambiar el funcionamiento por defecto manualmente, por lo que la flexibilidad que tenemos es total a la vez que agilizamos y facilitamos el proceso.

2.4.2 Don´t Repeat Yourself (DRY).

Don´t Repeat Yourself, o *no te repitas*, es una práctica de programación que se basa en no repetir el código en varios módulos de la aplicación.

Por ejemplo, si tenemos dos métodos o más que necesitan procesar las fechas sacadas de la base de datos para mostrarlas al usuario, podemos crear una función que realice esa tarea. Si más adelante la forma de mostrar las fechas cambia, solo deberemos modificar esta función. De esta forma se simplifica la realización de cambios, evitando posibles inconsistencias.

Esto se aplica en todas las fases y capas del desarrollo: código de la aplicación, base de datos y documentación generada.

2.4.3 ¿Por qué Grails?

Antes de empezar con el diseño del proyecto, se plantearon las diferentes tecnologías que se podían utilizar para el desarrollo.

Una de las peticiones que hizo la empresa, fue que la aplicación debía ser accesible desde distintos dispositivos. Crear un software para ordenadores, otro para tablets y otro para Smartphones sería mucho trabajo para una aplicación que no necesita optimizar tanto los recursos, por lo que se optó por crear una aplicación web adaptable a todo tipo de pantallas.

Haciendo una valoración de la experiencia de desarrollo, tanto mía como de la empresa, se llegó a la conclusión de que el mejor marco de trabajo en ese momento era escoger algún framework de *JAVA*.

Sabiendo esto, se realizó una búsqueda de los distintos Frameworks existentes para la creación de aplicaciones web. Entre ellos:

- Grails
- Spring
- Vaadin
- GWT
- Struts
- JSF

Una vez buscados los posibles frameworks a utilizar, se compararon sus principales características, en base a especificaciones y comentarios de otros desarrolladores en la web.

La conclusión a la que se llegó, fue que *Spring* y *Grails* son actualmente muy populares para el desarrollo de este tipo de aplicaciones web, gracias a su gran potencia, flexibilidad y productividad.

Minimizado el rango de frameworks a estos dos, se pudo hacer una exhaustiva búsqueda de sus características para poder decidir entre uno de ellos.

La siguiente tabla muestra las diferencias que existen entre los frameworks de *Spring* y *Grails*.

Spring	Grails
✗ Debemos conocer y saber utilizar todas las tecnologías que utiliza	✓ Nos abstrae de las tecnologías que hay por debajo. Mejor para novatos
✓ Más número de usuarios y documentación	✗ Menos usuarios y documentación
✗ Gran curva de aprendizaje	✓ Curva de aprendizaje pequeña
✗ Productividad moderada	✓ Alta productividad, menos tiempo de desarrollo
✓ Consume menos recursos	✗ Consume más recursos, sobretodo en el entorno de desarrollo
✓ Fácil de depurar	✗ Difícil de depurar si hay errores

Ilustración 21. Comparativa Spring y Grails

Tras leer varias comparativas y experimentar con ambos frameworks, se llegó a la conclusión de que *Grails* es más adecuado para este proyecto.

Se decidió así ya que partimos de 0 en lo que a frameworks de desarrollo se refiere, y *Grails* tiene una curva de aprendizaje mucho menor. Además, se exige realizar el proyecto en un tiempo pequeño, y éste agiliza el proceso de desarrollo automatizando muchas de las tareas del programador.

2.4.4 Plug-ins utilizados

Una de las razones por la cual Grails hace tan fácil el desarrollo de aplicaciones web es la cantidad de plugins que podemos añadir para facilitar la implementación. Para este proyecto hemos utilizado los siguientes.

2.4.4.1 Database Migration

Grails database migration plugin (v1.4.0) [22] se ha utilizado para gestionar cambios específicos en la base de datos mediante la aplicación. Por ejemplo, crear la vista que utiliza el buscador para encontrar candidatos.

2.4.4.2 Geb integration

Geb integration for Grails (v0.10.0) [23] permite ejecutar las pruebas funcionales con la tecnología *WebDriver* de *Selenium* en nuestro navegador web.

2.4.4.3 Quartz plugin

Quartz plugin for Grails (*v1.0.2*) *[24]* permite definir Jobs que se ejecuten periódicamente. Se ha utilizado para ejecutar un proceso a diario que compruebe las alertas que hay y enviarlas al correo del departamento de *RRHH*.

2.4.4.4 Mail support

Mail support to a running Grails application (*v1.0.7*) *[25]* proporciona un servicio para enviar emails desde la aplicación. Se ha utilizado para, con los Jobs del punto anterior, para enviar correos al departamento de *RRHH*.

2.4.4.5 Spring Security Core

Spring Security Core (*v2.0-RC4*) *[26]* nos proporciona la funcionalidad necesaria para restringir el acceso a la aplicación. Mediante anotaciones podemos definir qué roles de usuarios pueden acceder a cada vista y controlador.

2.4.4.6 Spring Security LDAP

Spring Security LDAP (*v2.0-RC2*) *[27]* proporciona al módulo de seguridad acceso al *LDAP* de la compañía para buscar los usuarios que tienen acceso a la aplicación con sus permisos y otros datos como el nombre o el email.

2.4.4.7 Grails CSV

Grails CSV plugin (*v0.3.1*) *[28]* permite traducir y parsear datos a un fichero .csv. Se ha utilizado para exportar los datos de los candidatos que han sido entrevistados para un proceso de selección.

2.5 METODOLOGÍA UTILIZADA

Cuando se empezó a desarrollar software, los programadores escribían código casi sin detenerse a pensar en los requisitos que debían tener las aplicaciones, o la mejor forma de llegar a cumplirlos. Esto funcionaba más o menos bien para software que requería una estructura sencilla.

Fueron pasando los años y las aplicaciones se volvieron más complejas. Por lo que se empezó a definir el sistema antes de sentarse a desarrollarlo.

Con esto surgieron algunas metodologías de desarrollo, como pueden ser:

- *Cascada*: Es la más simple de todas, con un proceso secuencial se van pasando por las fases de diseño, implementación, pruebas, integración y mantenimiento. Es sencillo pero sólo sirve para software pequeño y simple.

- *Incremental*: Se centra en implementar una parte concreta de la aplicación dejando las demás para el futuro. Cuando se acaba esta, se desarrolla otra, y así hasta que el programa está completamente funcionando.

- *Espiral*: Se pasa por los distintos estados varias veces, mientras se está desarrollando. De esta forma tendremos un mayor control sobre la evolución de la aplicación, y ésta será más flexible y tolerante a cambios.

En los últimos años el tamaño y complejidad del software ha aumentado considerablemente, hasta el punto de que se hace imposible de desarrollar sin aplicar una metodología de desarrollo.

Cada vez están surgiendo más metodologías distintas a las tradicionales, denominadas procesos ágiles, que hacen que el desarrollo sea más flexible y seguro. Como por ejemplo:

- *Extreme Programming* (XP): Es de los más destacados. Se basa en un desarrollo iterativo o incremental, con pruebas unitarias y refactorización del código. De esta forma se pueden ir cambiando los requisitos a medida que se desarrolla la aplicación. Esta metodología también facilita el trabajo en grupo, ya que pueden trabajar varias personas en un mismo módulo y el código está más revisado.

- *Scrum*: Define un conjunto de prácticas y roles, que se usarán como punto de partida para definir el proceso de desarrollo a ejecutar durante el proyecto. Define también los sprint, que son fases del desarrollo que duran entre 1 y 4 semanas donde el equipo desarrolla y muestra una porción del programa.

- *Test Driven Development* (TDD): El "desarrollo guiado por pruebas" trata de crear casos de prueba a partir de la funcionalidad requerida, y en base a esto desarrollar el código. Además, se llevan a cabo refactorizaciones en el código a medida que se va desarrollando haciendo el sistema más mantenible y flexible. Muchas veces se complementa junto a otras metodologías antes citadas.

A menudo estas prácticas no son utilizadas, o no se toman en serio, cuando se desarrolla un sistema software, lo que lleva a una detección tardía de los errores, o lo que es peor, a no detectarlos hasta que la aplicación está en funcionamiento.

Esto suele ser así ya que se tiene el pensamiento de que es necesario mucho tiempo para aplicar estas metodologías, lo que se traduce en aumento de costes y disminución de los beneficios.

No solo los costes son importantes para las empresas, también los tiempos. Actualmente la competencia de los mercados es máxima, y sacar un producto un mes antes o después puede significar el éxito o el fracaso de una aplicación, lo que lleva a acortar drásticamente los plazos de entrega.

Este proyecto se centra en realizar una aplicación software para la gestión de currículums, poniendo en práctica la metodología ágil *Test Driven Development*, asegurando que su funcionalidad es la requerida por el cliente.

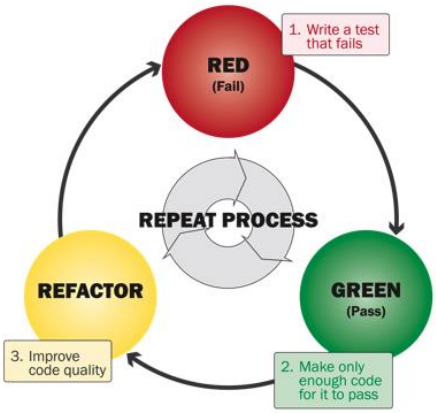


Ilustración 22. Diagrama de flujo habitual de TDD

2.6 TESTING DEL SOFTWARE

El testing es la forma que tenemos de medir y mantener la calidad del software para asegurar que los requisitos del cliente, la seguridad, confiabilidad y tolerancia a fallos se cumplen.

Con estos test no podremos demostrar que un programa funciona correctamente al 100%, ya que en la mayoría de los casos es imposible desarrollar las pruebas para todas las posibles variables. Por ejemplo, probar un método que realice la suma de dos enteros (de 32bits) tiene 2^{16} combinaciones posibles, con lo que desarrollar test para todas las combinaciones requeriría mucho tiempo y esfuerzo. Sin embargo, sí que nos ayudan a detectar los errores importantes y a aumentar la confianza que tenemos del código escrito.

Esta fase del desarrollo puede consumir hasta el 50% del coste total del sistema. Esta cifra podría aumentar en sistemas donde los errores son críticos, como en los sistemas bancarios. Es importante hacer bien y cuanto antes las pruebas, ya que cuanto más tarde se detecte un error, más caro resultará repararlo.

Por lo tanto, si le damos a las pruebas la importancia que merecen podremos detectar el mayor número de errores con la mínima cantidad de tiempo, esfuerzo y coste.

Hay distintos tipos de test *[29]* dependiendo del nivel de abstracción que tengamos. En la siguiente pirámide podemos ver estos tipos:

Ilustración 23. Pirámide de tipos de test

2.6.1 Test Unitarios

Las *pruebas unitarias*, en la base de la pirámide, sirven para comprobar el funcionamiento de cada módulo concreto del código (por ejemplo, un método de una clase). Asegurando así que cada módulo funciona correctamente por separado.

Al ejecutarse sobre un trozo de código perfectamente acotado, si el test falla será fácil encontrar el error y solucionarlo.

Estos test deben ser automatizables, sin necesidad de intervención humana para su ejecución. Además deben estar tan bien estructurados y comentados como el propio código del programa, ya que en el futuro podría ser necesaria una modificación.

A poco grande que se haga el proyecto, tendremos cientos de métodos que probar con varios casos de uso cada uno. La tarea de realizar pruebas unitarias a una aplicación de tamaño medio puede ser larga y tediosa, por lo que en general este tipo de test se realiza en los módulos clave del programa, aquellos que son claves en las actividades de negocio. Siguiendo con el ejemplo del sistema bancario, un método a probar con test unitarios será aquel que realice las transacciones de dinero, ya que una línea de código errónea podría hacer perder millones de euros.

Normalmente, los módulos que probamos necesitan interaccionar con otros para funcionar. A menudo, éstos no están aún implementados, o no podemos asegurar su correcto funcionamiento. Para desacoplar un módulo de otro en los test se utilizan los *Mocks* o *Stubs*.

Un *Mock* es un componente que imita el comportamiento de otros módulos del sistema, simulando la interacción entre el método bajo pruebas y los demás.

Por ejemplo, si estamos probando una clase *Usuario* y aún no tenemos las clases *DAO* para acceso a datos, podemos crear un *Mock* que simule un *UsuarioDAO*, indicando que al pedir un usuario devuelva “Pepito” y que al guardar un usuario devuelva “true”. De esta forma, podremos ejecutar la clase Usuario y recibir los datos ficticios que hayamos definido.

Mientras que los *Mocks* definen lo que se espera que el objeto bajo pruebas haga, los *Stubs* proporcionan respuestas predefinidas a los métodos de la clase que encapsulan*[30]*.

Por lo tanto, los *Mocks* nos permiten realizar lo mismo que los *Stubs*, pero con la funcionalidad añadida de poder definir qué métodos deberían ser llamados durante la ejecución del test, con qué argumentos, cuántas veces, etc.

Para este tipo de test se utilizan las pruebas de caja blanca, ya que debemos conocer la estructura del código para comprobar los distintos flujos de ejecución a probar.

2.6.2 Test de Integración

Las *pruebas integrales o de integración* tratan de juntar los módulos probados con los test unitarios, para probar que el conjunto funciona correctamente. Podemos probar desde dos módulos, hasta todos los que componen la aplicación. Prueban procesos simples y concretos, pasando por todas sus fases. Por ejemplo, el proceso de añadir un nuevo usuario.

Con estas pruebas también se pueden verificar requisitos no funcionales como el rendimiento, seguridad, usabilidad, fiabilidad, etc.

Por lo general, durante los test de integración también es necesario la utilización de *Mocks* para encapsular los módulos bajo pruebas.

2.6.3 Test Funcionales

Los *test funcionales* comprueban que las funcionalidades definidas durante el diseño de la aplicación han sido realizadas correctamente.

Se realizan siempre desde el punto de vista del usuario, no de los módulos del sistema. Por lo que el que realiza estas pruebas no necesita conocer la estructura del software.

Para este tipo de test se suelen realizar pruebas de caja negra. Estas se llevan a cabo sobre la interfaz de la aplicación, obviando el comportamiento interno. Con esto podremos detectar funcionalidades incorrectas o ausentes, errores de datos, errores de presentación y errores de rendimiento.

2.6.4 Test de Aceptación

Estas pruebas se realizan en el entorno de uso del cliente, para comprobar que el sistema es válido para él. Se deja que sea el propio usuario de la aplicación quien realice casos de uso reales, sacando así posibles fallos tanto de programación como de diseño.

Un ejemplo de realización de estos test podría ser poner en marcha la aplicación en un servidor de pruebas y pedir a uno o varios usuarios finales que la utilicen, pasando por la mayoría de requisitos definidos y observando los posibles fallos o dificultades que se puedan presentar.

La realización de test de aceptación es muy importante para determinar si el producto final es realmente el que se pedía y necesitaba el cliente.

2.6.5 Test de Regresión

Con las nuevas metodologías ágiles con desarrollos iterativos, donde se realizan cambios progresivos en el programa, surgen también los *test de regresión*. Estos test tratan de alertar de cualquier error o carencia de funcionalidad causados por un cambio en el software.

Este tipo de errores suelen darse al añadir nuevas funcionalidades o al llevar a cabo refactorizaciones en el código ya implementado y probado.

Las pruebas de regresión pueden estar dentro de los tipos anteriormente descritos (unitarios, integración, funcionales o aceptación) y se ejecutan cada cierto tiempo, comprobando que los cambios no han afectado al resto del programa.

Muchas veces estos test se automatizan con un servidor de *Integración Continua* (*CI*). Este servidor automatiza la tarea de la ejecución de los test. Nosotros vamos subiendo los cambios que hagamos en la aplicación y los test periódicamente (por ejemplo, cada día). Mientras tanto, el servidor se encarga de ejecutar todos los test de la aplicación, avisándonos si hay algún fallo.

Esto nos permite también tener una versión de la aplicación sin errores, que podemos poner en producción o enseñar a un cliente en cualquier momento.

2.7 TEST DRIVEN DEVELOPMENT (TDD)

Como ya se ha introducido, la metodología *Test Driven Development* se basa en la idea de realizar las pruebas antes de implementar el código del programa. Con esto se crea el código necesario para pasar los test. Por último, una vez pasadas las pruebas, se refactoriza el código de forma que quede más limpio y mantenible.

Desarrollando los test al principio del proyecto nos obligamos a pensar en la funcionalidad requerida, y no en el código que debe pasar las pruebas. Asegurando así que el código que pase dichas pruebas cumple con las funcionalidades requeridas.

Además, al desarrollar pronto las pruebas, si cometemos un error y cambiamos algo de lo que ya teníamos hecho nos daremos cuenta rápido al pasar nuestros tests. A esto se le llaman pruebas de regresión.

2.7.1 Ciclo de desarrollo

Para trabajar con la metodología *TDD*, debemos primero, definir con el cliente las funcionalidades requeridas, creando un diagrama de casos de uso y definiendo la lista de requisitos que ha de cumplir la aplicación. Hecho esto, seguiremos un ciclo de pasos:

1. Elegir un requisito concreto y fácil de implementar.

2. Escribir las pruebas de aceptación para ese requisito. Para ello, el programador debe entender las especificaciones acordadas con el cliente y ponerse en la situación del usuario cuando vaya a utilizar la aplicación.

3. Verificar que la prueba falla, ya que puede darse el caso de que la prueba sea errónea o que el requisito ya estuviese implementado.

4. Escribir la implementación para ese requisito de forma que pase las pruebas establecidas. Es importante desarrollar siempre un código limpio y ordenado.

5. Ejecutar todas las pruebas desarrolladas, verificando que tanto el nuevo módulo construido como lo que ya estaba hecho funciona correctamente.

6. Una vez pasadas las pruebas, refactorizamos el código para hacerlo más claro y eficiente, pasando de nuevo los test para no cometer errores.

7. Por último, se actualiza la lista de requisitos, eliminando el ya implementado y añadiendo otros nuevos que puedan ir surgiendo durante el desarrollo del software.

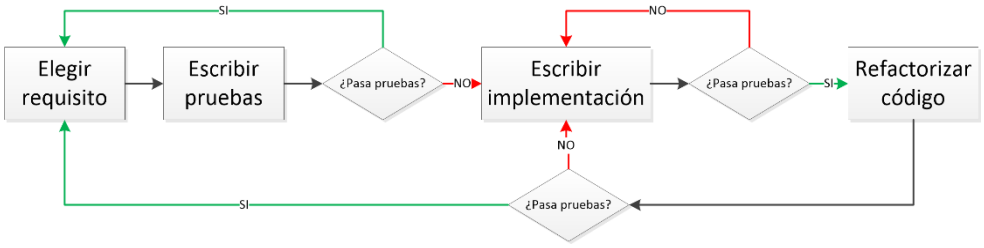


Ilustración 24. Esquema TDD

Los test, por lo tanto, no se realizan únicamente al final para comprobar la funcionalidad del software, sino que se trabaja con ellos durante todas las fases de desarrollo. En el siguiente gráfico se muestra el peso que tienen los test en cada fase de desarrollo:

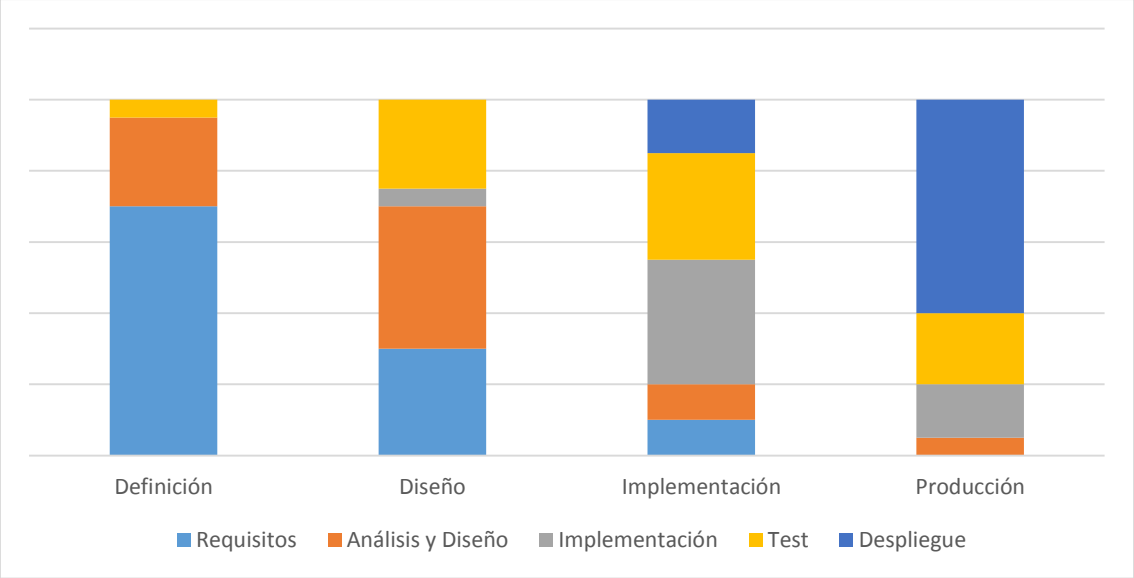


Ilustración 25. Fases de TDD

2.7.2 Los test también son documentación

Para desarrollar las pruebas en la metodología *TDD* planteamos primero cómo tiene que funcionar la aplicación en base a los requisitos, y con esto implementamos los métodos de test. Al hacer esto, no sólo se especificamos cómo queremos que se use el módulo, sino que además puede ejecutarse.

Por lo que no sólo es importante para no cometer errores, sino que puede ayudar en el futuro a otros desarrolladores, o incluso a nosotros mismos, a saber modificar o mejorar el sistema.

2.7.3 Ventajas

Las principales ventajas que presenta el uso de esta metodología son:

- El programador raramente tiene que utilizar el *debugger* o depurador para encontrar fallos en el código, ya que las pruebas de integración están bien definidas.
- Aplicaciones de mayor calidad en menos tiempo.
- Mayor confianza en el código.
- Complementa la documentación del sistema.
- Con la refactorización conseguimos que el código sea más legible, óptimo, y por lo tanto, fácil de mantener

2.7.4 Desventajas

Como todo, también trae consigo una serie de desventajas:

- Debemos tener una amplia experiencia en el desarrollo de pruebas para ser productivos.

- Para utilizar esta metodología deberemos automatizar las pruebas. Esto puede resultar complejo en algunas capas del desarrollo, como son:

    - Interfaces Gráficas de Usuario (*GUI*).

    - Objetos distribuidos.

    - Bases de datos.

Sin embargo, ya existen algunas herramientas que nos pueden ayudar con estas tareas:

- *Selenium*, por ejemplo, es una herramienta que permite crear pruebas funcionales para aplicaciones en entornos web. Ofrece una extensión para exploradores, la cual nos permite grabar y ejecutar las pruebas. Con esto, además de otros requisitos, comprobamos que la interfaz gráfica generada es correcta.
- También existen *Mocks*, que se centran en ejecutar pruebas para módulos o secciones concretas del sistema. Sirve para probar objetos distribuidos por separado, bases de datos, módulos que interactúan con otros aún sin implementar, o de aquellos que no disponen aún de las pruebas que verifiquen su comportamiento.
- Por último tenemos las *Bases de Datos en Memoria*. Estas bases de datos se inicializan vacías cada vez que arrancamos la aplicación, y permiten introducir unos datos conocidos para realizar las pruebas y después verificar los datos generados.

2.8 PRINCIPIO DE PARETO

El *Principio de Pareto [31]*, también conocido como la regla del 80-20 y definido por Vilfredo Pareto, dice que el 80% de los efectos proceden del 20% de las causas. Aplicando esto al esfuerzo, con el 20% del esfuerzo se consiguen el 80% de resultados.

Intuitivamente podemos pensar que al realizar un esfuerzo del 50%, conseguiremos un 50% de resultados, sin embargo esto no es así. Tampoco resulta siempre en un 80-20, es algo que puede variar, pero la realidad es que la proporción casi siempre es muy distinta al 50-50.

Siguiendo este principio, podemos decir que el 20% (más o menos) del programa, tendrá un 80% de importancia en comparación con el resto del sistema. Con lo que podemos concluir que deberíamos centrarnos en encontrar qué módulos componen ese 20% para darle una prioridad temporal y de recursos.

Por lo general, cuando desarrollamos un sistema un poco grande, no podemos hacer una definición perfecta antes de empezar a implementarlo, ya que a medida que vamos desarrollándolo nos irán surgiendo nuevos requisitos y funcionalidades que implementar. Aplicando el *principio de pareto*, podríamos pensar primero cuáles son las partes más importantes del sistema, para empezar a desarrollar por ahí y después ir amoldando la definición a lo que nos vaya surgiendo por el camino.

También podemos aplicar este concepto en otras áreas de la metodología de desarrollo que implementemos. Por ejemplo, si tenemos una fecha de entrega, lo mejor sería hacer primero el 20% del programa que más importancia y más utilización va a tener. De esta forma si no llegamos a la fecha límite podremos entregar un programa con la funcionalidad suficiente para su uso hasta que se complete el sistema.

El proyecto realizado presentaba algunas funcionalidades básicas que se debían cumplir para la entrega final, como son introducir nuevos candidatos, guardar su información, y tener un buscador bastante flexible. Desarrollando este 20% de la aplicación estaremos cubriendo el 80% del uso de la misma, por lo que, a estos módulos, se les ha dado una prioridad tanto de tiempo como de recursos.

En la primera versión de la aplicación, se implementó el 50% del software requerido, incluyendo ese 20% más importante. De esta forma se pudieron iniciar los test en producción mientras se completaba el resto del software.

3 DESARROLLO DEL PROYECTO

Con el objetivo de minimizar riesgos, gestionar cambios de forma eficaz y ofrecer un servicio de calidad que cumpla con las expectativas del proyecto, se estableció una ronda de diálogos con los empleados del departamento de *RRHH* y con el gerente de la empresa, que además ha supervisado cada etapa de desarrollo.

En una primera toma de contacto se expuso la necesidad de una aplicación que gestione toda la información relativa a los candidatos que pretenden un puesto a la empresa.

Una vez claras las principales funcionalidades requeridas, se comenzaron a diseñar los requisitos y estructuras que tomaría el proyecto.

En las siguientes secciones se expondrá el análisis del problema y la solución llevada a cabo para la finalización de este proyecto final de grado.

3.1 ANÁLISIS DEL SISTEMA

En este apartado se estudian las funcionalidades que requería la empresa *CIC*. Aquí se especifica la funcionalidad que el sistema de información final debe implementar.

Para ello, primero se llevó a cabo una identificación de los diferentes escenarios que se puedan dar, así como del personal que hará uso del sistema.

A continuación, se realizó una especificación de casos de uso, a partir de la cual se obtuvieron los requisitos tanto funcionales como no funcionales que debe cumplir la aplicación.

3.1.1 Identificación de Usuarios

La aplicación está destinada, principalmente, para agilizar las actividades desarrolladas en el departamento de *RRHH* de *CIC*, por lo que los principales usuarios serán los empleados de este departamento. Dentro de este grupo, tendrá que haber una (o varias) persona(s) que puedan administrar el contenido de la aplicación y sus parámetros.

Además, otros empleados de la empresa tendrán acceso a su ficha personal para cambiar detalles de su currículum e información personal.

Por lo tanto distinguimos tres tipos de usuario:

- *Empleado*. Puede acceder a su ficha personal para actualizar su información profesional.

- *Personal de RRHH*. Puede acceder a la gestión de personas, convocatorias y entrevistas.

- *Administrador* (o gerente) de *RRHH*. Además, puede configurar los parámetros de la aplicación.

A continuación se detallan las diferentes actividades que puede realizar cada tipo de usuario.

3.1.1.1 Empleado

Cualquier empleado de la empresa puede acceder a su ficha para ver y modificar parte de su información, como por ejemplo:

- Información personal, como la fecha de nacimiento, teléfonos, email, web personal o domicilio.

- Información curricular, como por ejemplo las habilidades, titulaciones, certificados o experiencia.

3.1.1.2 Personal de RRHH

El empleado normal será el usuario natural de la aplicación. Se encargará de las siguientes tareas:

- Gestionar candidatos, creando nuevos y añadiendo su información, como estados, CVs, capacidades, alertas, etc.

- Gestionar convocatorias, creando nuevas cuando surjan, añadiendo la información necesaria y seleccionando candidatos.

- Gestionar entrevistas, añadiendo evaluadores, ejecutando la entrevista con el candidato y guardando una valoración.

- Hacer búsquedas de candidatos, con distintos filtros.

- Ver estadísticas de los candidatos que han llegado en el último año.

3.1.1.3 Administrador

El administrador, además de poder realizar las tareas de los usuarios anteriores, también se encargará de añadir todos datos de configuración, que son: titulaciones, departamentos, evaluadores, habilidades, certificaciones, orígenes de los cvs, estados y redes sociales.

3.1.2 Especificación de casos de uso

En ingeniería de software, un caso de uso proporciona un posible uso de la aplicación en un momento determinado, dentro de un contexto determinado, llevado a cabo por un usuario determinado. Generalmente, se utiliza para definir el comportamiento de la aplicación con un diagrama fácil de interpretar para una persona no técnica. Con que suele utilizarse para definir la funcionalidad con el cliente.

Para simplificar el diagrama se ha utilizado un patrón de casos de uso que llamaremos *CRUD*. Este patrón nos permite especificar una o varias de las acciones *Create* (Crear), *Read* (Leer), *Update* (Modificar) y *Delete* (Borrar) definiéndolo en un único caso de uso. Estos casos de uso estarán definidos por las iniciales de las acciones *CRUD* pudiendo tener una o varias de ellas, seguida por la entidad sobre la que actúan.

En el siguiente diagrama se muestran los diferentes casos de uso que implementa la aplicación web:

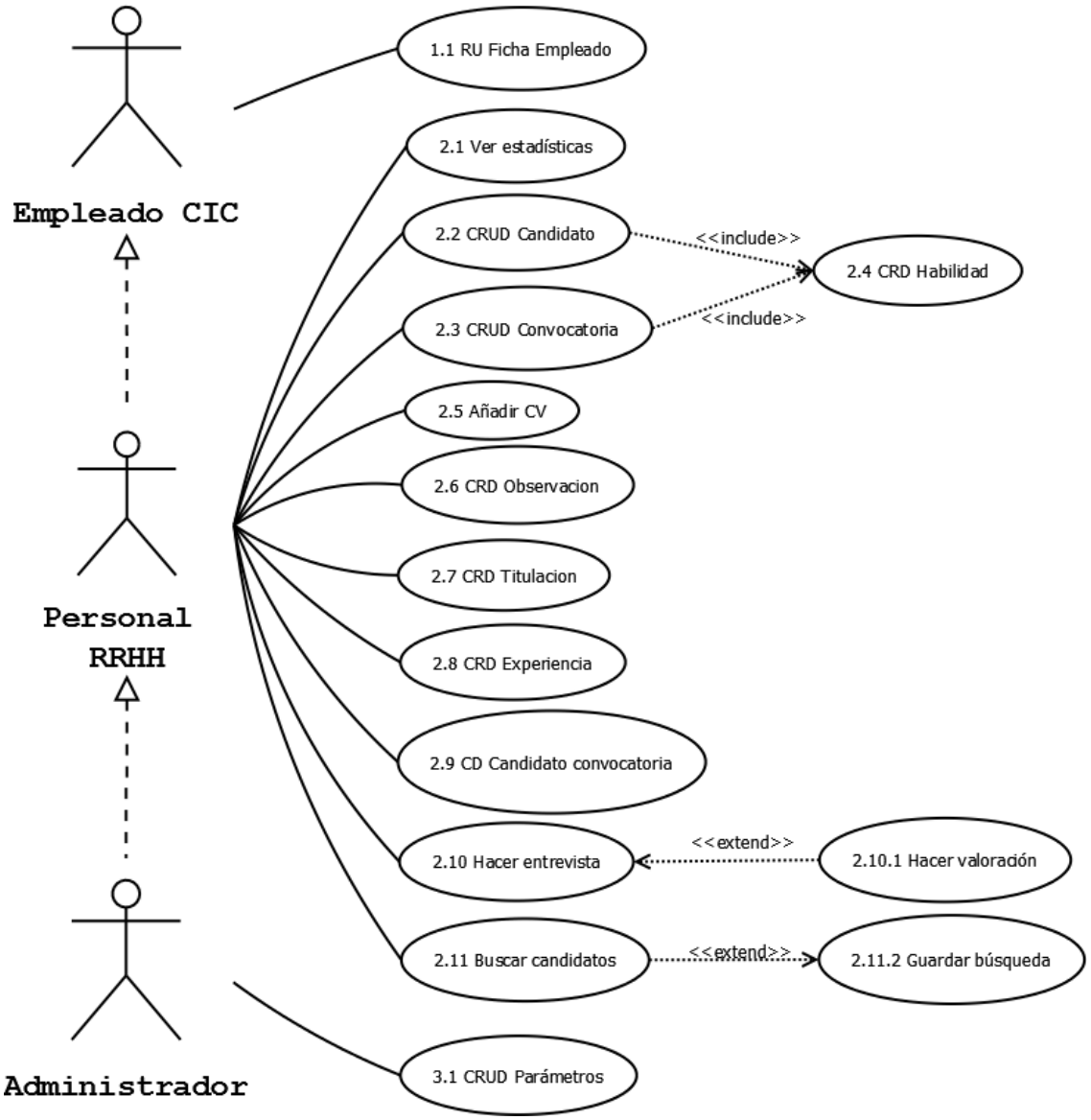


Ilustración 26. Diagrama de Casos de Uso del Sistema

Con este diagrama aclaramos lo ya dicho en el punto anterior, donde se explicaba que el empleado normal sólo puede ver y modificar su ficha, el empleado de *RRHH* puede ver y modificar toda la información sobre candidatos, convocatorias y entrevistas, y el administrador además puede modificar parámetros de la aplicación como los departamentos, titulaciones, departamentos, etc.

3.1.3 Especificación de requisitos

El objeto de la especificación es definir de manera clara y precisa todas las funcionalidades y restricciones del sistema que se desea construir. Los requisitos deben ser medibles, comprobables y sin ambigüedades o contradicciones.

A continuación se especifican los requisitos que debe cumplir el sistema. Estos se dividen en dos grandes bloques:

- *Funcionales*, aquellos requisitos que determinan la funcionalidad del software, esto es, todas las acciones que pueden realizar los diferentes roles de la aplicación.

- *No funcionales*, aquellos requisitos que determinan las restricciones o condiciones de otros aspectos como la calidad, estándares, costes, estabilidad y portabilidad.

El formato y los datos proporcionados en la especificación de requisitos se han basado en el estándar *IEEE Std 830-1998 [32]*. Se presentan con el siguiente formato:

Identificación [RF-X]	**[Nombre del requisito]**
	[Descripción del requisito]
	[Prioridad del requisito] Alta/Requerido ☐ Media/Deseable ☐ Baja/Opcional ☐

Ilustración 27. Formato Requisitos del Sistema

En esta tabla se describe la relación que hay entre los requisitos funcionales y los casos de uso determinados en el punto *3.1.2*.

Ilustración 28. Relación Casos de uso y Requisitos

3.1.4 Requisitos Funcionales

Los requisitos funcionales definen todas las acciones que deben ser implementadas en el sistema. Para una mayor claridad, el identificador del requisito indica a qué usuario se corresponde. De forma que:

- Los requisitos del 100 al 199 corresponden al rol de *Empleado de RRHH*.
- Los requisitos del 200 al 299 corresponden al rol de *Administrador*.
- Los requisitos del 300 al 399 corresponden al rol de Empleado de la empresa.

A continuación se especifican los requisitos funcionales del sistema para cada rol de empleado.

3.1.4.1 ROL PERSONAL RRHH

RF-101	**Ver estadísticas**
	La página principal de la aplicación deberá mostrar estadísticas relacionadas con los datos que maneja
	Alta/Requerido ☐ Media/Deseable ☒ Baja/Opcional ☐

RF-102	**Crear, ver y modificar candidato**
	El sistema deberá permitir añadir, ver y modificar candidatos
	Alta/Requerido ☒ Media/Deseable ☐ Baja/Opcional ☐

RF-103	**Borrar candidato**
	El sistema deberá permitir borrar candidatos
	Alta/Requerido ☐ Media/Deseable ☐ Baja/Opcional ☒

RF-104	**Añadir, ver y borrar domicilios a candidato**
	El sistema deberá permitir añadir, ver y modificar candidatos
	Alta/Requerido ☒ Media/Deseable ☐ Baja/Opcional ☐

RF-105	**Añadir, ver y borrar redes sociales al candidato**
	El sistema deberá permitir añadir, ver y borrar redes sociales a los candidatos
	Alta/Requerido ☐ Media/Deseable ☐ Baja/Opcional ☒

RF-106	**Actualizar estado de candidato**
	El sistema deberá dejar elegir entre actualizar el estado de los candidatos de forma automática en base a sus trabajos y titulaciones actuales o de forma manual eligiendo entre los estados del sistema
	Alta/Requerido ☐ Media/Deseable ☒ Baja/Opcional ☐

RF-107	**Ver fecha actualización**
	El sistema deberá indicar la fecha de la última vez que se modificó la información de cada candidato en su ficha
	Alta/Requerido ☐ Media/Deseable ☒ Baja/Opcional ☐

RF-108	**Ver timeline de candidato**
	El sistema deberá permitir ver en una línea temporal los elementos más importantes de cada candidato
	Alta/Requerido ☐ Media/Deseable ☐ Baja/Opcional ☒

RF-109	**Descartar candidato**
	El sistema deberá permitir descartar un candidato enviándolo así a una “lista negra” para que no salga en las búsquedas
	Alta/Requerido ☐ Media/Deseable ☒ Baja/Opcional ☐

RF-110	**Añadir, ver y modificar convocatorias**
	El sistema deberá permitir añadir, ver y borrar convocatorias
	Alta/Requerido ☒ Media/Deseable ☐ Baja/Opcional ☐

RF-111	**Borrar convocatoria**
	El sistema deberá permitir borrar convocatorias
	Alta/Requerido ☐ Media/Deseable ☐ Baja/Opcional ☒

RF-112	**Mostrar candidatos añadidos a la convocatoria**
	El sistema deberá permitir ver los candidatos que se han apuntado en cada convocatoria
	Alta/Requerido ☐ Media/Deseable ☒ Baja/Opcional ☐

RF-113	**Añadir, ver y borrar habilidad a candidato y convocatoria**
	El sistema deberá permitir asignar y eliminar capacidades con un nivel del 1 al 10 (incluidos) a cada candidato y a cada convocatoria
	Alta/Requerido ☒ Media/Deseable ☐ Baja/Opcional ☐

RF-114	**Añadir, eliminar y descargar Currículum Vitae**
	El sistema deberá permitir añadir, ver y descargar archivos con el currículum vitae de los candidatos.
	Alta/Requerido ☒ Media/Deseable ☐ Baja/Opcional ☐

RF-115	**Añadir, ver y eliminar observaciones a candidatos**
	El sistema deberá permitir añadir, ver y eliminar observaciones a los candidatos
	Alta/Requerido ☐ Media/Deseable ☒ Baja/Opcional ☐

RF-116	**Añadir y ver titulaciones de candidatos**
	El sistema deberá permitir añadir y ver las titulaciones de los candidatos
	Alta/Requerido ☒ Media/Deseable ☐ Baja/Opcional ☐

RF-117	**Borrar titulación a candidato**
	El sistema deberá permitir borrar titulaciones a los candidatos
	Alta/Requerido ☐ Media/Deseable ☒ Baja/Opcional ☐

RF-118	**Añadir y ver experiencias de candidatos**
	El sistema deberá permitir añadir y ver la experiencia de los candidatos
	Alta/Requerido ☒ Media/Deseable ☐ Baja/Opcional ☐

RF-119	**Borrar experiencia a candidato**
	El sistema deberá permitir borrar experiencias a los candidatos
	Alta/Requerido ☐ Media/Deseable ☒ Baja/Opcional ☐

RF-120	**Añadir y eliminar candidatos de convocatorias**
	El sistema deberá permitir añadir y eliminar candidatos a cada candidatura
	Alta/Requerido ☒ Media/Deseable ☐ Baja/Opcional ☐

RF-121	**Añadir entrevista a candidato**
	El sistema deberá permitir añadir los datos generados durante una entrevista
	Alta/Requerido ☒ Media/Deseable ☐ Baja/Opcional ☐

RF-122	**Añadir documento a entrevista**
	El sistema deberá permitir añadir documentos adjuntos a la entrevista
	Alta/Requerido ☐ Media/Deseable ☒ Baja/Opcional ☐

RF-123	**Añadir valoración de entrevista**
	El sistema deberá permitir añadir una valoración a las entrevistas realizadas
	Alta/Requerido ☒ Media/Deseable ☐ Baja/Opcional ☐

RF-124	**Buscar candidatos**
	El sistema deberá permitir buscar candidatos por información personal, capacidades y/o experiencia
	Alta/Requerido ☒ Media/Deseable ☐ Baja/Opcional ☐

RF-125	**Añadir filtros**
	El sistema deberá permitir añadir filtros de edad, experiencia, género y estado a la búsqueda
	Alta/Requerido ☒ Media/Deseable ☐ Baja/Opcional ☐

RF-126	**Buscar nivel habilidad**
	El sistema deberá permitir buscar por el nivel de la habilidad, con los signos (> < =)
	Alta/Requerido ☐ Media/Deseable ☒ Baja/Opcional ☐

RF-127	**Buscar en el árbol de habilidades**
	El sistema deberá permitir crear un árbol de habilidades, de forma que al buscar una habilidad también devuelva como resultado los candidatos que poséen las habilidades superiores.
	Alta/Requerido ☒ Media/Deseable ☐ Baja/Opcional ☐

3.1.4.2 ROL ADMINISTRADOR

RF-201	**CRD Habilidades**
	El sistema deberá permitir crear, listar y borrar posibles habilidades que pueden tener los candidatos
	Alta/Requerido ☒ Media/Deseable ☐ Baja/Opcional ☐

RF-202	**CRD Departamento**
	El sistema deberá permitir crear, listar y borrar los distintos departamentos de la organización
	Alta/Requerido ☒ Media/Deseable ☐ Baja/Opcional ☐

RF-203	**CRUD Titulación**
	El sistema deberá permitir crear, listar y borrar las posibles titulaciones de los candidatos.
	Alta/Requerido ☒ Media/Deseable ☐ Baja/Opcional ☐

RF-204	**CRD Orígenes**
	El sistema deberá permitir crear, listar y borrar los orígenes de los currículums de los candidatos.
	Alta/Requerido ☐ Media/Deseable ☐ Baja/Opcional ☒

RF-205	**CRD Certificaciones**
	El sistema deberá permitir crear, listar y borrar las distintas certificaciones de los candidatos.
	Alta/Requerido ☐ Media/Deseable ☐ Baja/Opcional ☒

RF-206	**CRD Estados**
	El sistema deberá permitir crear, listar y borrar los distintos estados de los candidatos.
	Alta/Requerido ☐ Media/Deseable ☐ Baja/Opcional ☒

RF-207	**CRD Redes**
	El sistema deberá permitir crear, listar y borrar las distintas redes sociales de los candidatos.
	Alta/Requerido ☐ Media/Deseable ☐ Baja/Opcional ☒

3.1.4.3 ROL EMPLEADO

RF-301	**RU Ficha empleado**
	El sistema deberá permitir a cada empleado ver y modificar su información personal.
	Alta/Requerido ☐ Media/Deseable ☒ Baja/Opcional ☐

3.1.5 Requisitos No Funcionales

Los requisitos no funcionales no se refieren a los servicios que proveerá el sistema, sino que especifican cómo debe rendir la aplicación.

RNF-01	**Diseño responsive**
	El sistema deberá adaptarse a los diferentes dispositivos web.
	Alta/Requerido ☒ Media/Deseable ☐ Baja/Opcional ☐

RNF-02	**Usabilidad**
	La navegación en la web deberá ser intuitiva y amigable, de forma que los usuarios no necesiten una información previa. Para ello, se debe poder acceder a toda la información en menos de 5 clics.
	Alta/Requerido ☐ Media/Deseable ☒ Baja/Opcional ☐

RNF-03	**Acceso restringido**
	El sistema deberá comprobar que el usuario de la aplicación pertenece a la empresa y que tiene permisos para acceder.
	Alta/Requerido ☒ Media/Deseable ☐ Baja/Opcional ☐

RNF-04	**Compatibilidad de navegadores**
	El sistema deberá ser navegable desde los navegadores web más comunes, como son: - Internet Explorer (>Versión 11.0) - Mozilla Firefoc (Versión 20.0) - Google Chrome (Versión 28) - Safari (versión 5.1.7) - Otros navegadores para móviles.
	Alta/Requerido ☒ Media/Deseable ☐ Baja/Opcional ☐

RNF-05	**Accesibilidad**
	El tiempo de carga de cada página no debe ser mayor a 1 segundo.
	Alta/Requerido ☒ Media/Deseable ☐ Baja/Opcional ☐

3.2 DISEÑO ARQUITECTÓNICO O LÓGICO

A continuación se describe el diseño realizado para la aplicación. Para ello se especifica la arquitectura que se va a utilizar y los diferentes componentes de la misma.

La arquitectura define el diseño a más alto nivel de la aplicación. Con esto tendremos los distintos componentes software y la forma en que interaccionan entre sí.

Para escoger la arquitectura adecuada debemos tener en cuenta un gran número de aspectos de la aplicación a desarrollar; como el tamaño, seguridad, metodología, etc.

Los tipos de patrones arquitectónicos más comunes son:

- *Cliente-servidor* (dos capas): Se trata de un modelo de aplicación distribuida en el que las tareas se separan entre una capa de persistencia y otra de usuario.
- *Objetos distribuidos*: Se compone de objetos que proveen servicios a otros objetos y usan servicios de otros objetos.
- *Peer-to-peer*: Sistemas descentralizados. Las computaciones pueden ser de cualquier dispositivo de la red con cualquier otro dispositivo de la red.
- *Modelo-vista-controlador* (tres capas): Utiliza 3 capas para separar los componentes de persistencia, los de las interfaces, y los modelos de negocio.

Para el presente proyecto se decidió utilizar el *Modelo-Vista-Controlador*, por ser el más recomendado en el desarrollo aplicaciones web, ya que al separar las diferentes funciones del programa en tres capas, hace más fácil su desarrollo y pruebas a la vez que dota al sistema de una mayor seguridad.

Además, permite un desarrollo modular y escalable, por lo que resulta más fácil añadir nuevas funcionalidades o cambiar un módulo por otro.

3.2.1 Modelo Vista Controlador (MVC)

El modelo vista controlador es un patrón de arquitectura software que separa la aplicación en distintas capas para hacerla más modular, testeable y segura, entre otras cosas.

Las capas de las que consta son las siguientes:

- *Modelo*: Gestiona los accesos que se realizan a la base de datos que contiene la información que utiliza la aplicación.
- *Vista*: Esta es la capa con la que la aplicación interactúa con el usuario. Esta es la capa con la que interactúa el usuario (normalmente una interfaz gráfica).
- *Controlador*: Contiene toda la lógica de negocio, recogiendo los datos necesarios del modelo y pasando la información necesaria a las vistas.

Para cada implementación pueden diferir algunos detalles en la implementación y en cómo se comunica cada capa. En el caso de esta aplicación, el flujo de control que sigue es el siguiente:

1. El usuario interactúa con la interfaz de usuario desde su navegador web. (por ejemplo, el usuario envía un formulario).
2. El controlador recibe la notificación de la acción solicitada por el usuario. Éste gestiona el evento que llega, frecuentemente a través de un gestor de eventos (handler) o callback.
3. El controlador realiza las operaciones pertinentes con los datos y accede al modelo actualizando la base de datos con la nueva información.
4. El controlador genera la vista de acuerdo a la petición del usuario, que es enviada a su navegador web.
5. La interfaz de usuario espera nuevas interacciones, comenzando el ciclo de nuevo.

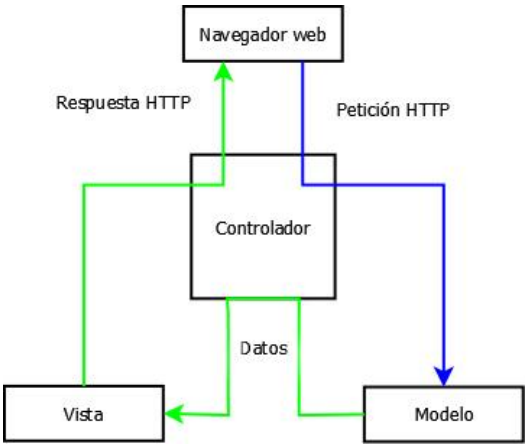


Ilustración 29. Esquema arquitectura MVC

En nuestro proyecto, el controlador se comunica también con un servidor de dominio, el cual tiene el directorio activo accesible mediante *LDAP* (*Lightweight Directory Access Protocol*). Este almacena la información de autenticación (usuario y contraseña) y fue utilizado para autenticar los empleados de la empresa de manera integrada al resto de entornos, aplicaciones y sistemas que estos utilizan.

Con todo esto, se construyó un diagrama de despliegue que expone los diferentes componentes que intervienen en el proyecto y cómo interaccionan entre sí.

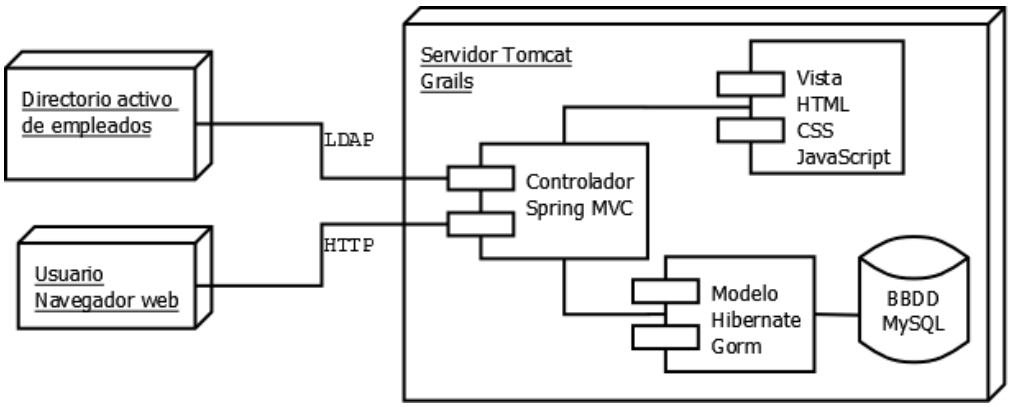


Ilustración 30. Diagrama de Despliegue del Sistema

El siguiente diagrama de componentes muestra cómo se ha diferenciado cada capa del modelo vista controlador.

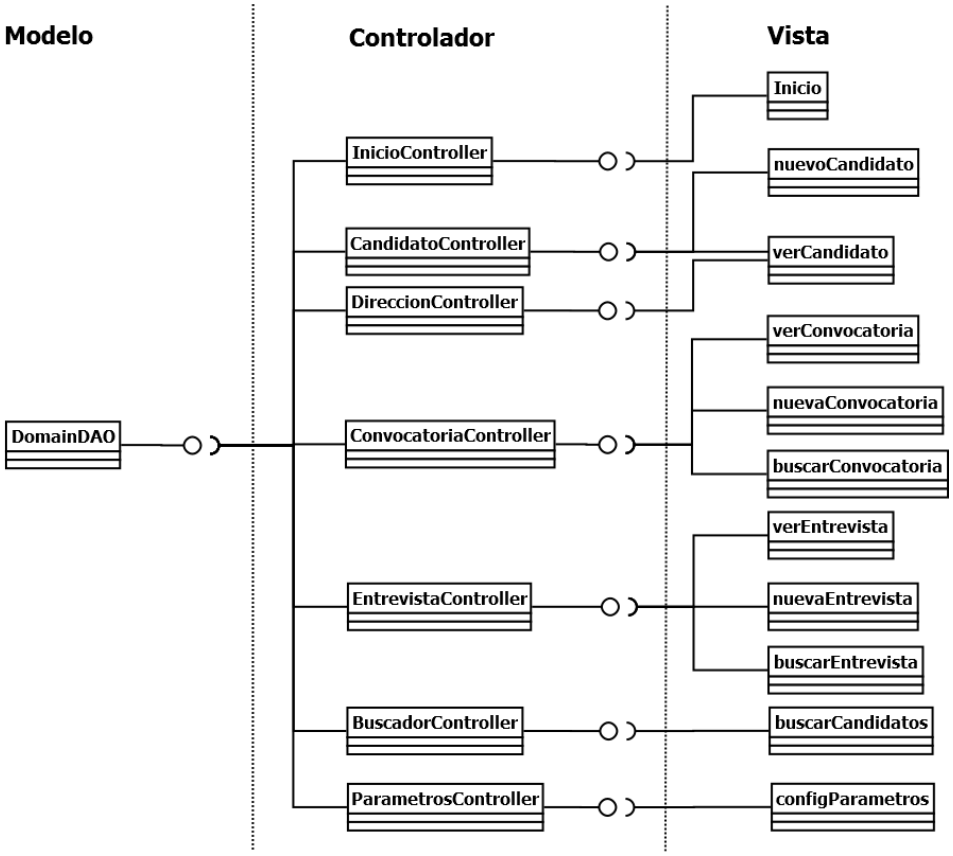


Ilustración 31. Diagrama de Componentes del Sistema

Las clases de la capa de dominio (*dominioDAO*) se han omitido ya que son demasiadas como para incluirlas en un diagrama de componentes. Además, éstas son gestionadas por el *ORM* (*Object Relational Maping*) de *Grails*, *GORM*, por lo que no son realmente relevantes de cara a la implementación.

3.3 IMPLEMENTACIÓN

Ahora que hemos elegido la metodología y la arquitectura web que vamos a emplear para el desarrollo del proyecto es hora de ponerlo en práctica.

En esta sección se expone cómo se ha implementado cada capa de la aplicación, para lo que se siguió el siguiente ciclo de trabajo:

- En primer lugar se implementa el modelo, ya que con este definimos los datos necesarios a almacenar y a mostrar posteriormente.
- Después, se tomó un requisito cada vez, escribiendo sus test correspondientes.
- Por último se implementó el controlador y la vista necesarios para dotarlo de la funcionalidad prevista.

3.3.1 Modelo

Como ya hemos introducido, una vez decidida la arquitectura a aplicar, se realizó el diseño y la implementación del modelo de datos. Éste es el encargado de guardar todos los datos que maneja la aplicación, así como sus tipos, relaciones y restricciones.

La persistencia en *Grails* se realiza, por defecto, mediante *GORM*, que utiliza *Hibernate*. Esto permite generar la *BBDD* a partir de las clases de dominio de nuestra aplicación.

En este caso se eligió *MySQL* como base de datos de la aplicación, por lo que se creó una base de datos en el entorno de desarrollo y otro en el de producción. También se configuraron los archivos de *Grails* necesarios para el acceso a ellas.

Gracias a la convención sobre la configuración de la que se habló en el punto *2.4.1*, lo único que debemos hacer para crear un objeto de dominio es crear una clase pojo dentro de nuestra aplicación con sus atributos y restricciones.

Un ejemplo de cómo mapea *Grails* un objeto de dominio es el siguiente:

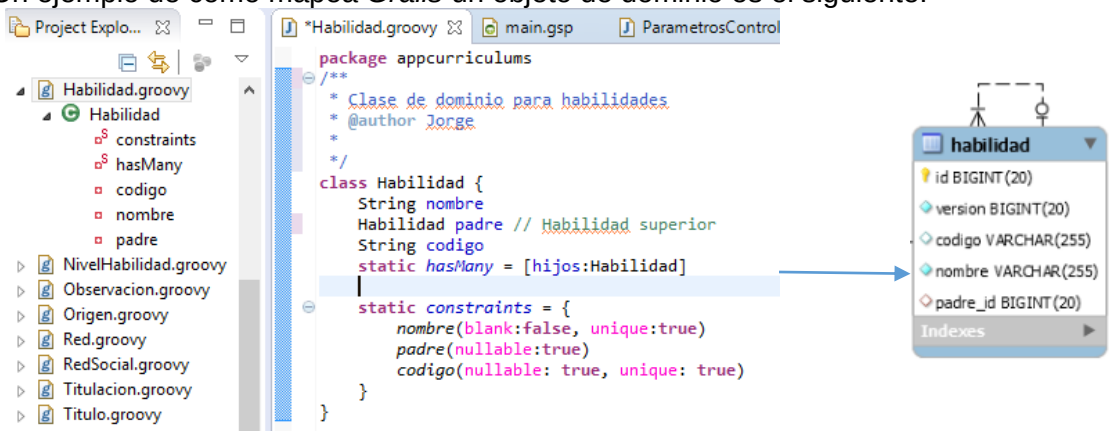


Ilustración 32. Ejemplo de mapeo mediante GORM

Al ser *Hibernate* quien mapea el modelo de datos debemos tener presentes las convenciones de nombres y restricciones que tiene *Grails*, para que construya la base de datos según nuestro *modelo ER* concreto.

Un ejemplo de las dificultades que pueden surgir es cómo mapear la herencia de clases de Java, ya que tenemos 3 formas:

- Una tabla para toda la jerarquía de clases.
- Una tabla por cada subclase.
- Una tabla para cada clase y subclase.

Podemos usar una u otra cambiando una sola línea de código, pero puede significar grandes diferencias en cuanto a la rapidez de funcionamiento del programa, incluso puede dar lugar a inconsistencias. Por lo que es muy recomendable tener siempre en cuenta cómo mapeamos nuestro modelo.

Una vez creado todo el modelo de datos y haciendo ingeniería inversa sobre la base de datos generada obtenemos el diagrama de tablas, que podemos ver en el *Anexo I*.

Los principales objetos de dominio sobre los que funciona el sistema de información son: *candidato*, *convocatoria* y *entrevistas*; todos estos unidos por *candidaturas*.

Como ya introdujimos en el apartado anterior, GORM es quien se encarga de proporcionarnos métodos de acceso al modelo. Para ello, nos genera una serie de métodos basados en sus convenciones de nombres.

Por ejemplo, si queremos sacar el candidato llamado "*Jorge*" y apellidado "*Andrés*", simplemente debemos hacer una llamada a este método autogenerado:

```
Candidato.findByNombreAndApellido("Jorge", "Andrés")
```

3.3.2 Test

En el punto *2.7* se explicó la metodología TDD, donde se exponía la importancia de crear un amplio rango de pruebas para cada módulo del software, de forma que aseguremos que éstos implementan correctamente todos los requisitos aprobados en la fase de diseño.

En esta metodología la finalidad de las pruebas es asegurar que se implementan correctamente todos los casos de uso, y además que cualquier cambio del código no afecte a otros requisitos ya implementados.

Dentro de los tipos de pruebas que hemos visto en la sección *2.6,* los que más se adecúan a esta finalidad son los *test funcionales*. Con estos test podremos asegurar que las actividades que va a realizar el usuario final están implementadas, además de avisarnos si modificamos erróneamente alguna clase que afecte a la implementación de otro requisito.

Aunque en menor medida, también se han utilizado *test unitarios* para comprobar el estado de la base de datos y en otros módulos más críticos y complejos de la aplicación, como es el buscador.

Es importante tener claros los pasos a seguir a la hora de realizar los test, ya que si se nos olvida probar un caso de prueba podría tener graves consecuencias con el software ya en producción.

Para ello, los pasos que se han seguido han sido los siguientes:

1. Por cada requisito, se han identificado los escenarios que tiene, eligiendo los más relevantes.
2. A continuación se han identificado todas las variables de entrada que deben tener.
3. Por cada variable de entrada, se han identificado los diferentes valores que pueden tomar, haciendo hincapié en los valores límite y especiales (como los valores límite, tamaños de Strings, el número 0, etc).
4. Se han especificado valores reales para cada caso de prueba definido en el paso anterior.
5. Se han combinado las opciones de valores de entrada identificados anteriormente para crear así los casos de prueba.
6. Por último, se han escrito los test pertinentes.

A modo de ejemplo, se explican los pasos seguidos para el desarrollo de los casos de prueba para los test funcionales del requisito *RF-101, Crear Candidato*.

RF-101	**Crear candidato**
	El sistema deberá permitir añadir candidatos
	Alta/Requerido ☒ Media/Deseable ☐ Baja/Opcional ☐

Como hemos definido anteriormente, el primer paso fue identificar los distintos escenarios que tiene. Se describió con la siguiente plantilla:

	Paso	**Descripción**
Escenario Principal **A: Actor** **S: Sistema**	1	A: Entra al formulario para añadir un candidato e introduce los datos
	2	S: Valida los datos
	3	S: Muestra la ficha del nuevo candidato
Extensiones	2b	Algún dato no es válido S: Vuelve a mostrar el formulario para añadir el candidato, proporcionando los errores correspondientes

Los dos siguientes pasos identifican y definen las variables de entrada. Para el caso de uso de crear un nuevo candidato, necesitamos definir sus datos personales:

Paso	**Variable**	**Restricciones**				
1	**Nombre**	String	Null			Blank
1	**Apellido1**	String	Null			Blank
1	**Apellido2**	String		Null		
1	**Fecha Nacimiento**	Fecha		Null		
1	**Genero**	Correcto		Null		
1	**DNI**	Tamaño=9	Null	Tamaño=8		Repetido
1	**Telefono1**	Tamaño=9		Tamaño=8	Null	“ “
1	**Telefono2**	Tamaño=9		Tamaño=8	Null	“ “
1	**E-mail**	Válido		Invalido (sin @)		
1	**Web**	Válido		Invalido (sin http://)		

A continuación, se especificaron valores reales para cada variable:

Paso	Variable	Restricciones					
1	**Nombre**	“Luis”	Null	“ “			
1	**Apellido1**	“Pérez”	Null	“ “			
1	**Apellido2**	“De Andrés”	Null				
1	**Fecha Nacimiento**	“26/05/1990”	Null				
1	**Genero**	Hombre	Mujer	Null			
1	**DNI**	“123456789”	Null	“12345678”	“123456789”		
1	**Telefono1**	“123456789”	“12345678”	Null	“ “		
1	**Telefono2**	“123456789”	“12345678”	Null	“ “		
1	**E-mail**	“email@gmail.com”	“emailgmail.com”				
1	**Web**	“http://www.web.com”	“web.com”				

El siguiente paso es combinar los posibles valores de cada variable para crear los casos de prueba:

RF-101	Casos de prueba			
Variable	CP 1	CP 2	CP3	CP 4
Nombre	“Luis”	“Luis”	Null	“ “
Apellido1	“Pérez”	“Pérez”	Null	“ “
Apellido2	“De Andrés”	“De Andrés”	Null	“ “
Nacimiento	“26/05/1990”	“26/05/1990”	Null	-----
Genero	Hombre	Mujer	Null	-----
DNI	“123456789”	Null	“123456789”	“12345678”
Telefono1	“123456789”	“12345678”	Null	“ “
Telefono2	“123456789”	“12345678”	Null	“ “
E-mail	“email@gmail.com”	“emailgmail.com”	Null	“ “
Web	“http://www.web.com”	“web.com”	Null	“ “

Ahora que hemos generado los casos de prueba a implementar con sus valores correspondientes, podemos crear una clase de test funcionales.

Para ejemplificarlo utilizaremos el *CP 2*, el cual debe devolver un error, ya que el DNI es obligatorio. Por lo tanto estamos ante un caso de prueba de la extensión *2b* de nuestra plantilla.

```
public class CandidatoCP2Test extends AppcurriculumsTest{
        private final int sleep = 500; // Espera de carga del navegador

        @Test
        public void testCandidato() throws Exception {
            //Entramos en la pagina principal
            driver.get(baseUrl + "/inicio/inicio");

            // Creamos un nuevo candidato
            driver.findElement(By.id("btn_candidato")).click();
            driver.findElement(By.id("btn_nuevoCandidato")).click();
            driver.findElement(By.id("nombre")).clear();
            driver.findElement(By.id("nombre")).sendKeys("Luis");
            driver.findElement(By.id("ape1")).clear();
            driver.findElement(By.id("ape1")).sendKeys("Pérez");
            driver.findElement(By.id("ape2")).clear();
            driver.findElement(By.id("ape2")).sendKeys("De Andrés");
            new Select(driver.findElement(By.id("f_nacimiento_day"))).selectByVisibleText("26");
            new Select(driver.findElement(By.id("f_nacimiento_month"))).sel.selectByIndex(5);
```

```
            new Select(driver.findElement(By.id("f_nacimiento_year"))).
                                                    selectByVisibleText("1990");
            driver.findElement(By.name("genero")).click();
            driver.findElement(By.id("dni")).clear();
            driver.findElement(By.id("dni")).sendKeys(""); // Dato incorrecto
            driver.findElement(By.id("tlf1")).clear();
            driver.findElement(By.id("tlf1")).sendKeys("123456789");
            driver.findElement(By.id("tlf2")).clear();
            driver.findElement(By.id("tlf2")).sendKeys("123456789");
            driver.findElement(By.id("email")).clear();
            driver.findElement(By.id("email")).sendKeys("email@gmail.com");
            driver.findElement(By.id("web")).clear();
            driver.findElement(By.id("web")).sendKeys("http://www.web.com");
            driver.findElement(By.id("create")).click();

            try {// Verificamos que muestra el error de dato incorrecto
                assertTrue(driver.findElement(By.xpath("//div[@id='msgErr']/div")).
                        getText().contains("Debe especificar el DNI del candidato"));
            } catch (Error e) {verificationErrors.append(e.toString());}
        }
}
```

Como podemos comprobar, lo que implementa este test es, mediante el driver de *Selenium*, abrir el explorador web para acceder a la página principal de la aplicación, emulando hacer clic sobre el enlace del menú para crear un nuevo candidato. Después el driver introducirá todos los valores en el formulario y clicará en el botón guardar, que tiene asignado el id "*create*".

Por último, se comprobará que el programa, en lugar de crear el candidato, muestra un mensaje de error "*Debe especificar el DNI del candidato*", indicando que para crear un nuevo candidato debemos especificar el DNI.

3.3.3 Vista

Con el modelo ya definido y construido, se plantearon las diferentes vistas que debía tener la aplicación. Estas interfaces gráficas de usuario (*GUI*) fueron diseñadas para facilitar la interacción del usuario con el sistema, siguiendo una estructura simple y limpia.

Como ya hemos comentado en el punto *2.2.6*, se ha utilizado un tema de *Bootstrap* para hacer más sencilla la tarea de construcción de la interfaz. Esta herramienta nos ha permitido crear una aplicación web visualmente atractiva, a la vez que adaptable a todos los tamaños de dispositivos mediante *diseño responsive*.

Tras una búsqueda y prueba de temas adecuados, se eligió el *SB Admin 2 [33]*, ya que el formato es adecuado para aplicaciones de gestión, aprovechando todo el espacio de pantalla y disponiendo de menús bien definidos.

Podemos ver una captura de la interfaz de la aplicación en el *Anexo II*, concretamente la página principal del sistema.

Para comprobar que la usabilidad de las interfaces se ajusta al perfil profesional del personal de *RRHH*, se realizaron sesiones de validación periódicas para verificar las funcionalidades y proponer mejoras.

En el siguiente mapa web se describen las distintas interfaces gráficas que se pactaron para la aplicación.

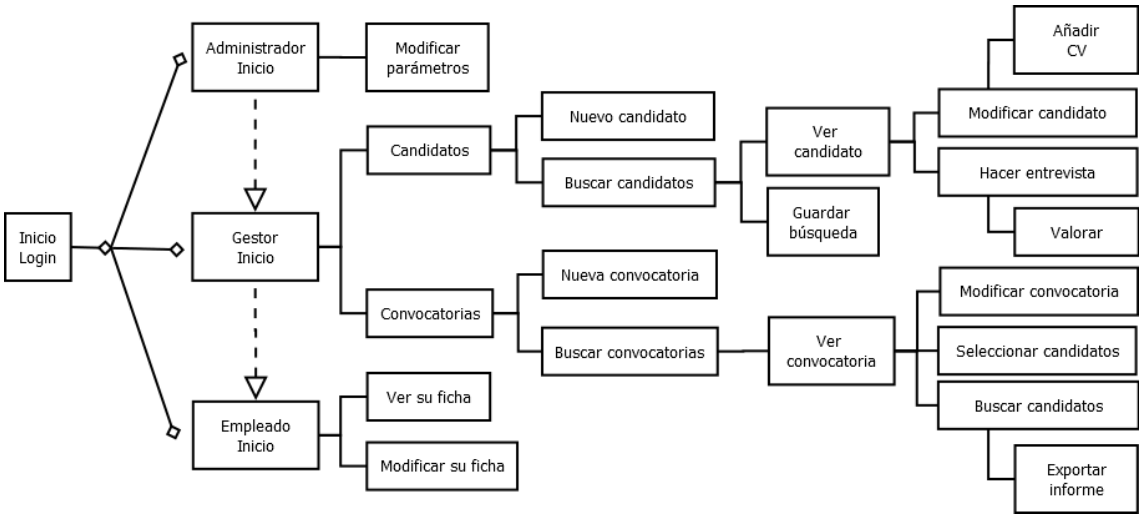


Ilustración 33. Mapa Web de la Aplicación

En *Grails* hay dos tipos de componentes para las vistas, las *views* o vistas normales y las *templates* o plantillas.

Una *view* define una página en concreto con todo su contenido, mientras que una plantilla es un componente de la interfaz, que se podrá insertar en la *view* para actualizar contenido asíncronamente o para reutilizarlo en varias páginas.

Tanto para una como para la otra utilizamos una mezcla de tecnologías, como son: *HTML5*, *CSS3* (en otro archivo), *JavaScript* y *GSP* (*Groovy Server Pages*). Este último es la tecnología que utiliza Grails para crear las vistas. De forma muy similar a las tecnologías *ASP* y *JSP* se ejecuta en el servidor para generar las vistas.

Un ejemplo de plantilla, es el siguiente:

```
<g:each var="capacidad" in="${candidato.capacidades.sort{a,b ->
                (a.nivel > b.nivel) ? -1 : 1}}">

        <span class="btn btn-outline btn-primary habilidad">
                ${capacidad.habilidad.nombre}
                (${capacidad.nivel)})
                <i onclick="borrarCapacidad(${capacidad.id})"
                        class="fa fa-times-circle">
                </i>
        </span>

</g:each>
```

En este código recorremos todas las habilidades o capacidades de un candidato para mostrarlas en la interfaz web. El resultado es el siguiente:

Al ser una plantilla, podemos insertarla en cualquier vista para sacar las habilidades de un candidato, y así no tener código duplicado.

3.3.4 Controlador

Como se ha explicado anteriormente, el controlador es el encargado de realizar las operaciones de negocio de la aplicación, recogiendo las peticiones de los usuarios, cogiendo los datos del dominio (modelo) y generando la vista que se devolverá al cliente.

En la siguiente imagen se detallan los distintos controladores que se expusieron en la fase de diseño, con sus correspondientes acciones.

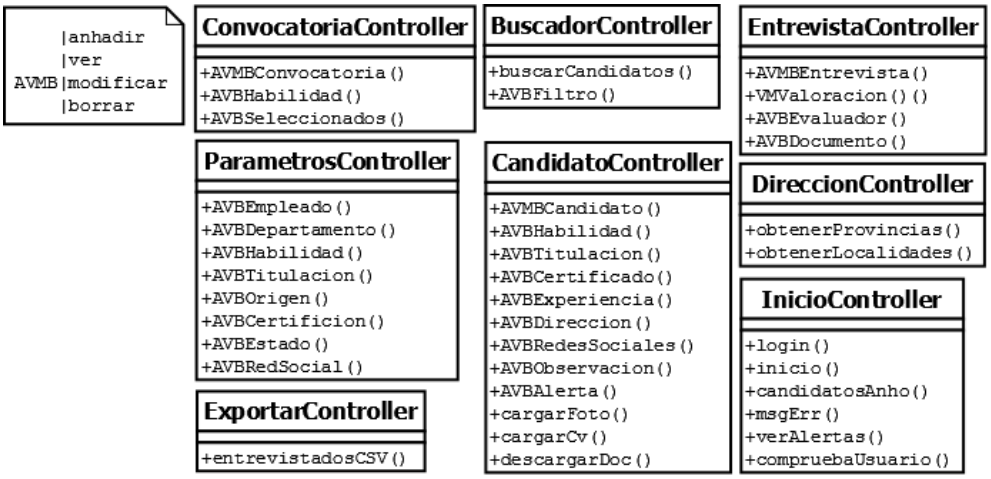


Ilustración 34. Controladores del Sistema

El código de los controladores es posiblemente el más complejo de toda la aplicación, por lo que se han tenido muy en cuenta las buenas prácticas de programación.

Se ha intentado que cada método tenga el menor número de líneas posibles, separando su contenido en varios, si es necesario por su complejidad funcional. También se ha comentado cada método para ayudar a entender que tareas realizan facilitar el mantenimiento y escalabilidad posterior.

A continuación se muestra la acción del controlador *CandidatoController* que añade una nueva *alerta* a un *candidato*:

```
/**
 * Anhade una alerta a un candidato
 * @return template: verAlertas. Si hay error, template: msgErr + verAlertas
 * @param idCandidato - id del candidato
 * @param f_alerta - fecha de la alerta
 * @param descripcion - descripcion de la alerta
 */
@Secured(['ROLE_ADMIN', 'ROLE_GESTOR'])
def anhadirAlerta() {
        // Recogemos los datos
        def c =                 Candidato.get(params.idCandidato)
        def texto =             params.texto
        def f_alerta =          params.f_alerta

        // Creamos la alerta
        def alerta = new Alerta(candidato: c, texto: texto, f_alerta: f_alerta,
                                email: "email@gmail.com")

        // Validamos los campos
```

```
        if (!alerta.validate()) {
                def errores = []
                alerta.errors.allErrors.each {
                        errores.add(messageSource.getMessage(it, null))
                }

                render(template: "/msgErr", model: [errores: errores])
        }else {
                // Anhadimos la alerta
                c.addToAlertas(alerta)
                c.f_actualizacion = new Date()
                c.save(flush:true)
        }

        // Devolvemos la vista generada
        render(template: "verAlertas", model: [candidato: c])
}//anhadirAlerta
```

Como podemos ver, el control de acceso en los controladores se realiza mediante anotaciones, y es el plugin de *Spring Security* quien lo gestiona. En este caso pueden añadir alertas el administrador y el gestor.

Dentro del método se recogiendo los datos, creando la alerta. Si todo es correcto, se añade el objeto creado a la base de datos mediante los métodos que nos proporciona *GORM* y se devuelve el *template* de alertas. Si hay algún dato erróneo, se devuelve la misma plantilla pero con mensajes de error.

4 EVALUACIÓN EN PRODUCCIÓN

En este apartado se explica cómo se desarrolló la puesta en producción del producto y las pruebas de aceptación y rendimiento realizadas.

4.1 PUESTA EN PRODUCCIÓN

Una vez desarrollados la gran mayoría de requisitos, se desplegó el software en el servidor de producción. Este servidor cuenta con un *Red Hat Enterprise Linux 6* con *2GB* de *RAM*, *2CPU´s amd64* y *40 GB* de *disco duro*.

El despliegue se realizó mediante la herramienta *Putty* (para el uso de *SSH* y *SCP*). Primero, se creó un usuario en el gestor de bases de datos, con el que se autentica la aplicación. A continuación, se importó la base de datos de la aplicación mediante el siguiente comando:

```
mysql –u usuario_mysql nombre_bbdd < fichero.sql
```

El siguiente paso fue desplegar el archivo *.war* generado de nuestra aplicación en la carpeta de despliegue del Tomcat. Esto se realizó mediante *FTP* con el programa *WinSCP*.

Para poner en marcha el servidor, otra vez con la herramienta *Putty*, se arrancó el *Tomcat* con el siguiente comando:

```
/etc/init.d/tomcat start
```

Con la base de datos en funcionamiento y la aplicación desplegada, se realizó la carga de datos inicial con todos los parámetros de *Titulaciones*, *Departamentos*, *Habilidades*, etc.

El versionado de la aplicación, que utiliza una estructura de 3 niveles típica (*major*, *minor*, *revision*), establece que el número de esta versión inicial es la *v1.0.0*.

Hecho esto, se realizaron reuniones de formación a los gestores de la aplicación, de manera que su adaptación al sistema sea lo más inmediata y eficaz posible.

Al final del desarrollo también se entregó un manual de usuario para facilitar la incorporación de nuevos empleados.

4.2 PRUEBAS DE ACEPTACIÓN

Una vez finalizada la primera versión de la aplicación y la puesta en producción de la misma, se dio paso a un proceso de aceptación de la misma.

Como mencionamos en la sección 2.6.4, las pruebas que verifican el cumplimiento de las funcionalidades requeridas por el cliente se llaman *pruebas de aceptación*.

Para ello, siguiendo la normativa de desarrollo de aplicaciones de *CIC* (bajo la metodología *CMMI nivel 2*), se establece un documento que recoge las pruebas a realizar y el resultado de las mismas. Este documento se llama PPI (Plan de Pruebas e Implantación) y contiene, a modo de fichas de trabajo una enumeración de los requisitos que la aplicación debe satisfacer y las pruebas que se van a realizar en cada uno de los casos para verificarlos.

Un ejemplo de plantilla que figura en dicho documento es la siguiente:

Módulo: Buscador de candidatos				
Requisito	Pruebas Relacionadas	Cumple		Comentarios
		SI	NO	
RF-124 … 	3.12.1. Buscar por nombre 3.12.2. Buscar por DNI 3.12.3. Buscar por apellido 3.12.4. Buscar por titulación …	X X X X 	 	Verificar candidato sin segundo apellido

Resultado de las pruebas del Módulo		
Correcto X Incorrecto	Personal de CIC : firma y fecha	Personal del cliente firma y fecha

La elaboración del documento *PPI* se realiza como parte del proyecto, de tal manera que primero se verifica su cumplimiento en el entorno que se ha utilizado para desarrollar el trabajo. Una vez superadas todas las pruebas que figuran en el *PPI* en el entorno de desarrollo, es cuando procede el paso al entorno de producción, donde nuevamente se verifican.

El documento *PPI,* finalmente aceptado, figura en el sistema de gestión de proyectos de *CIC*. Como datos destacados del mismo, cabe indicar en líneas generales que:

- Número de requisitos objetos de la prueba: 40
- Número de pruebas medio realizado por requisito: 5
- Número de pruebas totales: 212

La responsabilidad sobre la ejecución y aprobación del *Plan de Pruebas* suelen ser por parte de *CIC*, del *Tester/Analista* y/o *Jefe de Proyecto*. En este caso fue revisado por el Jefe de Proyecto *José Miguel Prellezo*. Una vez que dada su conformidad, se comenzaron a realizar todas las pruebas especificadas en el documento.

El documento *PPI* tiene un número de versión que se actualiza en cada revisión del documento. Cuando hay nuevos requisitos, o bien se enriquece el conjunto de pruebas a realizar, se genera una nueva versión de dicho documento. La última versión realizada es la *v4.0*.

Una vez completadas las pruebas de aceptación automatizadas, el departamento de *RRHH* se dispuso a probar y testear la herramienta en este entorno, proporcionando nuevas funcionalidades y cambios. Para ello, se limpiaron todos los registros de la *BBDD* para dejar el sistema completamente operativo para los usuarios del mismo.

Como se introdujo en el punto anterior, como complemento de las pruebas realizadas, se inició una fase de soporte inicial en el arranque de la herramienta, en la cual se ayudó de una forma cercana a los gestores en el uso del sistema.

CIC había iniciado recientemente un proceso de selección de candidatos para uno de los cursos de formación que imparte, por lo que se dieron las condiciones idóneas para contrastar en un caso real el correcto funcionamiento del sistema a modo de certificar la validez de las *pruebas de aceptación*.

Todos los pasos que se describen a continuación fueron realizados por el personal de recursos humanos (*gestores* y *administradores*), ya que son los principales perfiles que la empresa ha definido para el uso de la aplicación.

Para empezar, se creó un nuevo proceso de selección con toda la información relativa al curso de formación que se iba a impartir. Se añadieron también los candidatos que ya habían entregado su currículum a la empresa, así como los que fueron llegando más tarde. Mientras tanto, cada vez que se realizaba una entrevista a un candidato, también se incluía en el perfil del mismo, enlazándolo con el proceso de selección. Una vez cerrada la fecha límite para apuntarse al curso, el responsable del departamento de *RRHH* realizó la selección de los candidatos para el curso, en base a toda la información recogida días anteriores.

Durante todo este proceso, que duró un mes, se fueron comentando las sensaciones de navegación y las funcionalidades del software con los usuarios del mismo.

De las reuniones salieron algunos errores que corregir, así como nuevas funcionalidades que antes no habían surgido. Por ejemplo, después de probar la aplicación surgió la necesidad de guardar los certificados que tiene cada candidato. Funcionalidad de la que no se había hablado hasta ese momento, y que finalmente ha sido implementada.

Siguiendo el criterio metodológico de *CIC*, los nuevos requisitos se recogieron en el *DRF* (*Documento de Requisitos Funcionales)* que fue aprobado por la dirección del proyecto. Igualmente, se actualizó el documento *PPI* para recoger pruebas asociadas a los nuevos requisitos.

El ciclo de despliegue y pruebas asociadas se repitió nuevamente hasta dejar finalmente operativa la versión actual (1.1.0).

Con la implantación total del proyecto y la finalización del mismo, podemos afirmar que la aplicación se adapta perfectamente a las necesidades del departamento de recursos humanos.

4.3 PRUEBAS DE RENDIMIENTO

Paralelamente a las pruebas de aceptación de los requisitos se realizaron algunas pruebas de rendimiento. Con estas se buscaba verificar que se han cumplido los requisitos no funcionales.

Una de las pruebas realizadas fue el acceso con distintos tamaños de navegador para probar su adaptabilidad, lo que llamamos el *diseño responsive*. Se accedió desde pantallas tamaño PC, Tablet y Smartphone, comprobando que todas las funcionalidades son accesibles sin dificultades.

Uno de los fallos típicos de diseño es utilizar un único navegador web durante el desarrollo de la aplicación. Éstos tienen algunas diferencias en la forma que interpretan los estilos visuales, por lo que se puede ver diferente en unos y otros. Para evitar esto, también se ha accedido desde los navegadores: *Google Chrome*, *Firefox*, *Internet Explorer*, *Opera* y *Safari*.

Como se comentó en el apartado *2.4.3*, *Grails* es una tecnología bastante pesada en el entorno de desarrollo, ya que necesita muchos recursos para la recarga automática de recursos. Realiza re-compilaciones “al vuelo” para facilitar el cambio de código y el despliegue, por lo que ralentiza la ejecución y, a veces, las páginas pueden tardar en cargar 2 o 3 segundos.

Una vez puesto el software en producción, se realizaron pruebas desde varias máquinas accediendo a distintas vistas de la aplicación, verificando que el tiempo de carga es, en la mayoría de los casos, de menos de 1 segundo.

En cualquier caso, la experiencia de usuario para una herramienta de gestión de este tipo ha sido plenamente satisfactorio.

5 CONCLUSIONES Y TRABAJOS FUTUROS

Finalizado el desarrollo del proyecto, se analizaron los resultados del mismo, así como las posibles mejoras que puedan realizarse en el futuro. En los siguientes apartados se plantean las conclusiones tanto técnicas sobre el proyecto, como personales. Además se muestran las mejoras que se podrían realizar en la aplicación, para añadir importantes funcionalidades al software desarrollado.

5.1 CONCLUSIONES

Tras la gran aceptación de la aplicación por parte de los usuarios CIC, podemos afirmar que se ha cumplido con el principal objetivo del proyecto, la creación de un sistema que ayude a la empresa a gestionar la información profesional de sus empleados y de los candidatos a puestos de empleo o cursos.

La aplicación web se encuentra en producción desde abril del presente año y contiene en su base de datos alrededor de 200 candidatos y 9 procesos de selección abiertos. El personal de *RRHH* accede periódicamente al sistema para actualizar la información de entrevistas, añadir nuevos candidatos, actualizar perfiles, y en definitiva, gestionar la relación del personal y candidatos con el departamento.

Se han implementado todos los requisitos con una prioridad alta, la mayoría con prioridad media y casi la totalidad con prioridad baja, con que se espera que la aplicación satisfaga las necesidades del departamento de *RRHH*.

Tras utilizar el framework *Groovy on Grails,* se ha comprobado que su productividad, una vez se entiende cómo funciona el framework, es muy alta. La gestión de dependencias con *Maven* y la *gestión de la configuración* hacen que añadir nuevos módulos al sistema sea una tarea sencilla y rápida. Por otra parte, la reparación de algunos bugs provocados por las tecnologías que maneja *Grails* por debajo es bastante tediosa. Con todo esto, podemos concluir que el framework elegido ha sido el adecuado, ya que nos ha permitido realizar el proyecto requerido con los recursos proporcionados.

Por otro lado, se ha aplicado la metodología ágil *Test Driven Development*, comprobando las grandes ventajas que aporta en cuanto a seguridad del código y de la correcta implementación de los requisitos definidos.

Respecto a la utilización de esta metodología, podemos destacar que para obtener el máximo rendimiento se debe tener cierta experiencia en los distintos tipos de pruebas disponibles. Probablemente sea más efectiva aún para grupos de trabajo con varias responsabilidades en sus miembros, ya que los test son más efectivos cuando la persona que los realiza no es la misma que programa el módulo bajo pruebas. Esto es así ya que se minimizan los vicios inherentes al programador.

En el plano formativo, destacar que he podido poner en práctica gran parte de los conocimientos adquiridos en el estudio de la ingeniería. En estos últimos cursos estudiamos en detalle las etapas del desarrollo de un sistema informático: diseñar modelos de datos, crear interfaces acordes a los requisitos de los usuarios, configurar servidores y aplicar patrones de diseño a nuestra arquitectura, entre otras cosas.

Sin embargo, nunca había completado todo el proceso de desarrollo de un proyecto software desde el diseño inicial, hasta los posteriores mantenimientos. Verlo finalmente en explotación, mejorando las actividades de negocio de la empresa, otorga verdadera satisfacción profesional como Ingeniero Informático.

Además, la estancia en la empresa *CIC* ha contribuido ampliamente en la adquisición de nuevos conocimientos, experiencias y oportunidades para mi futuro laboral.

La informática está presente en todo tipo de sectores y organizaciones, por lo que cada proyecto realizado proporciona nuevos conocimientos acerca de las actividades de negocio en las que se basa. En este proyecto concreto, he podido comprobar cómo evalúa el departamento de *RRHH* las entrevistas que se realizan para proveer de personal un puesto laboral concreto, lo que seguro me ayudará también en un futuro próximo.

Por todo ello, califico esta experiencia como altamente positiva, ya que me ha ayudado a organizar y poner en práctica los conocimientos adquiridos en mis estudios de Grado en Ingeniería Informática.

5.2 TRABAJOS FUTUROS

Una aplicación de estas características nunca da por finalizado su proceso de creación. Periódicamente se solicitan nuevas funcionalidades que pueden ayudar a mejorar la rapidez y eficiencia con la que trabajan los empleados.

A continuación se exponen mejoras que fueron sugeridas durante y después del desarrollo del proyecto y que podrían ser implementados en el futuro.

5.2.1 Creación de modelos de entrevista

Una idea que surgió en las primeras fases del proyecto, fue crear un apartado para realizar las entrevistas a los candidatos directamente en el propio sistema.

El departamento de *RRHH*, actualmente dispone de unas plantillas en papel con preguntas típicas a modo de formulario, que se van cumplimentando durante las entrevistas a los candidatos.

Sería de gran ayuda poder crear “modelos de entrevistas” para cada perfil de empleado típico, y aplicar un modelo concreto en la configuración de cada entrevista a realizar. Esto permitiría añadir la información directamente en la aplicación sin tener que utilizar el papel y el tiempo del evaluador como intermediarios.

Sin embargo, por motivos de disponibilidad de tiempo y porque no era un requisito prioritario, no se ha implementado. Por el contrario se ofreció las funcionalidades de almacenar las conclusiones de cada entrevista y anexar al sistema las hojas escaneadas como archivos adjuntos.

Posiblemente las entrevistas tipificadas, serían una de las funcionalidades que primero se considerarían requisitos del proyecto, en caso de ser ampliado.

5.2.2 Integración con el Gestor de Proyectos.

Obviamente, esta no es la única aplicación que está desarrollando la empresa para mejorar la gestión interna.

Existe otro proyecto, llamado *MOPA*, cuya finalidad es gestionar los partes de trabajo del empleado. En esta herramienta se introducen semanalmente las horas que emplea el trabajador en cada proyecto, por lo que podríamos conocer las habilidades que va desarrollando. Por ejemplo, si un empleado ha estado trabajando 600h en un proyecto en *Java*, podríamos actualizar la información de nuestra aplicación, asignando a esa persona un nivel de 7 en dicha habilidad, realizando una equivalencia entre las horas de desarrollo y las correspondientes habilidades.

Se desarrollaría un servicio *REST*, de forma que pueda *MOPA* actualizar las habilidades de los empleados que gestiona nuestra aplicación, como se ha expuesto antes.

5.2.3 Formación complementaria del candidato

Una de las últimas funcionalidades que surgió durante las reuniones en las pruebas de aceptación es la posibilidad de añadir cursos complementarios a los candidatos, aparte de las titulaciones oficiales que tienen. Esto completaría la sección de conocimientos, junto a las habilidades, titulaciones y la experiencia.

5.2.4 Internacionalización

El producto ha sido desarrollado para una empresa en concreto, que es CIC. Sin embargo, puede haber otras que quieran disponer de una herramienta con estas características.

Con sólo cambiar algunos parámetros de configuración e imágenes, se podría implementar en otra empresa, incluso en el extranjero. Es posible internacionalizarla gracias al soporte *i18n* de *Grails*, el cual permite cargar las vistas en distintos idiomas de forma automática.

5.2.5 Mantenimiento de la aplicación

Como cualquier otro proyecto, en el futuro deberá realizarse un seguimiento del funcionamiento de la aplicación, para corregir posibles errores o mejorar su rendimiento.

Además de mejorar su funcionalidad, como ya se ha explicado en apartados anteriores, puede ser necesaria una refactorización del código para hacer este más eficiente y rápido.

En ocasiones, puede ser necesario también un cambio del entorno tecnológico, como por ejemplo del servidor o incluso del gestor de la base de datos.

A pesar de que este proyecto ha sido basado en test para diseñar cada requisito, pueden aparecer errores inesperados. Por esto, se ha solicitado a los usuarios de la aplicación que proporcionen un feedback del uso de la aplicación, para poder reportar cualquier bug y solventarlo lo antes posible.

6 REFERENCIAS

[1] *Web corporativa de eProwin* http://cv.eprowin.com/

[2] *Web de Groovy* http://www.groovy-lang.org/

[3] *Web de Grails* https://grails.org/

[4] *Web de Maven* https://maven.apache.org/

[5] *Web de tecnologías web* http://www.w3schools.com/

[6] *Web de Bootstrap* http://getbootstrap.com/

[7] *Web de JUnit* http://junit.org/

[8] *Web de Spock* https://code.google.com/p/spock/

[9] *Web MySQL* https://www.mysql.com/

[10] *Web de Hibernate* http://hibernate.org/

[11] *Web de Groovy/Grails Tool Suite* https://grails.org/products/ggts

[12] *Web de Selenium* http://www.seleniumhq.org/

[13] *Web de Tomcat* http://tomcat.apache.org/

[14] *Web de MySQL Workbench* https://www.mysql.com/products/workbench/

[15] *Web de Subversion* https://subversion.apache.org/

[16] *Web de Tortoise* http://tortoisesvn.net/

[17] *Web de Dia* http://dia-installer.de/

[18] *Web de Microsoft Office* https://products.office.com/es-es/home

[19] *Web de Putty* http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html

[20] *Web de WinSCP* http://winscp.net/

[21] *Web de JXPlorer* http://jxplorer.org/

[22] *Web del plugin Database Migration* http://grails.org/plugin/database-migration

[23] *Web del plugin Geb Integration* https://grails.org/plugin/geb

[24] *Web del plugin Quartz Plugin* http://grails.org/plugin/quartz

[25] *Web del plugin Mail Support* http://grails.org/plugin/mail

[26] *Web del plugin Spring Security Core* http://grails.org/plugin/spring-security-core

[27] *Web del plugin Spring Security LDAP* http://grails.org/plugin/spring-security-ldap

[28] *Web del plugin Grails CSV* http://grails.org/plugin/csv

[29] *Laurie Williams (2006). "Testing Overview and Black-Box Testing Techniques" (Visitado: Octubre 2014)* http://agile.csc.ncsu.edu/SEMaterials/BlackBox.pdf

[30] *Martin Flowers (2007). "Mocks aren't Stubs" (Visitado: Noviembre 2014)* http://martinfowler.com/articles/mocksArentStubs.html

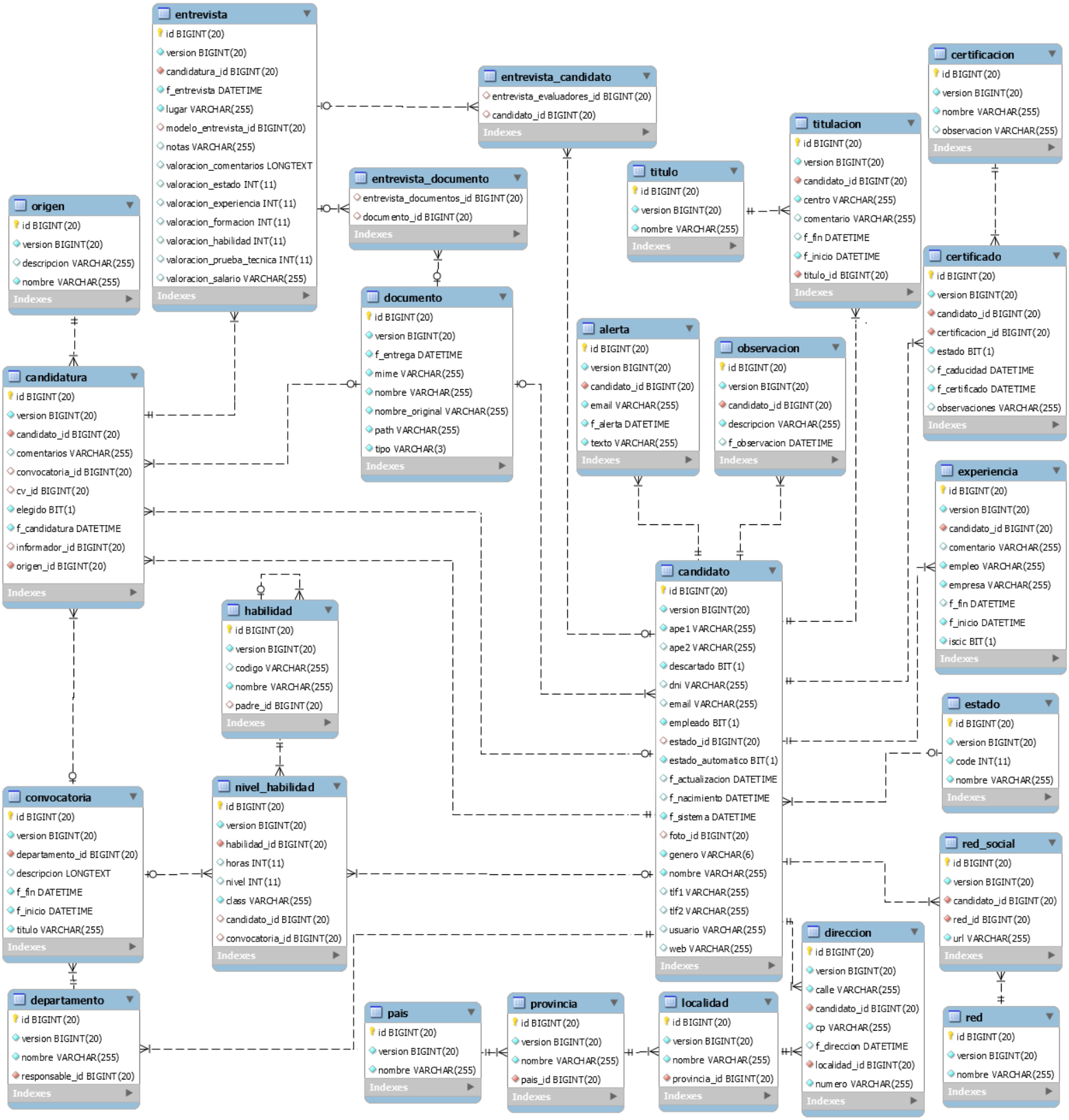
[31] *Principio de Pareto (Visitado: Octubre 2014)* http://davidtopi.com/el-principio-de-pareto-la-regla-del-8020/#.VH7ceTGG_fc

[32] *Gonzalo Méndez (2008). "Especificación de requisitos según el estándar IEEE Std 830-1998" (Visitado: Diciembre 2014)* https://www.fdi.ucm.es/profesor/gmendez/docs/is0809/ieee830.pdf

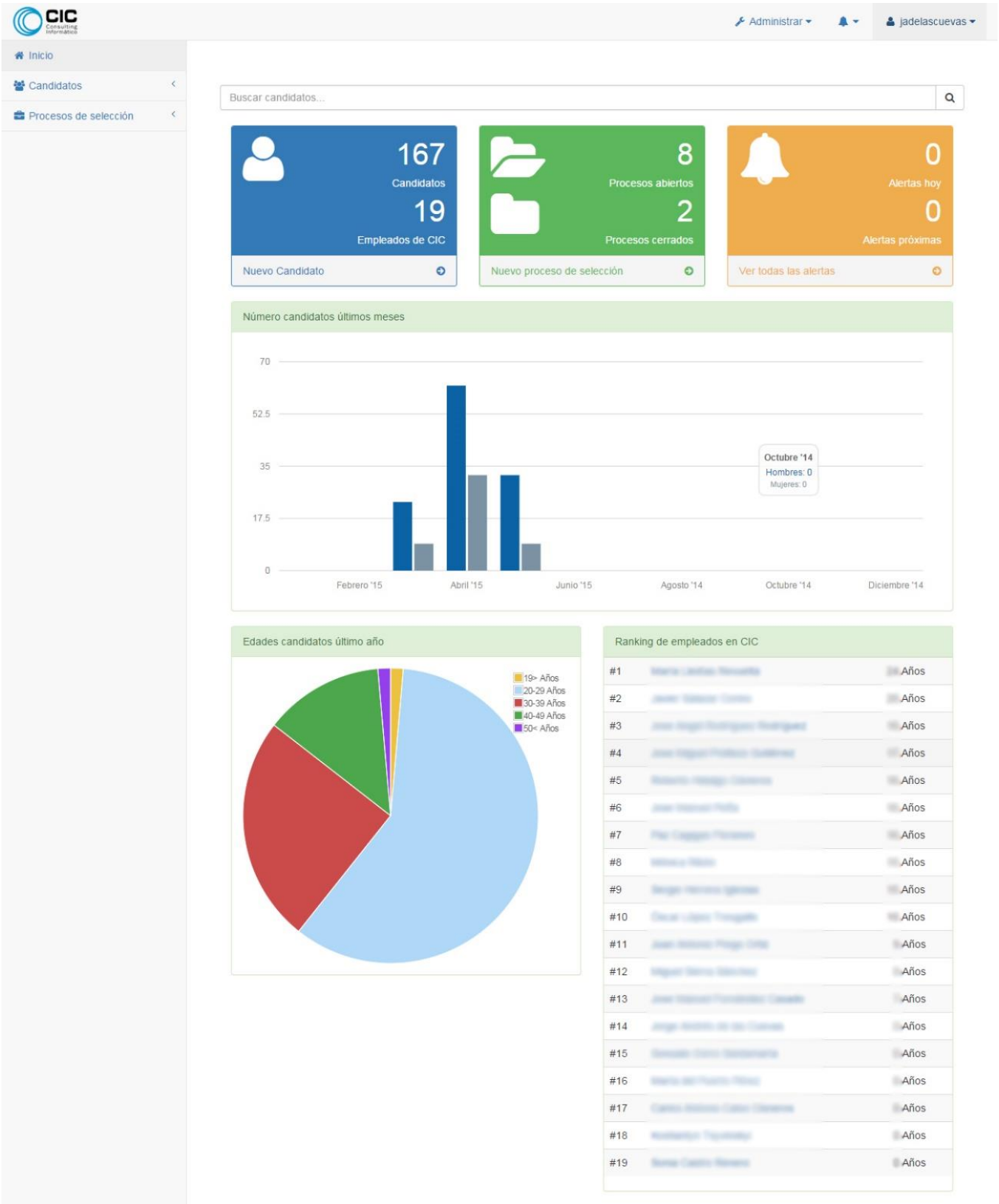
[33] *Theme SB Admin 2 (Bootstrap)* http://startbootstrap.com/template-overviews/sb-admin-2/

[34] *Glen Smith and Peter Ledbrook (2014). "Grails in Action"*

ANEXO I


entrevista
id BIGINT(20)
version BIGINT(20)
candidatura_id BIGINT(20)
f_entrevista DATETIME
lugar VARCHAR(255)
modelo_entrevista_id BIGINT(20)
notas VARCHAR(255)
valoracion_comentarios LONGTEXT
valoracion_estado INT(11)
valoracion_experiencia INT(11)
valoracion_formacion INT(11)
valoracion_habilidad INT(11)
valoracion_prueba_tecnica INT(11)
valoracion_salario VARCHAR(255)
Indexes
entrevista_candidato
entrevista_evaluadores_id BIGINT(20)
candidato_id BIGINT(20)
Indexes
certificacion
id BIGINT(20)
version BIGINT(20)
nombre VARCHAR(255)
observacion VARCHAR(255)
Indexes
entrevista_documento
entrevista_documentos_id BIGINT(20)
documento_id BIGINT(20)
Indexes
titulo
id BIGINT(20)
version BIGINT(20)
nombre VARCHAR(255)
Indexes
titulacion
id BIGINT(20)
version BIGINT(20)
candidato_id BIGINT(20)
centro VARCHAR(255)
comentario VARCHAR(255)
f_fin DATETIME
f_inicio DATETIME
titulo_id BIGINT(20)
Indexes
origen
id BIGINT(20)
version BIGINT(20)
descripcion VARCHAR(255)
nombre VARCHAR(255)
Indexes
documento
id BIGINT(20)
version BIGINT(20)
f_entrega DATETIME
mime VARCHAR(255)
nombre VARCHAR(255)
nombre_original VARCHAR(255)
path VARCHAR(255)
tipo VARCHAR(3)
Indexes
certificado
id BIGINT(20)
version BIGINT(20)
candidato_id BIGINT(20)
certificacion_id BIGINT(20)
estado BIT(1)
f_caducidad DATETIME
f_certificado DATETIME
observaciones VARCHAR(255)
Indexes
alerta
id BIGINT(20)
version BIGINT(20)
candidato_id BIGINT(20)
email VARCHAR(255)
f_alerta DATETIME
texto VARCHAR(255)
Indexes
observacion
id BIGINT(20)
version BIGINT(20)
candidato_id BIGINT(20)
descripcion VARCHAR(255)
f_observacion DATETIME
Indexes
candidatura
id BIGINT(20)
version BIGINT(20)
candidato_id BIGINT(20)
comentarios VARCHAR(255)
convocatoria_id BIGINT(20)
cv_id BIGINT(20)
elegido BIT(1)
f_candidatura DATETIME
informador_id BIGINT(20)
origen_id BIGINT(20)
Indexes
experiencia
id BIGINT(20)
version BIGINT(20)
candidato_id BIGINT(20)
comentario VARCHAR(255)
empleo VARCHAR(255)
empresa VARCHAR(255)
f_fin DATETIME
f_inicio DATETIME
iscic BIT(1)
Indexes
habilidad
id BIGINT(20)
version BIGINT(20)
codigo VARCHAR(255)
nombre VARCHAR(255)
padre_id BIGINT(20)
Indexes
candidato
id BIGINT(20)
version BIGINT(20)
ape1 VARCHAR(255)
ape2 VARCHAR(255)
descartado BIT(1)
dni VARCHAR(255)
email VARCHAR(255)
empleado BIT(1)
estado_id BIGINT(20)
estado_automatico BIT(1)
f_actualizacion DATETIME
f_nacimiento DATETIME
f_sistema DATETIME
foto_id BIGINT(20)
genero VARCHAR(6)
nombre VARCHAR(255)
tlf1 VARCHAR(255)
tlf2 VARCHAR(255)
usuario VARCHAR(255)
web VARCHAR(255)
Indexes
estado
id BIGINT(20)
version BIGINT(20)
code INT(11)
nombre VARCHAR(255)
Indexes
convocatoria
id BIGINT(20)
version BIGINT(20)
departamento_id BIGINT(20)
descripcion LONGTEXT
f_fin DATETIME
f_inicio DATETIME
titulo VARCHAR(255)
Indexes
nivel_habilidad
id BIGINT(20)
version BIGINT(20)
habilidad_id BIGINT(20)
horas INT(11)
nivel INT(11)
class VARCHAR(255)
candidato_id BIGINT(20)
convocatoria_id BIGINT(20)
Indexes
red_social
id BIGINT(20)
version BIGINT(20)
candidato_id BIGINT(20)
red_id BIGINT(20)
url VARCHAR(255)
Indexes
direccion
id BIGINT(20)
version BIGINT(20)
calle VARCHAR(255)
candidato_id BIGINT(20)
cp VARCHAR(255)
f_direccion DATETIME
localidad_id BIGINT(20)
numero VARCHAR(255)
Indexes
departamento
id BIGINT(20)
version BIGINT(20)
nombre VARCHAR(255)
responsable_id BIGINT(20)
Indexes
pais
id BIGINT(20)
version BIGINT(20)
nombre VARCHAR(255)
Indexes
provincia
id BIGINT(20)
version BIGINT(20)
nombre VARCHAR(255)
pais_id BIGINT(20)
Indexes
localidad
id BIGINT(20)
version BIGINT(20)
nombre VARCHAR(255)
provincia_id BIGINT(20)
Indexes
red
id BIGINT(20)
version BIGINT(20)
nombre VARCHAR(255)
Indexes

ANEXO II


CIC
Consulting Informático
Administrar
jadelascuevas
Inicio
Candidatos
Procesos de selección
Buscar candidatos...
167
Candidatos
19
Empleados de CIC
Nuevo Candidato
8
Procesos abiertos
2
Procesos cerrados
Nuevo proceso de selección
0
Alertas hoy
0
Alertas próximas
Ver todas las alertas
Número candidatos últimos meses
70
52.5
35
17.5
0
Octubre '14
Hombres: 0
Mujeres: 0
Febrero '15
Abril '15
Junio '15
Agosto '14
Octubre '14
Diciembre '14
Edades candidatos último año
19> Años
20-29 Años
30-39 Años
40-49 Años
50< Años
Ranking de empleados en CIC
#1
#2
#3
#4
#5
#6
#7
#8
#9
#10
#11
#12
#13
#14
#15
#16
#17
#18
#19
Años