Contenido

[Tablas 2](#_Toc481608621)

[1. Introducción: 3](#_Toc481608622)

[2. Diseño de pruebas: 4](#_Toc481608623)

[2.1. Pruebas unitarias y Pruebas funcionales: 4](#_Toc481608624)

[2.2. Diseño de pruebas: Pruebas repositorio: 5](#_Toc481608625)

[2.2.1. Diseño de pruebas unitaria: 5](#_Toc481608626)

[2.2.2. Diseño de pruebas funcionales: 5](#_Toc481608627)

[2.3. Diseño de pruebas: Pruebas de la aplicación web 5](#_Toc481608628)

[2.3.1. Diseño de pruebas funcionales: 5](#_Toc481608629)

[2.3.2. Diseño de pruebas unitarias: 6](#_Toc481608630)

[7](#_Toc481608631)

[2.3.3. Diseño de pruebas de navegabilidad: 7](#_Toc481608632)

[3. Carga de los datos: 8](#_Toc481608633)

[3.1. Carga de los datos: Valores 8](#_Toc481608634)

[3.1.1. UserLogin: 8](#_Toc481608635)

[3.1.2. Grupos: 10](#_Toc481608636)

[4. Ejecución de pruebas: 11](#_Toc481608637)

[4.1. Resumen de las pruebas: 11](#_Toc481608638)

[4.2. Errores de la aplicación de ejemplo: 11](#_Toc481608639)

# Tablas

[Tabla 3‑1 UserLogin 1 7](#_Toc481608599)

[Tabla 3‑2 UserLogin 2 8](#_Toc481608600)

[Tabla 3‑3 UserLogin 3 8](#_Toc481608601)

[Tabla 4‑1 Resumen de pruebas 10](#_Toc481608602)

[Tabla 4‑2 Ejemplo prueba: 1 11](#_Toc481608603)

[Tabla 4‑3 Ejemplo prueba: 2 11](#_Toc481608604)

[Tabla 4‑4 Ejemplo prueba: 3 11](#_Toc481608605)

[Tabla 4‑5 Ejemplo prueba: 4 12](#_Toc481608606)

[Tabla 4‑6 Ejemplo prueba: 5 12](#_Toc481608607)

[Tabla 4‑7 Ejemplo prueba: 6 13](#_Toc481608608)

# Introducción:

Este documento denominado “Pruebas del sistema. Es un integrante de un conjunto de documentos cuya finalidad es el definir y diseñar una “Aplicación web que gestione un repositorio”.

Para poder describir mejor el problema deberemos desarrollar el significado de lo que se va a definir, es decir, “Aplicación web que gestione un repositorio”. Un repositorio es según la Wikipedia: *Un* ***repositorio de software*** *es un lugar de almacenamiento del cual pueden ser recuperados e instalados los paquetes de software en un ordenador.*

Nuestro repositorio mantendrá un lugar de almacenamiento dentro del ordenador que lo instale, con el fin de contener diversos proyectos que albergarán a su vez, programas de diversos lenguajes de programación que podrán ser ejecutados. Nuestro repositorio devolverá no el programa almacenado, sino, la información de interactuar con él mediante unas entradas planificadas por el usuario.

Nuestra aplicación web, gestionara la comunicación con nuestro repositorio. Pidiéndole información y mostrándosela al usuario de una manera sencilla para su entendimiento. Se insta que si se quiere saber más sobre el objetivo de la aplicación se lea el documento general de la memoria.

El objetivo de este documento es el realizar una memoria sobre las pruebas**.** El cual su función es definir todo el conjunto de actividades como diseño, realización de las pruebas, solución de las pruebas.

# Diseño de pruebas:

Para el estudio y diseño de las pruebas de la aplicación del repositorio se ha tenido en cuenta, la cobertura máxima de las pruebas. Esta aplicación es muy extensa y si quisiéramos generar una cobertura de la mayoría del código, sería un trabajo igual o superior al desarrollo del mismo.

Hay que entender que existe dos capas en la aplicación con dos funcionalidades diferente.

* **El repositorio**: Esta capa está orientada en gestionar el negocio de la aplicación. Las pruebas que realizaremos estarán orientadas a comprobar el código. Para esta capa realizaremos una batería de pruebas unitarias dentro del código y para la comunicación pruebas funcionales.
* **La aplicación**: La aplicación web está destinada a la comunicación con el usuario, su objetivo es más hacer cosas, que saber cómo hacerlas. Como el objetivo es el usuario, realizaremos pruebas funcionales utilizando los casos de uso del análisis del sistema.

## Pruebas unitarias y Pruebas funcionales:

* **Prueba unitaria:** es una forma de comprobar el correcto funcionamiento de una unidad de código. Por ejemplo, en diseño estructurado o en diseño funcional una función o un procedimiento, en diseño orientado a objetos una clase. Esto sirve para asegurar que cada unidad funcione correctamente y eficientemente por separado.
* **Prueba funcional:** Prueba basada en la ejecución, revisión y retroalimentación de las funcionalidades previamente diseñadas para el software. Las pruebas funcionales se hacen mediante el diseño de modelos de prueba que buscan evaluar cada una de las opciones con las que cuenta el paquete informático. Dicho de otro modo, son pruebas específicas, concretas y exhaustivas para probar y validar que el software hace lo que debe y, sobre todo lo que se ha especificado.

## Diseño de pruebas: Pruebas repositorio:

### Diseño de pruebas unitaria:

Para comprobar el correcto manejo de las órdenes del usuario y la gestión de los diversos grupos, ejecuciones o proyecto. Se va a realizar varios baterías de pruebas unitarias. Dichas pruebas se han creado para todos los modelos de la aplicación sin excepción. Comprobando cada uno de forma independiente:

* La posibilidad de acceder a la base de datos, crear datos, modificar datos o acceder a ellos para devolverlos.
* La comprobación de las salidas y entradas de las diversas funciones del negocio, analizando la salida de dichas funciones de forma errónea o correcta.
* El correcto funcionamiento de las clases y funciones auxiliares que facilitan el funcionamiento del negocio.

Se ha puesto atención en hacer comprobaciones positivas y negativas. Es decir, comprobar el buen funcionamiento cuando es esperado y los errores cuando las operaciones de los diversos, controladores, funciones y clases devuelven error.

### Diseño de pruebas funcionales:

El objetivo de las pruebas funcionales es la interacción de un usuario con la aplicación. Esto permite comprobar el funcionamiento de los diversos conjuntos de sistemas que realizan actividades determinadas.

Estas pruebas caben destacar que no se adentran en el código, analizan el código como una caja negra. Si existen fallos en las pruebas funcionales, deberá existir en las unitarias. Y si no existe fallo en las unitarias es que las coberturas unitarias no es el correcto.

La iteración entre el usuario y el sistema en el repositorio es mediante el uso de controladores que gestionan la arquitectura REST. Es por eso que enfocaremos las pruebas funcionales en ellos, los controladores REST.

## Diseño de pruebas: Pruebas de la aplicación web

### Diseño de pruebas funcionales:

El objetivo de las pruebas funcionales es la interacción de un usuario con la aplicación. Esto permite comprobar el funcionamiento de los diversos conjuntos de sistemas que realizan actividades determinadas entre sí para resolver los requisitos de sistema.

Estas pruebas caben destacar que no se adentran en el código, analizan el código como una caja negra. Si existen fallos en las pruebas funcionales, deberá existir en las unitarias. Y si no existe fallo en las unitarias es que las coberturas unitarias no es el correcto.

Cabe destacar que la iteración entre usuario y sistema en la aplicación web, lleva por medio una manipulación de la vista por parte del modelo. Obviaremos esto y dejaremos la corrección de la salida en la vista, del objetivismo del tester.

Para el diseño de las pruebas se ha seguido un modelo en forma de “Hoja de cebolla” se ha ido implementando las funcionalidades más básicas de cada capa. Han sido probados sus comportamientos, como se esperaban. Después se ha integrado con su capa superior y se ha realizado las pruebas de las capas superiores.

En caso de haber habido algún fallo relacionado con la capa inferior, se ha bajado a la capa inferior a modificar el fallo. Tras eso se ha realizado pruebas de integración de nuevo. Todo esto se ha realizado hasta llegar a las capas que pueden demostrar los servicios que cumplen los casos de uso donde se han utilizado pruebas funcionales de validación.

Las pruebas de validación son pruebas de caja negra, como se ha explicado anteriormente. Donde se piensa que un código debe aportar una salida esperada para ser correcto. En caso contrario existe un fallo en su implementación. Por eso debe ser reparado y vuelto a probar.

### Diseño de pruebas unitarias:

Para comprobar el correcto manejo de las ordenes de los usuarios y de la gestión de los diversos grupos, ejecuciones o proyectos. Se va a realizar varias baterías de pruebas unitarias. Dichas pruebas se han creado para todos los modelos de la aplicación sin excepción. Comprobando cada uno de forma independiente:

* Comunicación con el repositorio web. Interpretación de los mensajes, realización de órdenes y manipulación de los datos recibidos o enviados.
* Validación de los datos para ser entregados al usuario o al sistema del repositorio.
* Creación de vistas complejas y registro de eventos del usuario.

Estas pruebas caben destacar que se adentran en el código, analizan el código como una caja blanca. Si existen fallos en las pruebas funcionales, deberá existir en las unitarias. Y si no existe fallo en las unitarias es que las coberturas unitarias no es la correcta.

Para realizar las pruebas unitarias en Spring, se ha usado un framework habitual en las tecnologías Java llamado JUnit. Para angularJS se ha utilizado dos programas unidos para dar más cobertura denominados Jasmine y Karma.

### 

### Diseño de pruebas de navegabilidad:

El objetivo de las pruebas de navegabilidad es el comprobar todas las navegaciones posibles de la aplicación web. Esto sirve para descubrir posibles fallos de los modelos de la aplicación web, después de analizar un evento realizado por el usuario.

Realizaremos una batería de pruebas simples funcionales siguiendo el esquema del mapa de navegabilidad descrito en el documento diseño del sistema. Nos enfocaremos en el resultado de la navegación en vez de en el contenido de las vistas.

# Carga de los datos:

Con el objetivo de tener unos datos válidos para realizar las pruebas mencionadas en el diseño de las pruebas, se ha creado un conjunto predefinido que permita ejecutar y comprobar las pruebas en un entorno controlado. La carga de datos contiene:

* Tres usuarios enlazados con sus correspondientes entidades personales. Uno de ellos será administrador, los otros dos serán usuarios.
* Un grupo principal alojado en el espacio público, con un subgrupo a su cargo.
* Proyectos ejecutables en cada uno de los grupos, los proyectos pueden ser de diferentes tipos de proyectos.

A continuación, se describirán los valores exactos de cada elemento, estos deberían ser inyectados directamente mediante el lenguaje de interacción de la base de datos denominado Cypher. O también realizando una aplicación que utilice la capa de negocio DAO del repositorio.

## Carga de los datos: Valores

### UserLogin:

Tabla ‑ UserLogin 1

|  |  |  |
| --- | --- | --- |
| **Entidad** | **UserLogin** | |
| **Variable** | Tipo | Valor |
| **User** | String (Email) | [admin@uniapi.com](mailto:admin@uniapi.com) |
| **Pass** | String (SHA-1) | “contraseña” SHA-1. |
| **Rol** | Enum (UserType:  {USER,ADMIN}) | ADMIN |
| **creationTime** | Date | TODAY |
| **Hashcode** | String (SHA-1) | this.toString() SHA-1 |

Tabla ‑ UserLogin 2

|  |  |  |
| --- | --- | --- |
| **Entidad** | **UserLogin** | |
| **Variable** | Tipo | Valor |
| **User** | String (Email) | [user1@uniapi.com](mailto:user1@uniapi.com) |
| **Pass** | String (SHA-1) | “contraseña” SHA-1. |
| **Rol** | Enum (UserType:  {USER,ADMIN}) | USER |
| **creationTime** | Date | TODAY |
| **Hashcode** | String (SHA-1) | this.toString() SHA-1 |

Tabla ‑ UserLogin 3

|  |  |  |
| --- | --- | --- |
| **Entidad** | **UserLogin** | |
| **Variable** | Tipo | Valor |
| **User** | String (Email) | [user2@uniapi.com](mailto:user2@uniapi.com) |
| **Pass** | String (SHA-1) | “contraseña” SHA-1. |
| **Rol** | Enum (UserType:  {USER,ADMIN}) | USER |
| **creationTime** | Date | TODAY |
| **Hashcode** | String (SHA-1) | this.hash() SHA-1 |

Todos los usuarios llevaran el mismo tipo de persona, ya que no es necesario el que los datos personales del usuario sean distintos.

|  |  |  |
| --- | --- | --- |
| **Entidad** | **Person** | |
| **Variable** | Tipo | Descripción |
| **Name** | String | Persona |
| **Subname** | String | Apellido1 Apellido2 |
| **Birthday** | Date | NOW |
| **Birthplace** | String | Ciudad |
| **Province** | String | Provincia |
| **Country** | String | Pais |
| **Biografy** | String | Pequeño texto para que la persona se pueda expresar. |
| **profileImageUrl** | String(URL) | https://www.mastermagazine.info/termino/wp-content/uploads/Usuario-Icono.jpg |
| **dateCreation** | Date | NOW |
| **hashcode** | String (SHA-1) | this.hash() |

### 3.1.2. Grupos:

Este grupo debe ser creado por el usuario admin. Y debe contener como miembro a los dos usuarios.

|  |  |  |
| --- | --- | --- |
| **Entidad** | **Group** | |
| **Variable** | Tipo | Descripción |
| **Name** | String | Algoritmia |
| **Type** | Enum (GroupType:{  PUBLIC,  PRIVATE,MAIN}) | PUBLIC |
| **Description** | String | Sección de algoritmia |
| **CreationDate** | Date | NOW |
| **SharingGroupPermissions** | String[] | [YES,YES,YES,YES] |
| **GroupCreationPermissions** | String[] | [YES,YES,YES,YES] |
| **MemberGestionPermissions** | String[] | [YES,YES,YES,YES] |
| **ProjectPropertiesPermissions** | String[] | [YES,YES,YES,YES] |
| **Hashcode** | String(SHA-1) | this.hash() SHA-1 |

Este grupo debe ser subgrupo del grupo “departamento de informática” y ha de ser creado por el user2. No deberá contener ningún miembro

# Ejecución de pruebas:

A continuación, se detallan los resultados de la ejecución de las pruebas, así como algunos errores obtenidos durante el desarrollo, lo cual justifica la necesidad de tener y hacer uso de pruebas detalladas.

A la hora de realizar el presente documento se tiene el desarrollo de la aplicación finalizado y el total de las pruebas devuelve un resultado positivo. Por lo tanto, los modelos, controladores y vistas superan todas las pruebas preparadas para comprobar su funcionamiento.

## **Resumen de las pruebas:**

Tras el diseño de las pruebas automáticas y las pruebas funcionales siguiendo los casos de uso del análisis del sistema, reunimos todas las pruebas realizadas por el desarrollador. En un resumen, cabe destacar que la mayor parte de las pruebas han sido realizadas de forma ágil (en contra posición a una forma monolítica). Es decir, al mismo tiempo que se realizaba las funciones.

Tabla ‑ Resumen de pruebas

|  |  |  |
| --- | --- | --- |
| Tipo | Pruebas | Aserciones |
| Pruebas unitarias | 93 | 564 |
| Pruebas funcionales | 74 | 1193 |
| Total | 167 | 1757 |

Ciertos aspectos de la aplicación, como por ejemplo la ejecución de trabajos de frontend, son difíciles de comprobar con frameworks de pruebas automatizadas, por lo que requieren una comprobación directa del programador. De ahí que este tipo de pruebas no se hayan incluido en el cómputo de la Tabla Resumen de pruebas.

## Errores de la aplicación de ejemplo:

Durante el desarrollo de la aplicación se han dado situaciones en las que las pruebas avisaban de que el comportamiento de algún aspecto de la aplicación había dejado de ser el correcto. Razón suficiente para demostrar la importancia de las pruebas en el desarrollo de software, pues en caso de que los errores no fueran comprobados podrían llegar a la versión estable disponible al público.

A continuación, se expone la relación de algunas pruebas que fallaron durante el desarrollo:

Tabla ‑ Ejemplo prueba: 1

|  |  |
| --- | --- |
| Prueba | Búsqueda de userLogin por hashcode |
| Descripción de la prueba | Se solicita al DAO el usuario existente con su Hashcode |
| Salida esperada | La entidad UserLogin relacionada con el hashcode |
| Salida encontrada | NULL |
| Posible error encontrado | Fallo en la generación del hashcode en la función userLogin.hash() |

Tabla ‑ Ejemplo prueba: 2

|  |  |
| --- | --- |
| Prueba | Modificación de url de repositorio |
| Descripción de la prueba | Se modifica la url del repositorio del proyecto. |
| Salida esperada | Correcto, en la jerarquía de proyectos una modificación del proyecto. |
| Salida encontrada | Ninguna respuesta |
| Posible error encontrado | No se ha implementado la función |

Tabla ‑ Ejemplo prueba: 3

|  |  |
| --- | --- |
| Prueba | Sincronización de proyectos con el repositorio |
| Descripción de la prueba | Se realiza la comunicación GIT con el repositorio para obtener los datos del proyecto |
| Salida esperada | Correcto, en la jerarquía de proyectos una modificación del proyecto. |
| Salida encontrada | Correcto, pero en la jerarquía no se obtenía nada |
| Posible error encontrado | Se supone que no se espera a que el servicio acabe para finalizar la comunicación |

Tabla ‑ Ejemplo prueba: 4

|  |  |
| --- | --- |
| Prueba | Seleccionar e añadir elementos de input en la entrada por defecto |
| Descripción de la prueba | En el input de selección de elementos de entrada por defecto, debería aplicarse un cierto comportamiento. |
| Salida esperada | Ante el botón (+) se añade un elemento a la lista del input. Ante un botón (-) se elimina. |
| Salida encontrada | Añade elementos pero no los elimina |
| Posible error encontrado | Fallo en la programación del elemento |

Tabla ‑ Ejemplo prueba: 5

|  |  |
| --- | --- |
| Prueba | Listado de ejecuciones en ventana ejecuciones |
| Descripción de la prueba | Por solicitud del usuario se solicitan a la API un listado de ejecuciones y se representan en la vista |
| Salida esperada | Vista divida en varios listados según su estado de ejecución: Finalizado, en ejecución… |
| Salida encontrada | Las ejecuciones se repiten en las vistas |
| Posible error encontrado | Fallo del uso del framework de relleno del modelo |

Tabla ‑ Ejemplo prueba: 6

|  |  |
| --- | --- |
| Prueba | Modificación de los permisos de un grupo |
| Descripción de la prueba | El usuario solicita a la API que se le modifiquen los permisos a un grupo |
| Salida esperada | Los permisos del grupo y de todos sus hijos han sido modificados |
| Salida encontrada | Los permisos del padre han sido modificados. Los de los hijos no |
| Posible error encontrado | Fallo en la utilización de la query de modificación en profundidad de NEO4J |