1. **Introducción:**

Este documento denominado “Manual del usuario”, es un documento 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 programas de diversos lenguajes que podrán ser ejecutados. Nuestro repositorio devolverá no es programa almacenado, sino, la información de interactuar con él.

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 un manual del usuario, para que todos los lectores e revisores de este articulo puedan utilizar el software a desarrollar. Para ello se listará un conjunto de instrucciones y descripciones del sistema que creemos que serán de utilidad para poder utilizar el sistema.

1. **Obtención del código:**

Para poder obtener los códigos de los diversos programas usuraremos la plataforma de repositorios GIT conocida como **Github**. Github provee un repositorio permanente para nuestro programa con un seguidor de modificaciones muy notorio que facilitara a cualquier persona el descargarse versiones modificadas o mejoradas.

<https://github.com/RaulGF92/UniApi>

La dirección previamente mostrada se referencia al espacio donde el repositorio se alberga, ahí se podrá ver todas las modificaciones, paso por paso, que el desarrollador a usado. Cualquier persona con conocimientos básicos sobre GIT puede descargárselo. No obstante, mostraremos las dos formas de realizarlo:

* Descargando a través de la página web: En este caso el usuario deberá introducirse en la página indicada por el documento. Se le mostrara una página web como la indicada en la figura de a continuación. Ahí deberá pulsar en el botón “*Clone or download*”.

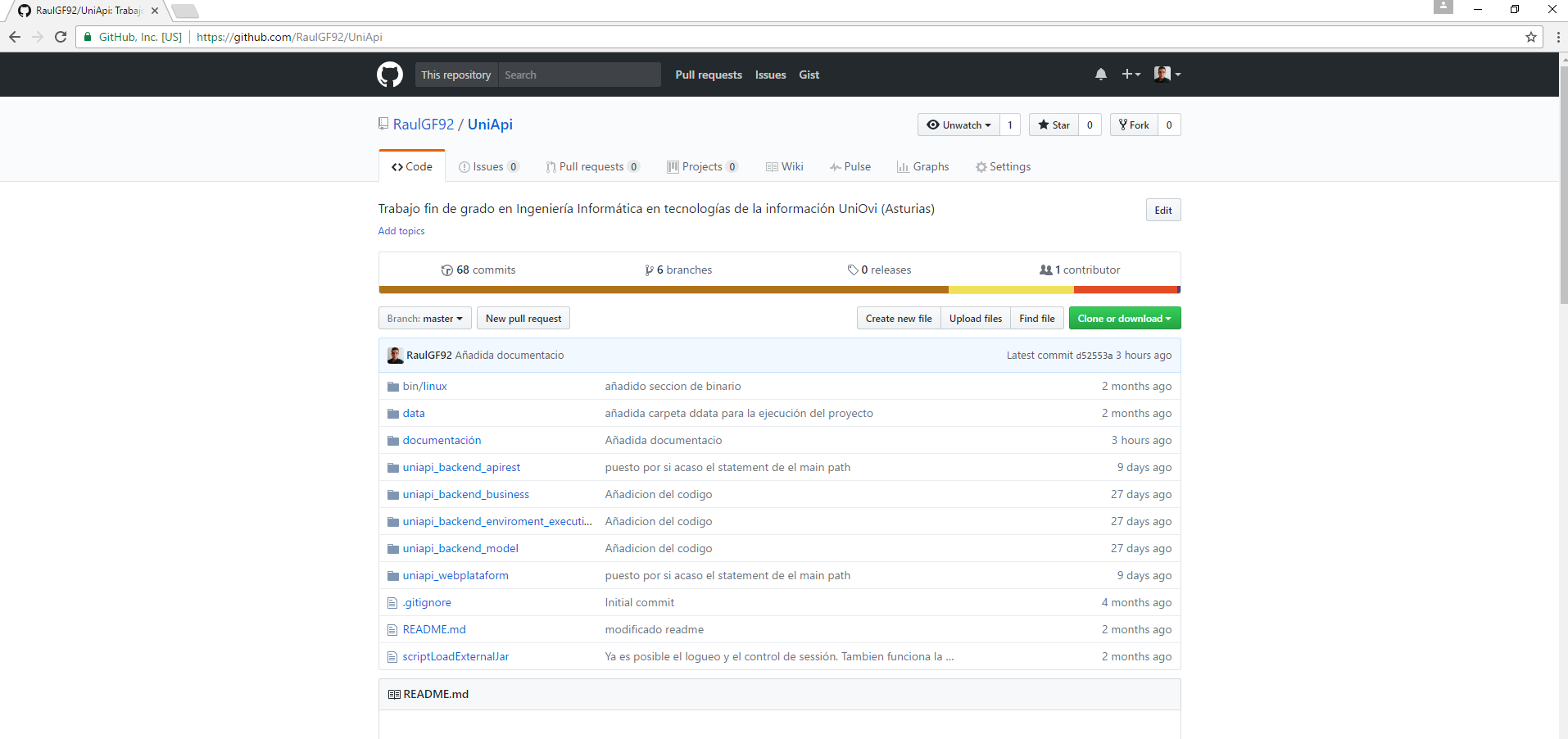


Ilustración . Página del repositorio

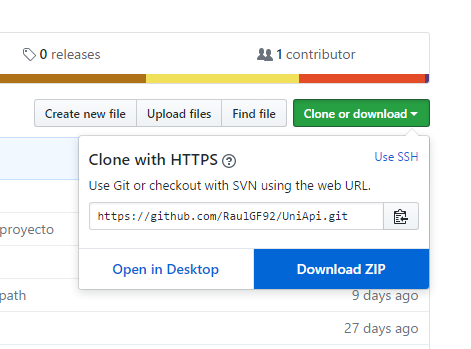


Ilustración . Botón de descarga GIT

Tras pulsar el botón de download Zip. Se descargará un archivo Zip donde se contendrá el proyecto UniApi. Es necesario descomprimirlo y depositarlo en el sistema de archivos del servidor que albergara el servicio.

* Mediante consola de comandos GIT: Para ello debemos conocer la dinámica del protocolo GIT y sus comportamientos. Para ello realizaremos un clonado del repositorio usando la URL que se nos indica posteriormente. Tras eso el protocolo se ocupará automáticamente de la descarga.

1. **Instalación del repositorio:**

Para instalar el repositorio necesitaremos los datos del repositorio además de un IDE denominado Eclipse. Usaremos este IDE por que gestiona Maven un gestor de paquetes Java de forma transparente para el desarrollador. Aunque es posible el uso de maven usando el prompt del sistema operativo.

* 1. **Generaciones ejecutables:**

En esta parte realizaremos un ejecutable que se utilizara para ejecutar el repositorio a partir de todos los datos de la jerarquía uniapi, descargada o clonada por el usuario en el repositorio indicado a inicio de este documento.

* + 1. **Eclipse: Conversión Maven**

Para empezar con la conversión inicializamos el eclipse. Nos preguntara donde albergar el workspace. Es recomendable indicarle que debe de ser la jerarquía uniapi donde se alberga los datos. El eclipse se empezará a cargar y nos proporcionará un espacio de trabajo vacío.

Entramos en las opciones File y elegimos la opción importar. Ahí nos desplega un sinfín de opciones elegimos la opción “**Existing projects in the workspace**”.

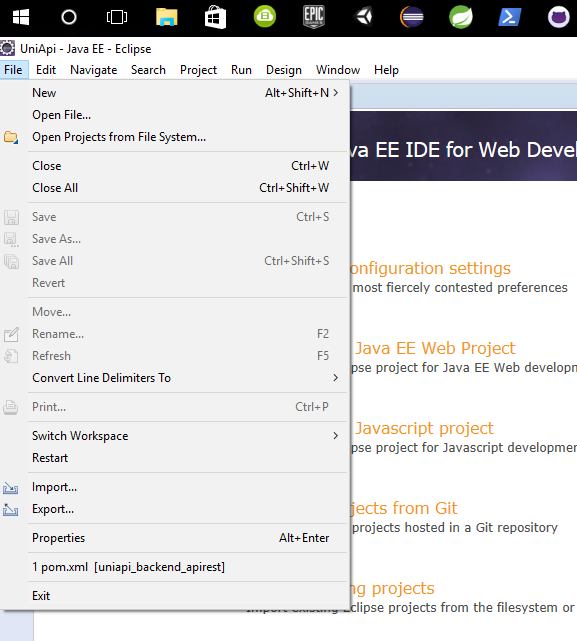


Ilustración . Seleccion import eclipse

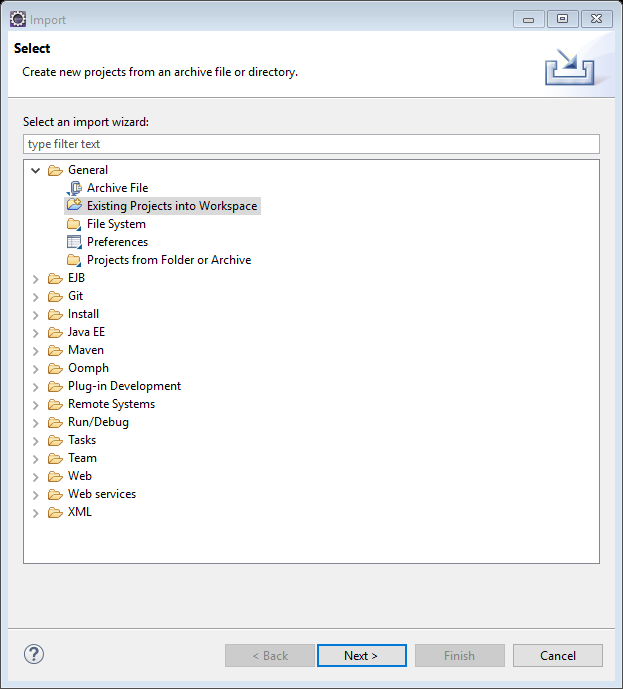


Ilustración . Multiples opciones de import

Tras seleccionar la opción se nos mostrara otra ventana donde deberemos seleccionar el check box “**root directive**”. Siguiendo una ventana que nos ayuda a navegar por la jerarquía seleccionamos la carpeta uniapi. Se nos indicara los tres proyectos de uniapi, seleccionaremos los tres y por ultimo daremos a finish.

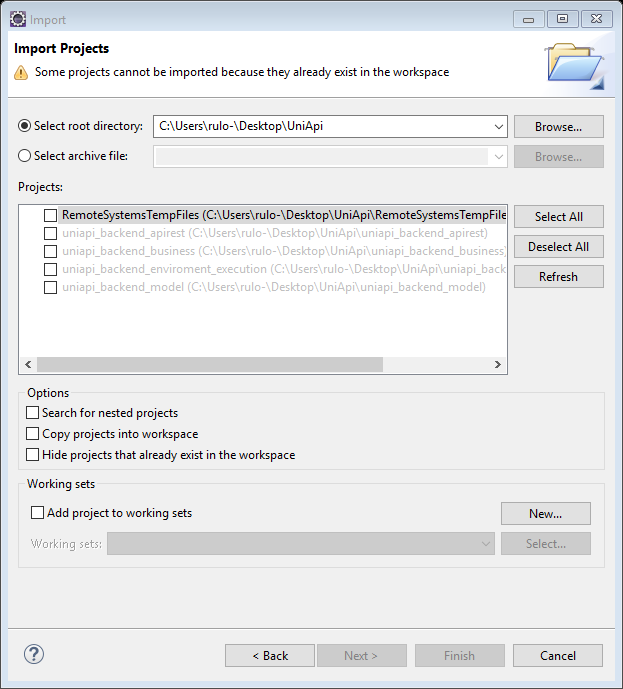


Ilustración . Import projects final

Tras la importación tendremos cuatro proyectos java. Para terminar la conversión deberemos ir uno por uno pulsando el botón derecho. Llendo a la pestaña configure y por ultimo pulsando convert in maven Project.

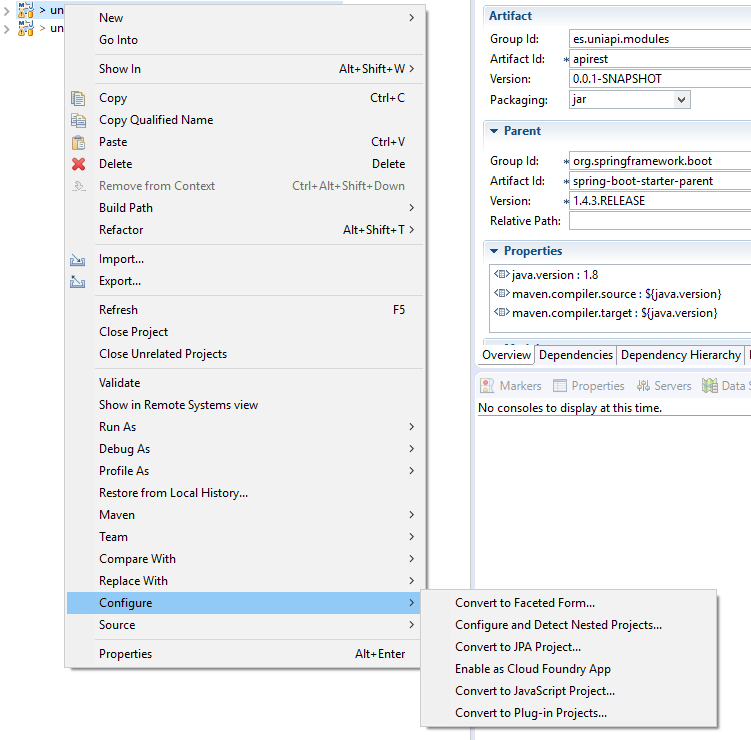


Ilustración . Convert project maven

Con esto todos los proyectos de nuestro workspace serán maven. Por ultimo selecciona cada proyecto de nuevo. Pulsa botón derecho, clicka la opción maven y clicka update. Esto actualizara los proyectos. Esta acción generara muchos errores, pero luego serán arreglados.

* + 1. **Eclipse: Incluir e Ejecutar**

Tras contener todos los proyectos maven instalados e actualizados en el workspace vamos a compilarlos y ejecutarlos. La aplicación del repositorio es una aplicación en pila, es decir, las capas superiores utilizan datos de la capa inferior. Es necesario compilar los datos para que las capas puedan utilizarlos. Para realizar esta operación iremos de la capa más inferior a la más superior construyendo sus proyectos, compilándolos y entregándolos a las capas superiores.

1. **Modelo de datos**: nos colocamos en el proyecto “**uniapi\_backend\_model**” y lo seleccionamos, pulsamos el botón derecho y nos dirigimos a run, donde pincharemos maven build…

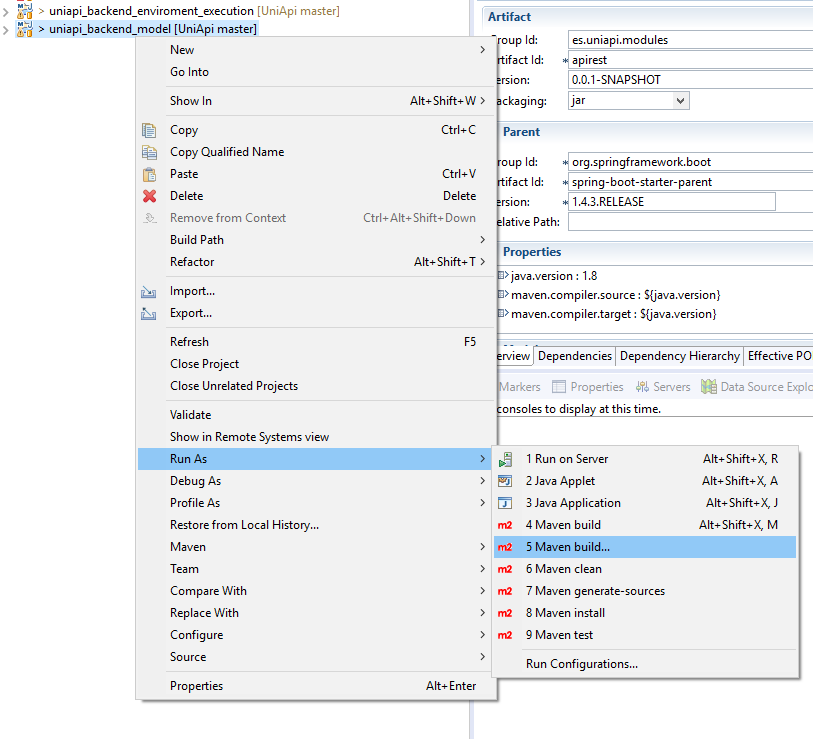
****

Ilustración . Maven run as...

Tras pinchar en la opción nos aparecerá una ventana de opción de maven en la pestaña “Goals” elegimos el siguiente código:

clean build package

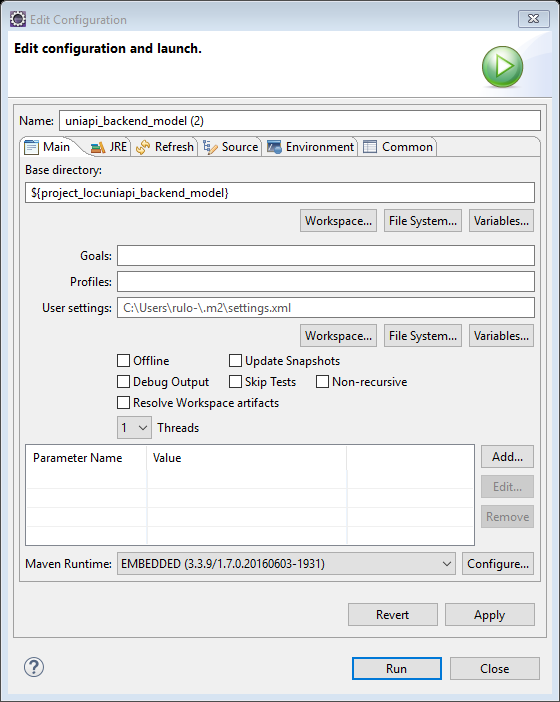


Ilustración . Opciones maven

Tras realizar esto tendremos un ejecutable dentro de la carpeta target del proyecto justo tal que en la figura siguiente:

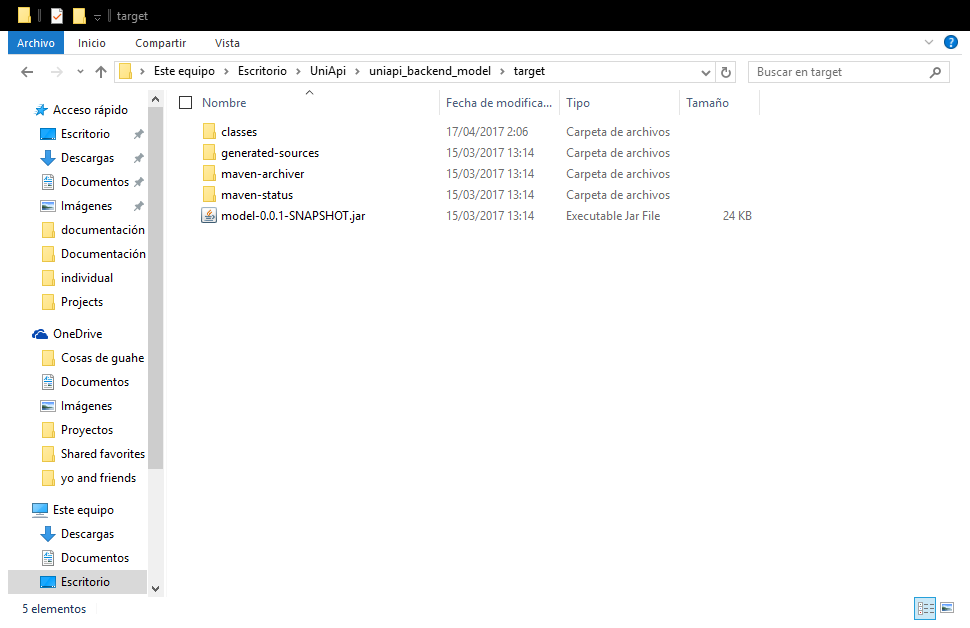


Ilustración 9. Carpeta target

Deberemos crear en la carpeta de todos los proyectos, menos este mismo, una capeta denominada “externalJar”. La carpeta debe ser denomina igual que lo descrito ya que si no, el proceso no funcionara. Tras la creación de la carpeta introducimos el “**model-0.0.1-SNAPSHOT.jar**” en todas las externalJar creadas de los proyectos: “**uniapi\_backend\_enviroment\_execution**”, “**uniapi\_backend\_business**”, “**uniapi\_backend\_apirest**”.

2. **Capa de ejecución**: Para esta capa realizaremos el mismo procedimiento en el proyecto “**uniapi\_backend\_enviroment\_execution**”. Utilizaremos maven con la instrucción descripta. Conseguiremos un denominado “**enviroment\_execution-0.0.1-SNAPSHOT.jar**” tras eso es necesario el trasladarlo a las capas superiores (“**uniapi\_backend\_business**”, “**uniapi\_backend\_apirest**”) a sus carpetas externalJar.

3. **Capa de negocio**: Realizaremos las mismas instrucciones en el proyecto “**uniapi\_backend\_business**” conseguiremos un archivo denominado “**business-0.0.1-SNAPSHOT.jar”** en la carpeta target correspondiente a su proyecto, este archivo será necesario trasladarlo al externalJar del proyecto “**uniapi\_backend\_apirest**”.

4. **Capa REST**: Esta capa es la última. Para que todo funcione correctamente deberá albergar todos los ejecutables como sigue la siguiente figura:

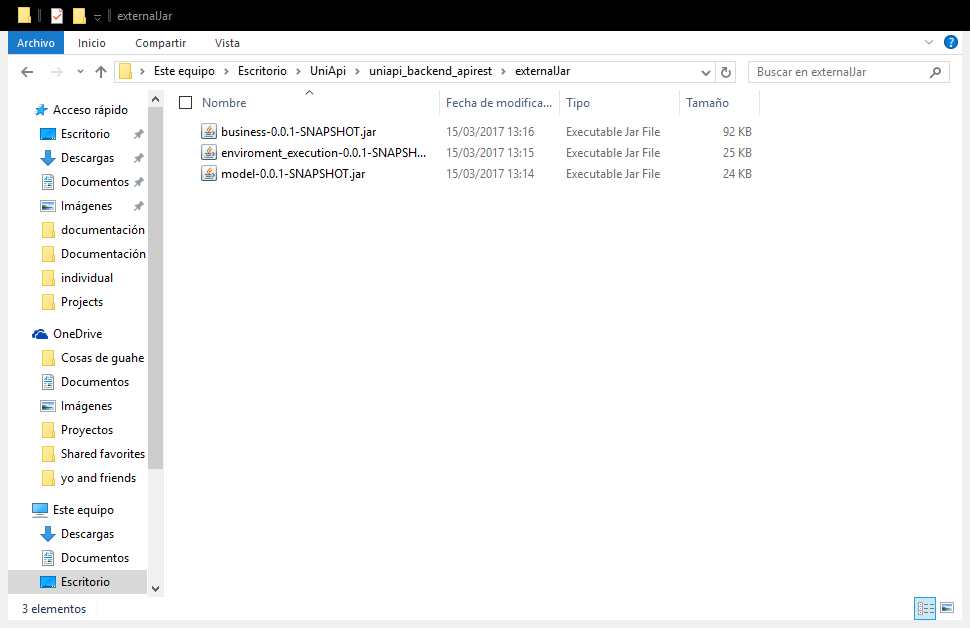


Ilustración . Estado final externalJar del proyecto REST

No es necesario aplicar la compilación a esta capa ya que no se necesita su ejecutable como dependencia. No obstante, puede hacerse para asegurarse que no hay errores si la compilación es correcta. Existirá en su target un archivo denominado “**apirest-0.0.1-SNAPSHOT.jar**”.

**Posibles errores**:

1. **Fallo en las dependencias**: Asegúrate que los archivos que pasas a las carpetas externalJar, contienen los mismos nombres que los descritos en este documento.
2. **JRE Than JDK**: Este fallo se da en los proyectos maven que no tienen un JRE igual que el del compilador que enlaza eclipse. Para eso deberás entran en las propiedades del proyecto (botón derecho en el proyecto). Buscar la pestaña Java y la subpestaña Java Compiler. También deberías observar el Java Buid Path, y comprobar que es la librería JavaSE es el mismo que el JDK.

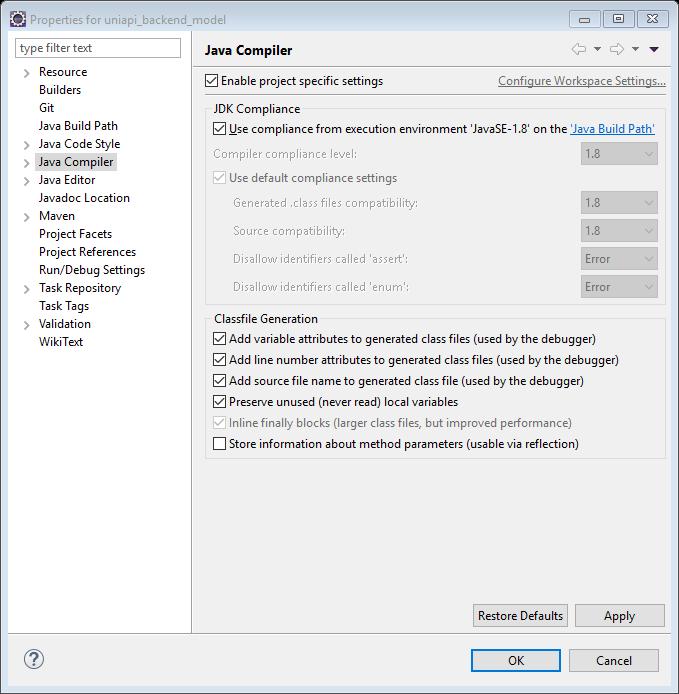


Ilustración . Comprobación JDK

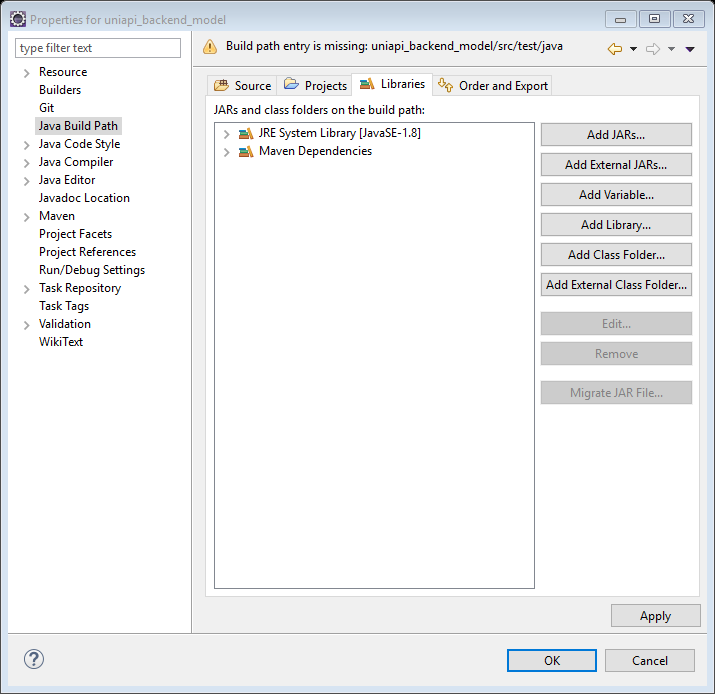


Ilustración . Comprobación JavaSE

* 1. **Instalación de base de datos:**

Para realizar este repositorio se ha utilizado como base de persistencia la base de datos Neo4j. Para la realización de esta instalación Neo4j difiere del sistema operativo en el que opere. Se describirán las dos instalaciones en función de su núcleo Windows o Linux:

* **Windows**: Para instalar la base de datos en Windows vamos a la siguiente dirección url.

<https://neo4j.com/download/community-edition/>

en él se nos descargara un .exe. Instalaremos ese ejecutable y se nos dispondrá un programa denominado neo4j Community editon.

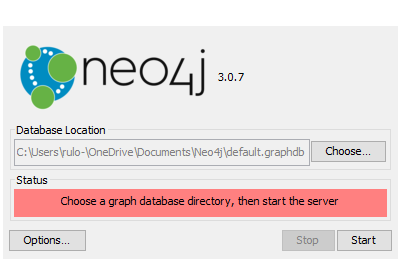


Ilustración . Ventana neo4j

Podemos instalar la base de datos donde queramos, no obstante, la carpeta uniapi tiene una carpeta denominada data. Esta carpeta tiene como objetivo albergar toda la información de la aplicación. Cuando Obtengamos la dirección que queramos solo pulsamos el botón Start, y se nos proporcionara en Status una dirección donde podremos acceder para hacer una gestión de consola.

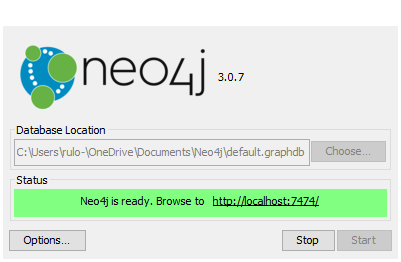


Ilustración . Ventana neo4j Iniciada

* **Linux**: Para instalar la base de datos en Linux iremos a la URL listada a continuación y descargamos la opción de Linux.

<https://neo4j.com/download/community-edition/>

Tras eso obtendremos un archivo comprimo donde se encontrará una carpeta denominada con neo4j-versionDeSoftware. La recomendación del desarrollador es que se descomprima la carpeta albergándola en la carpeta uniapi. Para iniciar la base de datos en linux, será necesario introducirse en la carpeta descomprimida. Buscar la carpeta bin y ejecutar el comando.

./neo4j-community-3.1.1/bin/neo4j console

* + 1. **Generación de estado inicial:**

Para poder realizar una correcta ejecución del repositorio es necesario que la base de datos este en un estado exacto o correcto. Para ello el repositorio necesita unas entidades mínimas para garantizar un correcto funcionamiento. Actualmente el desarrollador no tiene automatizado la generación del caso inicial. Aun así, cuando se realice estas instrucciones si se encuentran ya creadas las entidades es que se ha solventado este problema.

Para empezar, necesitaremos verificar si en la carpeta uniapi existe la carpeta **data**/ la cual deberá contener tres hijos **conf/**, **projects/**, **tmpExecution/**. Si no es así se requiere la creación de estas carpetas.

Después de eso, inicializaremos la base de datos siguiendo las instrucciones del anterior punto. Arrancaremos el navegador web e introduciremos la siguiente dirección URL:

<http://localhost:7474/>

Esto nos introducirá en la base de datos. Nos solicitara que cambiemos la contraseña, lo ejecutaremos e iremos al archivo de la carpeta **data/conf/uniapiConfig.properties** ahí cambiaremos las credenciales a las que instalamos para que el repositorio pueda conectarse a la base de datos.

Después de esto generaremos dos entidades necesarias para el funcionamiento de la aplicación. La entidad administradora original y a entidad del grupo público. Para ello escribimos las siguientes Querys en el lenguaje chyper.

1. CREATE (a:UserLogin {user:”admin@admin.com”,pass:admin,creationTime:0,rol:ADMIN,hashcode:””})
2. CREATE(g:Group{name:”public”,creationDate:0,type:”MAIN\_GROUP”,hashcode:”publicHASHCODE”,sharingGroupPermissions:”[YES,YES,YES]”,projectPropertiesPermissions:”[YES,YES,YES]”,memberGestionPermissions: ”[YES,YES,YES]”,groupCreationPermissions:”[YES,YES,YES]”,description:”PUBLIC GROUP”})
3. MATCH (u:UserLogin),(g:Group) WHERE u.user=”admin@admin.com” AND g.hashcode=”publicHASHCODE” CREATE (u)-[k:KNOWS {dateFrom:0}]->(g);

Después de esto, se podrá inicializar el repositorio sin fallo alguno.

* 1. **Ejecución del repositorio:**

Si se ha realizado correctamente los pasos tendremos por una parte todos los proyectos compilados e inyectados en los diversos proyectos, hasta llegar al proyecto final donde se tendrá todos los proyectos. Aquí tenemos dos tres opciones de ejecución del repositorio:

* **Ejecución mediante eclipse**: Seleccionamos el proyecto de “**uniapi\_backend\_apirest**. Aplicamos la misma metodologia que el punto 3.1 para los demás proyectos, solo que el comando a introducir en el espacio de Goals de la ventana de “maven build…” es diferente. Introduciremos entonces los siguientes comandos para generar la ejecución:  
  clean build package

spring-boot:run

* **Ejecución mediante consola y Maven**: para ello deberemos ir por la jerarquía de consola hasta el espacio de uniapi. Luego deberemos de entrar en el proyecto “**uniapi\_backend\_apirest**”. Después introducimos los siguientes comandos para generar la ejecución:

maven clean build package

maven spring-boot:run

* **Ejecución mediante .jar**: Si utilizamos el primer comando que introducimos en las otras dos opciones generamos un .jar, en la carpeta target. Este .jar puede ser ejecutado generando el siguiente comando:

java -jar apirest-0.0.1-SNAPSHOT.jar

1. **Instalación del servicio web:**
   1. **Instalación de Node.js:**

Para poder iniciar el servicio web del proyecto necesitaremos instalar el motor Node.js de JavaScript. Este motor puede ser instalado desde su página web principal donde puede ser instalado para diversos sistemas operativos. A continuación, se proporciona unas instrucciones básicas para su instalación en diversos sistemas operativos:

* **Windows**: Debe instalarse siguiendo esta dirección web. En él se descargará un instalador. Seguir las instrucciones, e instalarlo. Para poder manejarlo suele venir (en casos de W10) con un acceso directo a un prompt que viene desplegado la variable de entorno de Node.js. En caso contrario mirar en google como desplegar una variable de entorno en el sistema.

<https://nodejs.org/es/>

* **Linux**: Para Linux la cosa es más sencilla solo deberán ejecutar la siguiente instrucción y se instalara automáticamente. (la instrucción es para Ubuntu, para demás instrucciones usen el gestor de paquetes oportuno)

sudo apt-get install node

* 1. **Arranque servidor web:**

Para poder realizar el arranque del servidor web deberemos introducirnos con el prompt del sistema operativo que utilicemos en la jerarquía de la carpeta de UniApi.

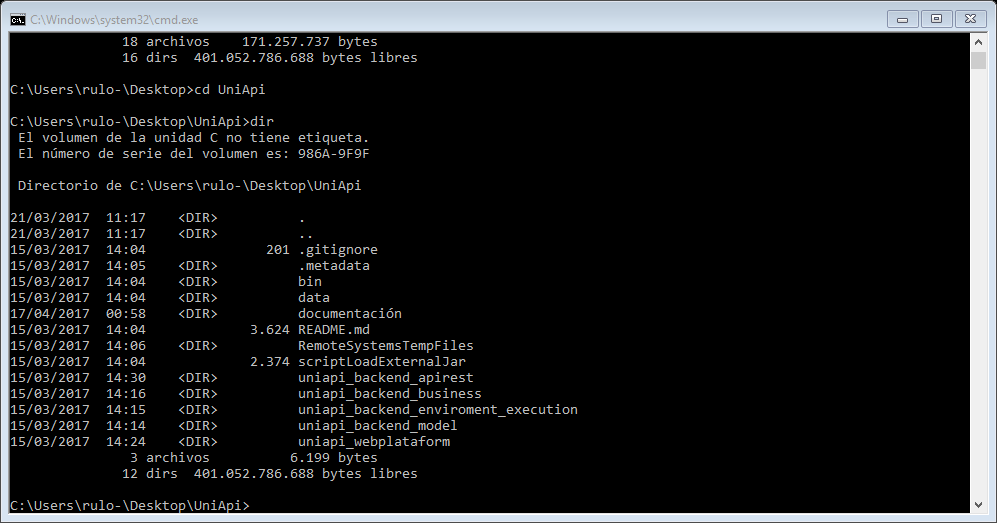


Ilustración . Jerarquía de uniapi (prompt)

Tras acceder a la jerarquía de UniApi deberemos introducirnos en la carpeta “**uniapi\_webplataform**” y tras eso ordenar a Node.js que ejecute el programa.

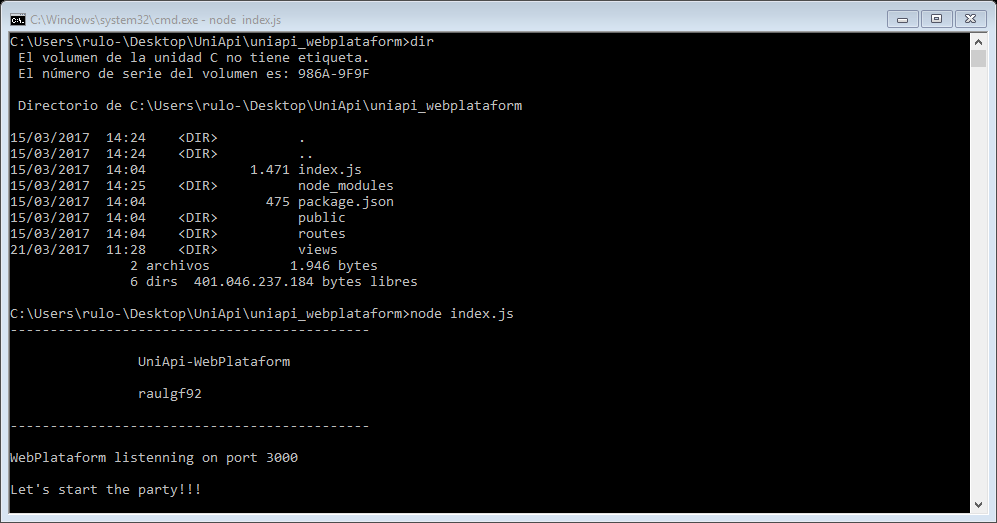


Ilustración . Ejecución del servidor web

1. **Arranque rápido en Ubuntu server:**

Para facilitar el desarrollo, el desarrollador ha producido un arranque rápido del sistema. Este arranque automatiza todas las tareas que están descritas en este documento. Cabe destacar que para que el arranque funcione, el sistema deberá estar en un estado de programas que serán descritos a continuación.

* 1. **Instrucciones de arranque:**

Para arrancar de manera rápida deberemos ubicarnos en la carpeta de la jerarquía principal (carpeta uniapi) mediante el Bash de Ubuntu (conocido como terminal). Tras estar en la ubicación necesaria realizamos la ejecución de las siguientes instrucciones:

cd bin/Linux

./uniapi

* 1. **Listado de programas:**

Los siguientes programas deberán estar instalados correctamente para que funcione el arranque y no se cree que incluso así, este funcione correctamente.

* Maven
* Node.js
* Neo4j (ubicado en la carpeta uniapi)

1. **Manual de utilización del repositorio:**

El repositorio utiliza como método de comunicación el protocolo HTTP. Este protocolo contiene unas directrices especiales que hay que conocer. Pero si se quiere trabajar fácilmente con él, cualquier navegador o ciertas librerías de Python facilitan el uso de HTTP.

* 1. **Iteración habitual:**

Antes de empezar a utilizar el repositorio deberás autenticarte usando la parte del protocolo de comunicación 6.2.1. Si se autentica correctamente se le otorgara una ID de acceso. Después de eso cualquier acción que se ordene al sistema deberá contener el ID. Todas las acciones son sin estado, es decir, terminado la acción el sistema olvidara la acción realizada.

* 1. **Protocolo de comunicación**

El lenguaje que entiende el repositorio es el protocolo HTTP. Son ordenes realizadas mediante un protocolo que debe ser conocido para poder comunicarse a este nivel, No obstante, el navegador o diversas librerías de Python facilitan la comunicación. En este protocolo se entiende que TODAS las entidades (proyectos, grupos, usuarios) están representadas por un ID que será el que lo identifique. Cuando se quiera realizar una acción se deberá saber los ID’s de los afectados.   
http://DIRECCIONIP:8080/ACCION/QUE/SE/ORDENA

Para poder conectarnos al repositorio deberemos conocer la dirección IP o el DNS donde se alberga el servidor, tras eso se tendrá que conectar al puerto :8080 donde se albergara el repositorio. Luego para ordenar una acción se utiliza dominós de URL. Posteriormente se ha realizado una serie de tablas para facilitar la comprensión del repositorio.

http://127.0.0.1:8080/raulgf92@gmail.com/contrasena1

* + 1. **Autenticación:**

Tabla . Protocolo repositorio: Autenticación

|  |  |  |  |
| --- | --- | --- | --- |
| Método | Descripción | Entradas  (Body) | Salidas |
| /{email}/{contraseña}  (HTTP method:GET) | Acción que permite la autenticación en el sistema. | Vacío | Mensaje con el ID del usuario. |
| /{idUser}/whoami  (HTTP method:GET) | Acción que devuelve la entidad del usuario que usa el id | Vacío | Devolverá un mensaje con los datos de autenticación y personales. |

* + 1. **Grupos:**

Todas estas acciones llevan por delante el comando **“/group/”** se deberá de concatenar por delante para que acción funcione **(dir+/group/+accion)**.

* + - 1. **Grupos acciones básicas:**

Tabla .Protocolo repositorio: Grupo acciones básicas

|  |  |  |  |
| --- | --- | --- | --- |
| Método | Descripción | Entradas  (Body) | Salida |
| /{idUser}/{groupID}  (HTTP method:GET) | Devuelve la entidad correspondiente al id del grupo entregado | Vacío | Devuelve un mensaje con la entidad de grupo solicitada. |
| /{idUser}/{groupID}  (HTTP method:DELETE) | Elimina la entidad correspondiente al id del grupo entregado | Vacío | Devuelve el resultado de la acción realizada. |
| /{idUser}/{groupID}  (HTTP method:PATH) | Modifica la entidad correspondiente al id del grupo entregado | Entidad grupo con los datos modificados que se va a modificar | Devuelve el resultado de la acción realizada |
| /{idUser}/all  (HTTP method:GET) | Devuelve todos los grupos creados por el usuario | Vacío | Devuelve un listado de ID’s de grupos que fueron creados por el usuario. |
| /{idUser}/create  (HTTP method:GET) | Facilita la creación de los grupos mediante un ejemplo | Vacío | Devuelve una entidad grupo con datos rellenados con ejemplos para poder crear el grupo. |
| /{idUser}/create  (HTTP method:POST) | Crea el grupo entregado por el usuario | Entidad grupo con los datos que se quieren crear | Devuelve el resultado de la acción realizada |

* + - 1. **Grupos miembros:**

Tabla .Protocolo repositorio: Grupo miembros

|  |  |  |  |
| --- | --- | --- | --- |
| Método | Descripción | Entradas  (Body) | Salidas |
| /{idUser}/{groupID}/member  (HTTP method:GET) | Acción que permite visualizar todos los miembros de un grupo | Vacío | Devuelve un conjunto de ID’s de usuarios. |
| /{idUser}/{groupID}/member/create  (HTTP method:GET) | Acción que permite conseguir un ejemplo de método de entrada para crear miembros de grupos | Vacío | Devuelve una entidad mensaje que se podrá usar para comunicar con el sistema. |
| /{idUser}/{groupID}/member/create  (HTTP method:POST) | Acción que permite crear miembros para un grupo. | Mensaje de comunicación para generar un miembro | Devolverá el resultado sobre la ejecución de la acción. |
| /{idUser}/{groupID}/  member/{memberID}  (HTTP method:DELETE) | Acción que permite borrar un miembro de un grupo | Vacío | Devolverá el resultado sobre la ejecución de la acción |

* + - 1. **Grupos subgrupos:**

Tabla .Protocolo repositorio: Grupo subgrupos

|  |  |  |  |
| --- | --- | --- | --- |
| Método | Descripción | Entradas  (Body) | Salidas |
| /{token}/{groupID}/subgroups  (HTTP method:GET) | Accion que devuelve todos los subgrupos de un grupo | Vacío | Devuelve un conjunto de ID’s de grupos |
| /{token}/{groupID}/subgroups/{subgroupID} (HTTP method:GET) | Acción que permite conseguir la información de que un grupo es subgrupo de otro | Vacío | Devuelve una entidad mensaje que contiene la información de la orden |
| /{token}/{groupID}/subgroups/{subgroupID}  (HTTP method:POST) | Acción que permite hacer que un grupo sea subgrupo de otro grupo | Vacío | Devolverá el resultado sobre la ejecución de la acción. |
| /{token}/{groupID}/subgroups/{subgroupID} (HTTP method:DELETE) | Acción que permite revertir que un grupo sea subgrupo de otro | Vacío | Devolverá el resultado sobre la ejecución de la acción |

* + - 1. **Grupos proyectos:**

Tabla .Protocolo repositorio: Grupo proyectos

|  |  |  |  |
| --- | --- | --- | --- |
| Método | Descripción | Entradas  (Body) | Salidas |
| /{idUser}/{groupID}/contain/project  (HTTP method:GET) | Acción que permite ver todos los proyectos contenidos en un grupo. | Vacío | Devuelve un mensaje con un conjunto de ID’s de proyectos contenidos en un grupo |
| /{idUser}/{groupID}/contain/project/{projectID}  (HTTP method:GET) | Acción que permite visualizar la información de cuando un proyecto es contenido en un grupo | Vacío | Devuelve un mensaje con la información de la acción |
| /{idUser}/{groupID}/contain/project/{projectID}  (HTTP method:POST) | Acción que permite introducir proyectos en el grupo | Vacío | Devolverá el resultado sobre la ejecución de la acción. |
| /{idUser}/{groupID}/contain/project/{projectID}  (HTTP method:DELETE) | Acción que permite borrar un proyecto de un grupo | Vacío | Devolverá el resultado sobre la ejecución de la acción |

* + 1. **Proyectos:**

Todas estas acciones llevan por delante el comando **“/project/”** se deberá de concatenar por delante para que acción funcione **(dir+/project/+accion).**

Tabla .Protocolo repositorio: Proyectos

|  |  |  |  |
| --- | --- | --- | --- |
| Método | Descripción | Entradas  (Body) | Salida |
| /{idUser}/{projectID}  (HTTP method:GET) | Devuelve la entidad correspondiente al id del proyecto entregado | Vacío | Devuelve un mensaje con la entidad de proyecto solicitada. |
| /{idUser}/{projectID}  (HTTP method:DELETE) | Elimina la entidad correspondiente al id del proyecto entregado | Vacío | Devuelve el resultado de la acción realizada. |
| /{idUser}/{projectID}  (HTTP method:PATH) | Modifica la entidad correspondiente al id del proyecto entregado | Entidad proyecto con los datos modificados que se va a modificar | Devuelve el resultado de la acción realizada |
| /{idUser}/all  (HTTP method:GET) | Devuelve todos los proyectos creados por el usuario | Vacío | Devuelve un listado de ID’s de proyecto que fueron creados por el usuario. |
| /{idUser}/create  (HTTP method:GET) | Facilita la creación de los proyectos mediante un ejemplo | Vacío | Devuelve una entidad proyectos con datos rellenados con ejemplos para poder crear el proyecto. |
| /{idUser}/create  (HTTP method:POST) | Crea el proyecto entregado por el usuario | Entidad proyecto con los datos que se quieren crear | Devuelve el resultado de la acción realizada |

* + 1. **Ejecuciones:**

Todas estas acciones llevan por delante el comando **“/execution/”** se deberá de concatenar por delante para que acción funcione **(dir+/execution/+accion)**.

Tabla .Protocolo repositorio: Ejecuciones

|  |  |  |  |
| --- | --- | --- | --- |
| Método | Descripción | Entradas  (Body) | Salida |
| /{idUser}/{groupID}/{projectID}  (HTTP method:POST) | Crear una entidad ejecución y la ejecuta. Usando un proyecto con projectID albergado en el grupo de groupID | Archivo JSON, que será utilizado como la entrada de la ejecución del proyecto. | Devuelve el resultado de la acción realizada |
| /{idUser}/{executionID}  (HTTP method:GET) | Devuelve la entidad correspondiente al id de la ejecución entregada | Vacío | Devuelve un mensaje con la entidad de ejecución solicitada. |
| /{idUser}/{executionID}  (HTTP method:DELETE) | Elimina la entidad ejecución representada por executionID. En el caso de que este en ejecución se detendrá. | Vacío | Devuelve un mensaje con la entidad de ejecución solicitada. |
| /{idUser}/{executionID}/project  (HTTP method:GET) | Devuelve la información del proyecto que está ejecutando la entidad ejecución representada por el executionID | Vacío | Devuelve un mensaje con la entidad del proyecto de la ejecución solicitada. |
| /{idUser}/running  (HTTP method:GET) | Devuelve todas las ejecuciones del usuario que se están ejecutando. | Vacío | Devuelve un mensaje con el conjunto de las identidades de las ejecuciones en ejecución |
| /{idUser}/finish  (HTTP method:GET) | Devuelve todas las ejecuciones del usuario que hayan finalizado | Vacío | Devuelve un mensaje con el conjunto de las identidades de las ejecuciones finalizadas |

* + 1. **Administración:**

Todas estas acciones llevan por delante el comando **“/admin/”** se deberá de concatenar por delante para que acción funcione **(dir+/admin/+accion)**.

Tabla .Protocolo repositorio: Administración

|  |  |  |  |
| --- | --- | --- | --- |
| Método | Descripción | Entradas  (Body) | Salida |
| /{idUser}/createAccount  (HTTP method:POST) | Crea una cuenta con el rol que elija el administrador | Entidad userLogin con los datos rellenos para su creación | Devuelve el resultado de la acción realizada |
| /{idUser }/allUsers  (HTTP method:GET) | Devuelve todos los usuarios del sistema | Vacío | Devuelve un mensaje con el conjunto de identidades de usuarios |
| /{idUser}/allGroups  (HTTP method:GET) | Devuelve todos los grupos del sistema | Vacío | Devuelve un mensaje con el conjunto de identidades de grupos |
| /{idUser}/allProjects  (HTTP method:GET) | Devuelve todos los proyectos del sistema | Vacío | Devuelve un mensaje con el conjunto de identidades de proyectos |
| /{idUser}/allExecution  (HTTP method:GET) | Devuelve todas las ejecuciones del sistema | Vacío | Devuelve un mensaje con el conjunto de identidades de ejecuciones |

* + 1. **Navegación:**

Todas estas acciones llevan por delante el comando **“/path/”** se deberá de concatenar por delante para que acción funcione **(dir+/path/+accion)**.

Tabla .Protocolo repositorio: Navegación

|  |  |  |  |
| --- | --- | --- | --- |
| Método | Descripción | Entradas  (Body) | Salida |
| /{idUser}/main  (HTTP method:GET) | Devuelve los proyectos públicos y los grupos superiores del usuario | Vacío | Devuelve un mensaje con todas las identidades de proyectos y grupos que contiene la navegación actual |
| /{idUser}/enter/{groupID}  (HTTP method:GET) | Devuelve todos los proyectos y grupos que puede visualizar el usuario | Vacío | Devuelve un mensaje con todas las identidades de proyectos y grupos que contiene la navegación actual |
| /{idUser}/public  (HTTP method:GET) | Devuelve la información del grupo publico | Vacío | Devuelve un mensaje con la entidad del grupo publico |

* 1. **UniApi (JS):**

Para facilitar el desarrollo de aplicaciones web utilizando el repositorio, se ha diseñado una librería para angular.js extrapolable a cualquier tecnología que use JavaScript (utilizando JQuery). Esta API está hecha mediante funciones que utilizan el protocolo explicado en el punto 6.2. La API facilita a los desarrolladores de aplicaciones web que no quieren entender el protocolo de comunicación del repositorio el usarlo.

Para poder usarlo. Deberemos instalarlo en la página web incluyéndolo con el siguiente trozo de código HTML. Cabe destacar que deberá poder ser accedido por el servidor web que sustenta la aplicación web. Las funciones son:

Tabla UniApi.js: Nombre de funciones

|  |  |  |
| --- | --- | --- |
| Nombre de las funciones | | |
| Whoami() | **getMyProjects**() | **deleteSubgroup**() |
| changePass() | **getMyGroups**() | **getMainPathGroups**() |
| changeBio() | **getProject**() | **getPathGroups**() |
| createAccount() | **updateProject**() | **projectsInsideGroup**() |
| getAllUsers() | **deleteProject**() | **getProjectsInsideGroup**() |
| getAllProjects() | **createProject**() | **getExecutionsRunning**() |
| getAllGroups() | **createGroup**() | **getAllExecutionsRunning**() |
| getGroup() | **updateGroup**() | **getExecution**() |
| deleteGroup() | **inviteToGroup**() | **getProjectOfExecution**() |
| getAllGroupMembers() | **removeMemberOfTheGroup**() | **getExecutionFinish**() |
| getSubgroups() | **getPathProjects**() | **getAllExecutionsFinish**() |
| createSubgroup() | **putGroupProject**() | **executedProject**() |
| getMainPathProjects() | **deleteProjectInsideGroup**() |  |

1. **Manual de utilización de la aplicación web:**

La aplicación web utilizara para comunicarse con el usuario una serie de formularios y listados que facilitaran al usuario la interacción y gestión con el repositorio. A continuación, se desplegará unos tutoriales para realizar diferentes tareas.

* + 1. **Autenticarse y crear una cuenta:**
    2. **Navegar por los grupos:**
    3. **Crear un grupo:**
    4. **Crear un proyecto:**
    5. **Crear una ejecución:**
    6. **Gestionar un proyecto propio:**
    7. **Gestionar un grupo propio:**
    8. **Gestionar un proyecto propio:**
    9. **Gestionar una ejecución propia:**