Escuela Superior
de Ingeniería y Tecnología
Universidad de La Laguna

Trabajo de Fin de Grado

Grado en Ingeniería Informática

Github Classroom Teacher Assistant

Github Classroom Teacher Assistant

Carlos Andrés Castro García

La Laguna, *15* de *Marzo* de 2019

D. **Casiano Rodríguez León**, con N.I.F. 42.020.072-S profesor Catedrático de Universidad adscrito al Departamento de Ingeniería Informática y de Sistemas , como tutor

C E R T I F I C A (N)

Que la presente memoria titulada:

“Github Classroom Teacher Assistant”

ha sido realizada bajo su dirección por D. **Carlos Andrés Castro García**,

con N.I.F. 42.221.693-H.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 21 de noviembre de 2018

Agradecimientos

A mis padres, mi hermano y mi abuela por estar ahí durante todos los años del grado.

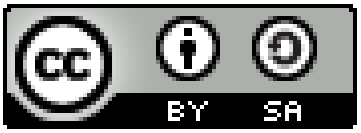
A mi compañera de estudios por darme fuerzas día a día durante todo este último curso, prestándome su apoyo en todo momento y convirtiendo días enteros de trabajo en momentos muy agradables.

A mis amigos, por ayudarme a desconectar y relajarme siempre que lo necesité.

A mis compañeros del grado, siempre atentos a cualquier cosa en la que pudiesen ayudar.

A mi tutor Casiano por la gran implicación durante todo el proyecto.

Licencia



Resumen

Github es una plataforma web dedicada al desarrollo de proyectos con Control de Versiones Git. Aporta un gran abanico de herramientas que facilitan el trabajo en proyectos, tanto de forma colaborativa como de forma individual. En el ámbito educativo, Github presenta herramientas muy interesantes, como el uso de Organizaciones.

Una Organización es un elemento de Github que permite agrupar:

- *Usuarios con el fin de formar equipos.*
- *Repositorios dentro de la organización.*

Estas dos características aportan un gran potencial al uso de Organizaciones, ya que en caso de un profesor, le permiten registrar los distintos grupos de trabajo dentro de cada una de sus asignaturas y agrupar todos sus repositorios.

Github Classroom utiliza las Organizaciones para aportar nuevas herramientas a los profesores, permitiendo crear y realizar asignaciones de tareas a cada uno de los grupos de trabajo, o a cada uno de los usuarios de forma independiente. Estas funcionalidades simplifican el proceso de asignación de tareas, pero a la hora de realizar su seguimiento, es necesario el uso de un navegador web, una terminal y un editor de texto.

Este trabajo tiene como objetivo reducir todas esas herramientas a sólo una: el editor de texto.

Palabras clave: Github, Git, Repositorio, Organización.

Abstract

Github is a web-based platform used to the project development with Git version control. It provides a wide range of tools that simplify working in individual and collaborative projects. Github have some interesting tools to the educational area, like Organizations.

An Organization is a Github element that allows to group:

- *Users in order to form teams.*
- *Repositories inside the Organization.*

These two features bring great potential to the Organizations, in case of teachers, they allow to create differents working teams in a classroom and to group all their repositories.

Github Classroom uses the Organizations to provide new tools to teachers, allowing them to create and make assignments to each of the working groups, or to each of the users individually. These features simplify the assignment process, but in case of monitoring them, it is necessary to use a web browser, a terminal and a text editor.

This project tries to reduce all those tools to only one: the text editor.

Keywords: Github, Git, Repository, Organization.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

1.1 Contexto.

La importancia del uso de Github en el desarrollo de aplicaciones es muy grande, tanto que en casi todos los caso se hace uso de él, o de otra herramienta que cubra sus mismas funciones. En el ámbito educativo tenemos otras plataformas que complementan Github, como es el caso de Github Classroom. Crear un repositorio es una tarea muy simple haciendo uso de Github, así como asignarlo a una persona, pero hacerlo para un número moderado de personas lo convierte en una tarea repetitiva y costosa en lo que respecta al gasto de tiempo. Github Classroom trabaja para solucionar ese déficit permitiendo crear y asignar repositorios a grandes grupos de personas de forma simultánea. Para ello, es necesario crear una organización en Github, que será asociada a un elemento classroom en Github Classroom. Una vez creada la asignación, se genera un link, a través del cual todos los interesados pueden generar su repositorio de forma automática.

Figura 1.1: Logo Github

Figura 1.2: Logo Github Classroom

Una vez resuelto dicho problema, un profesor puede preparar distintas asignaciones de forma rápida y cómoda, pero a la hora de realizar el seguimiento de los repositorios, el proceso se ralentiza una vez más. Para comprobar el funcionamiento de los proyectos, es necesario realizar la descarga de los repositorios uno a uno. Github Classroom presenta una aplicación con el fin de atender a dicha situación, GitHub Classroom Assistant, que permite realizar la clonación de todos los repositorio de manera simultánea. Esta es la única función de esta nueva aplicación, que pese a solventar el problema de forma correcta, lo hace de forma engorrosa. Para su uso debemos autenticarnos, pero una vez hecho no nos muestra nuestras asignaciones sino que requerimos un enlace generado en la página web.

Con todas estas herramientas nos encontramos que para realizar el proceso de seguimiento

de un proyecto debemos hacer uso de 4 programas al mismo tiempo:

- Navegador web: Necesitamos hacer uso de la página de Github Classroom para obtener el link de descarga.

- Terminal: Tanto para clonarlos de forma individual, como para comprobar el funcionamiento del proyecto.

- Editor de texto: Revisar el código o modificarlo.

- GitHub Classroom Assistant: Realizar la clonación de los repositorios.

Esta situación nos hace pensar en la posibilidad de agrupar todas estas funcionalidades en una única herramienta y con el editor de texto Atom, dicha posibilidad cobra sentido con el desarrollo de un plug-in. Atom es un editor de texto desarrollado en Electron con soporte de plug-ins escritos en NodeJs.

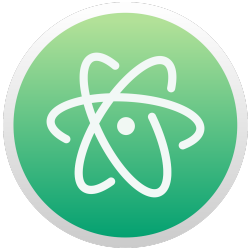


Figura 1.3: Logo Atom

Figura 1.4: Logo NodeJs

1.2 Objetivos.

Este trabajo busca reducir la cantidad de herramientas necesarias para realizar el seguimiento de un proyecto haciendo uso del editor de texto Atom. Se construirá un plug-in con capacidad de mostrar todas nuestras organizaciones y realizar la clonación de los repositorios que contienen.

1.2.1 Autenticación.

El primer objetivo dentro del proyecto consiste en lograr la autenticación del usuario con github, para así generar un token que permita acceder a toda la información del usuario haciendo uso de su API.

1.2.2 Usabilidad.

Es una característica fundamental que se ha priorizado en todo momento dentro de las limitaciones que la plataforma Atom permite. Este editor mantiene muchas funcionalidades bloqueadas, por lo tanto, se han de dedicar muchos recursos con el fin de adaptar el plug-in a un estado que se considere usable.

1.2.3 Organizaciones.

Las organizaciones son el pilar fundamental del proyecto, por lo tanto se trabajará en todo momento alrededor de ellas. Se deberá obtener toda la información que puedan aportar con el fin de evitar posibles errores al trabajar con sus distintos repositorios. Se desea mostrar una vista listando todas las organizaciones de las que dispone el usuario.

1.2.4 Repositorios.

Los repositorios son un factor delicado a la hora de desarrollar el proyecto, ya que el fin del proyecto es realizar su descarga de forma correcta y cómoda.

1.2.5 Buscador.

Otro objetivo dentro del proyecto es la implementación de un buscador. Este elemento tendrá varias finalidades:

- La primera será poder encontrar una organización específica en caso de poseer una amplia lista de ellas.

- La segunda finalidad será poder crear una lista de repositorios haciendo uso de expresiones regulares que permita realizar la clonación únicamente de un conjunto de elementos determinado.

1.3 Retos.

Desde el inicio del proyecto, se encontró un conjunto de 7 retos que se debía solventar durante el desarrollo del proyecto.

1.3.1 Atom.

Para el desarrollo de plug-ins para Atom es necesario hacer uso de una estructura determinada por la plataforma. Atom posee una documentación desde la cual obtener los distintos

procedimientos que se han de seguir, pero hay muchas carencias en ella que abarca, desde la ausencia absoluta de ejemplos en el uso de sus distintas entidades y funciones, hasta la inexistencia en dicha documentación de muchas funciones de uso regular.

Un procedimiento rutinario a la hora de realizar un nuevo proyecto, como es el estudio de la documentación de la plataforma se torna en uno de los mayores retos de todo el proyecto.

1.3.2 Almacenamiento.

Para el acceso a la información de cada usuario es necesario el uso de un token procedente de Github. Este token es enviado una única vez, por lo que tras generarlo, no se puede volver a solicitar. Para solventar este problema se ha de buscar la forma de almacenar esta información de forma segura, para evitar la generación de un token por cada vez que se quiere acceder al plug-in.

1.3.3 Obtención de Información.

La obtención de información es uno de los pasos críticos dentro del proyecto. Github Classroom carece de API por lo que es necesario encontrar un camino alternativo para ello. Sabiendo que Github Classroom trabaja con las organizaciones de Github, se procede a analizar esta API en su lugar.

1.3.4 Asincronía.

La asincronía es la mayor dificultad a la hora de desarrollar este proyecto. Para la consecución del trabajo es necesario realizar múltiples peticiones, pero estas han de ir en un orden concreto, por ejemplo, el token inicial se ha de obtener antes de solicitar las organizaciones. Por otro lado, esperar por cada una de las respuestas puede ralentizar el funcionamiento del plug-in en exceso, por lo que hay que detectar cuando es necesario realizar esta espera y cuando puede realizarse en segundo plano.

1.3.5 Encapsulamiento.

El encapsulamiento es un una característica fundamental en la programación orientada a objetos. En este caso, el proceso de encapsulamiento complica en gran medida el control de los procesos asíncronos.

1.3.6 Clonación.

El proceso de clonación requiere una gran cantidad de comprobaciones para evitar problemas no previstos.

- Directorios. Decidir cuál es el directorio por defecto en el que se realizará la clonación es un paso muy importante. En caso de decidir un directorio estático, este debe existir en todos los sistemas donde se ejecute el plug-in. Por otro lado, si se decide utilizar el directorio del proyecto, habrá que tener en cuenta que se puede ejecutar Atom fuera de proyectos. Otra situación a tener en cuenta es la existencia o inexistencia del directorio de clonación.

- Existencia del repositorio. Al realizar la clonación de un repositorio se pueden producir 2 situaciones, que el repositorio no existiese previamente o que si existiese. En caso de que se hubiese clonado previamente el repositorio, se tratará de realizar la actualización del repositorio. Este proceso puede generar errores, como conflictos al realizar un PULL de la información.

1.3.7 Multiplataforma.

El último reto que se nos presentará durante la ejecución del proyecto es habilitar el plug-in para ser utilizado en las principales plataformas, tanto Windows como Linux y MacOs. Para ello, habrá que prestar una especial atención al sistema de directorios a la hora de realizar la clonación de repositorios.

Capítulo 2 Procedimientos

Durante el proyecto se han de realizar una serie de procedimientos con el fin de concluir con éxito los objetivos marcados, desde la elección de las distintas herramientas de las que se hará uso, hasta la publicación del plug-in en el gestor de paquetes de atom.

2.1 Selección de herramientas.

El primer paso a la hora de realizar el proyecto será la elección de herramientas y el estudio de sus documentaciones. Es el procedimiento más largo del proyecto. Las herramientas requeridas son:

- Editor de texto. El editor de texto que elegiremos es Atom. Atom tiene soporte para plug-ins escritos en NodeJs, lo que además nos permite hacer uso de todos los módulos que este framework nos presenta.

- Github API. Desde npm encontramos un paquete que nos permite realizar la conexión a la API de Github de forma cómoda, Octonode. Esta herramienta nos permite tanto solicitar un token de autenticación como solicitar las listas de repositorios y organizaciones, lo que es la herramienta ideal para este fin.

- Comandos Git. Para el uso de git desde el framework NodeJs encontramos el módulo simple-git. Esta herramienta nos permitirá realizar la clonación de repositorios y en caso de que ya exista, realizar el PULL de su información. Además nos ofrece suficiente información para manejar los distintos errores.

- Edición del HTML. La creación de elementos HTML y su edición presenta un gran problema en Atom, ya que muchos eventos se encuentran limitados por la plataforma. React nos permite encontrar formas alternativas para utilizar todos los elementos deseados sin estar limitados a estas restricciones. Para el uso de React en NodeJs utilizaremos los paquetes React y ReactDom.

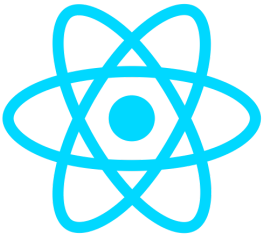


Figura 2.1: Logo React.

- Manejo de Directorios. En el momento de hacer uso de git para realizar la clonación de un repositorio en el equipo, deberemos comprobar la existencia del directorio objetivo. Si este directorio no existe deberemos crearlo, pero si existe, comprobaremos la existencia del propio repositorio. Para este conjunto de procesos utilizaremos los paquetes de NodeJs File System y Path.

2.2 Desarrollo.

Una vez seleccionadas las herramientas que usaremos durante el proyecto comenzaremos con su desarrollo. El primer paso del desarrollo será el diseño de los elementos que tendrá. El plug-in tendrá un total de 4 vistas:

- Log-in. La primera vista estará dedicada al log-in. Con ella se realizará la solicitud del token que nos autenticará.

- Listado de Organizaciones. Contendrá todas las organizaciones del usuario. Estas organizaciones se encontrarán paginadas, y al hacer click en una de ellas, iremos al conjunto de repositorios que tenemos. Además, debe tener la posibilidad de filtrar las organizaciones con el fin de encontrar 1 en concreto.

- Listado de Repositorios. Mostrará la lista de repositorios a los que tenemos acceso dentro de una organización. Debe contener un input de filtrado, y un botón que clonará todos los repositorios de la lista. Al hacer click en un repositorio en concreto, este será clonado en la dirección de trabajo. Por otro lado, en esta vista encontraremos un botón que nos mostrará un elemento Modal Aauxiliar.

- Modal Auxiliar. Es donde podremos seleccionar nuestro directorio de trabajo, y donde encontraremos todos los errores que se hayan producido al realizar una clonación.

Principalmente, dividiremos el código en 5 clases:

- GhcrtaModal. Es una clase auxiliar que crea un elemento Modal (Modal Auxiliar) que se usará tanto para la elección de directorios, como para comprobar los distintos errores que se produzcan durante la clonación de repositorios.

- GhcrtaRepo. Esta clase contendrá todas las funciones que se pueden hacer con un repositorio.

- GhcrtaOrgs. Esta clase contendrá todas las funcionalidades que se pueden hacer con una organización, como mostrar sus repositorios, realizar búsqueda de repositorios, etc. Mostrará la lista de repositorios.

- GhcrtaClient. Es el primer punto de contacto con la API de Github, por lo que realizará la autenticación y mostrará la primera vista con todos las Organizaciones del usuario. Además de contener la lista de organizaciones también genera la vista de Log-In.

- GhcrtaView. Será el contenedor principal de las vistas del proyecto.

El siguiente paso del desarrollo será la implementación de los elementos diseñados en el paso anterior. Este proceso se puede dividir en varias fases o hitos.

- Autenticación. Esta primera fase consistirá en el desarrollo de la autenticación de la aplicación. Además, se ha de lograr que el usuario solo tenga que realizar el log-in la primera vez que accede a la herramienta.

- Vistas. Se construirán las vistas anteriormente mencionadas.

- Búsqueda. Los repositorios y organizaciones se pueden filtrar correctamente.

- Clonación. Se clonan correctamente los repositorios deseados.

- Directorio de trabajo. Se puede modificar el directorio de clonación de forma correcta.

El último paso del proceso de desarrollo será realizar el control de errores. Este procedimiento consistirá en ofrecer rutas alternativas al software en caso de que se produzcan errores en cualquiera de las funciones utilizadas.

2.3 Publicación.

El último procedimiento a realizar, una vez el software esta finalizado será la publicación del plug-in en el gestor de paquetes de Atom. Una vez realizada la publicación, el paquete será accesible para todo el mundo y se podrá instalar con un simple click.

Capítulo 3

Desarrollo

En el anterior capítulo hablamos sobre los distintos procedimientos necesarios para la realización del proyecto. En este, entraremos a detallar el proceso de desarrollo del proyecto, una vez tenemos todas las herramientas elegidas.

3.1 Estructura base.

Atom permite la creación de la estructura base de un paquete mediante un comando. Para ejecutar este comando, debemos abrir la paleta de comando mediante:

View/Toggle Command Palette

O también mediante el acceso rápido ctrl + shift + p. Una vez la paleta de comandos está abierta, ejecutas *Generate Package.*

Este comando generará una estructura que contiene el siguiente esquema de archivos:

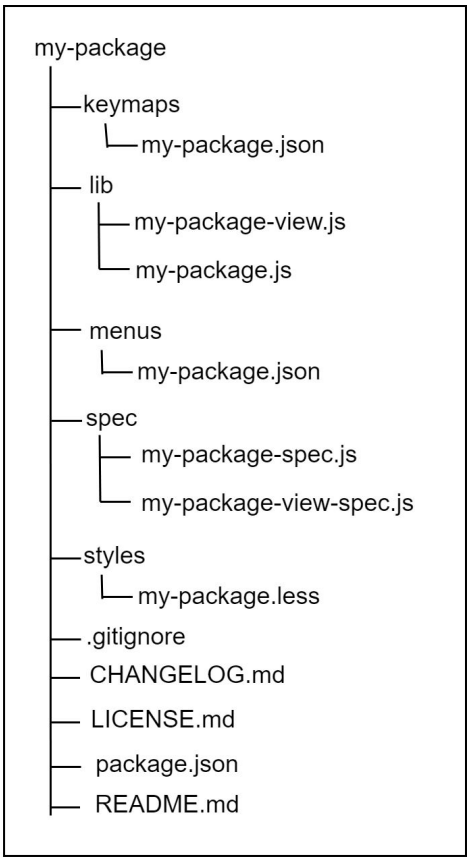


Figura 3.1: Estructura de Paquete

- Keymaps. El fichero my-package.json generado en este directorio se utiliza para definir accesos rápidos de teclado para los distintos comandos de nuestra aplicación.

- Lib. En este directorio se ubicará todo el código de nuestro proyecto. El fichero my-package.js contiene el despliegue del proyecto, mientras que my-package-view.js es una vista.

- Menú. En el fichero my-package.json se definirán los comandos que aparecerán en el menú Packages de Atom.

- Spec. Directorio reservado para añadir las pruebas para el proyecto. Atom utiliza Jasmine como framework de pruebas.

- Style. Directorio reservado para los estilos del proyecto. Para este apartado, Atom hace uso de Less.

El package.json del proyecto funcionará de la misma manera que en node.js. Recoge informaciones del proyecto, como nombre, enlace a github, o dependencias entre otros.

Esta estructura inicial ejecutará la ventana modal mostrada a continuación:

Figura 3.2: Vista estructura inicial.

3.2 Vista Contenedor.

Una vez generada la estructura inicial del paquete, borramos la vista existente señalada en la **Figura 3.2**, y creamos nuestra vista contenedora.

La vista contenedora será un nuevo panel situado por defecto en la franja derecha del editor de texto. Para la creación de este panel, debemos crear un nuevo div en el DOM, al que asignaremos como id: container. Por tanto, nuestra clase GhcrtaView creará este elemento en su

constructor, haciendo uso del siguiente código:

```
this.element = document.createElement('div');
this.element.setAttribute('id', 'container');
```

Una vez definido este elemento en el constructor, procedemos a crear varias funciones con el fin de garantizar el correcto funcionamiento que debe tener.

- getTitle(). Esta función debe devolver el nombre que tendrá el encabezado del panel en Atom. En nuestro caso devuelve: *GHCRTA*

- getURI(). Esta función devuelve una dirección única dentro de atom en la que podemos montar un comando. En nuestro caso, para la ejecución de nuestro panel usamos: *atom://ghcrta*

- getDefaultLocation(). En esta función, definiremos la posición por defecto del panel. Al devolver *right,* indicamos que su posición por defecto debe estar a la derecha del editor.

- getAllowedLocations(). Sirve para definir las distintas posiciones en las que queremos permitir que se sitúe nuestro panel. Devuelve una lista de sus posiciones disponibles, incluyendo en este plug-in *right* y *left.*

3.3 Login.

La siguiente fase dentro del proceso de desarrollo será la construcción de un login que nos permita realizar la autenticación con Github. Para ello, crearemos una nueva clase, *GhcrtaClient*. En esta clase definiremos una función *renderLogin*, que toma como atributos el contenedor donde se pintará el login, y un mensaje de error, y renderiza mediante React el formulario para introducir usuario y contraseña.

Los datos serán recogidos por la función *getLogin*, que se encargará de realizar la solicitud de autenticación con github haciendo uso del módulo Octonode. Si las credenciales son correctas, la respuesta a la petición devolverá un token de acceso personal que será creado en su usuario de github, y será enviado al plug-in, completando así la autenticación y llamando al método *logedIn*. En caso de que el token ya exista en github, se informará de que debe ser borrado y se redirigirá a la vista de login. Por otro lado, si las credenciales son erróneas, se llamará a la vista de

login con el mensaje de error *Usuario o contraseña incorrecto.*

3.4 Almacenamiento.

Como mencionamos en la fase anterior, al realizar el login recibiremos un token para el acceso a github. Si este token no es almacenado, debemos solicitar uno nuevo por cada autenticación que realicemos, lo que supone un problema.

Encontramos dos métodos de almacenamiento de información haciendo uso de Atom:

- Estados. Podemos mantener información entre sesiones haciendo uso de los estados. Esto nos permite retomar la sesión en un plugin, según la información que deseemos mantener, tras cerrar el editor de texto. El problema de este método es que si se cierra Atom sin tener abierto el plug-in, este proceso no se lleva a cabo.

- Schemas. Podemos definir un schema en el package.json del proyecto que guarda los datos deseados en la configuración de Atom. Esta información es permanente, y podremos acceder a ella en cualquier momento.

El método del que haremos uso es el segundo, ya que a diferencia del primero, es mucho más seguro. Para ello, incluiremos una entrada al package.json de tipo string con el fin de almacenar el token:

```
"configSchema": {
  "token": {
    "type": "string",
    "default": "",
    "description": "Authentication github token"
  }
}
```

Luego, haciendo uso de los métodos de configuración de atom lo almacenamos:

```
atom.config.set('ghcrta.token', token)
```

Y podremos acceder a él de la misma forma:

```
atom.config.get('ghcrta.token')
```

3.5 Organizaciones.

Una vez creado el proceso de login, se procede a crear la vista con el listado de organizaciones dentro de GhcrtaClient. Una vez se dispone del token, se llama al método *logedIn* que realiza la primera solicitud para obtener las organizaciones y muestra el listado. La estructura donde se muestra está dividida en tres secciones:

- La franja superior contiene un input para la búsqueda de organizaciones. Este input toma el texto escrito y lo introduce en una expresión regular, filtrando la lista para que solo se mantengan los elementos que encajen con la expresión. De esta manera, podemos usar el input tanto escribiendo el nombre de la organización deseada (**Figura 4.6**) como utilizando expresiones regulares(**Figura 4.5**).

- La franja central ocupa la mayor parte del panel creado. En esta franja se mostrará la lista de organizaciones. Las organizaciones se muestran en grupos de 15 siempre que sea posible.

- La franja inferior contiene dos botones encargados de manejar la paginación. Uno para pasar a la siguiente página y otro para volver a la anterior.

Al igual que el login, la solicitud de las organizaciones la realizaremos con Octonode. Este paquete permite realizar la solicitud paginada de elementos, al indicar de cuanto se desea cada lista y el número de página deseado, haciendo uso de la siguiente función:

ghme.orgs(pag, size, callback)

Pese a tener esta posibilidad de paginado, no será la que utilizaremos. Las peticiones a github son un proceso muy lento, y en caso de paginar directamente con estas peticiones, el programa tardaría mucho en realizar cualquier acción. Por tanto, se ha elegido realizar la solicitud de todas las organizaciones y paginarlas una vez obtenidas.

Las peticiones con Octonode solo devuelven un máximo de 100 elementos. Para no tener que realizar muchas peticiones y esperar antes de que la información sea usable, se ha procedido a realizar la espera de la información para los primeros 100 elementos, y el resto se realizará en background.

logedIn realiza la petición para obtener las primeras 100 organizaciones y pinta el esqueleto que tendrá la vista. Por otro lado, utilizamos una función recursiva *getBackgroundOrgs* que obtiene el resto de organizaciones existentes. A continuación, definimos y llamamos a *paginatedOrgs* que mostrará dentro del esqueleto las organizaciones deseadas.

3.6 Control de asincronía.

Una vez hemos llegado a este punto, estamos utilizando dos llamadas asíncronas dependientes entre ellas, la autenticación y la solicitud de organizaciones. Al no haber realizar un control de esta asincronía, el plug-in falla al ejecutarlo, ya que antes de confirmar que la autenticación es correcta intenta obtener las organizaciones.

Para el manejo de la sincronía, utilizamos Promesas, que tienen la siguiente estructura:

```
let promise = new Promise((resolve, reject) =>{/*peticiones deseadas*/})
promise.then((response)=>{/*ha acabado la petición*/})
```

Mediante el atributo *resolve* indicamos qué elementos deseamos obtener. Una vez ha sido resuelta la petición, se ejecutará el método *then*, donde incluiremos el código que seguirá a la petición. De esta forma, podemos pausar el flujo del programa hasta obtener la información necesaria. También usaremos las promesas para realizar las peticiones de organizaciones y de repositorios.

3.7 Repositorios.

Una vez resuelto el problema de la asincronía, procedemos a desarrollar la siguiente parte del proyecto, las listas de repositorios. La implementación de esta parte es casi igual a la de las organizaciones, y utilizaremos la misma estrategia para la solicitud y paginación de los elementos. La definición de esta vista se llevará a cabo en una nueva clase, *GhcrtaOrgs*.

Cuando hacemos click en una organización, llamaremos al método *getOrgRepos*, que tomando como atributos el nombre de la organización y el contenedor donde serán mostrado los repositorios, crea la estructura que tendrá la nueva vista y una elemento de tipo GhcrtaOrgs. A continuación, el método *getRepos* de GhcrtaOrgs será llamado, solicitando aquí la lista de repositorios y mostrando la información por pantalla. Al igual que con la lista de organizaciones, esta clase utiliza el método *paginatedRepos* para pintar la lista paginada.

La estructura de esta vista es ligeramente distinta a la anterior, estando dividida en 4 partes:

- Franja superior. Contiene al igual que en la vista anterior el input del buscador, pero además tiene un botón DIR que abrirá el Modal Auxiliar, y un número que indicará si se han producido errores al realizar alguna acción, como el clonado.

- Justo debajo de la franja superior, hay una pequeña sección que indica la organización en la que nos encontramos, y un botón que permite volver a la vista anterior.

- Franja central. Ocupa la mayor parte del panel que estamos usando y mostrará la lista de repositorios.

- Franja inferior. Al igual que en la vista de organizaciones, ésta franja contiene las opciones de paginado y además un botón que realizará la clonación de todos los repositorios de la lista.

3.8 Input de búsqueda.

La opción de búsqueda es bastante importante en el desarrollo del plug-in. En el caso de la vista de organizaciones sólo se usa para encontrar la organización deseada, pero en la vista de repositorios además permite utilizar este filtrado para la clonación de varios elementos de manera simultánea. Si no hemos filtrado, al utilizar el botón *Clone All* se clonarán todos los repositorios de la organización, pero si hemos realizado una búsqueda, se clonarán todos los elemento que coincidan con esta selección.

Los input de búsqueda contienen un evento onChange, que llamando a los métodos *searchOrgs* y *searchRepo* modifican el listado al que accede el método de escritura de las listas para mostrar solo los elementos coincidentes. En caso de que el input se vacíe, se retomará la lista original.

3.9 Clonación.

Una vez tenemos las vistas principales creadas, y funcionan correctamente, procedemos a implementar el proceso de clonación. Para acceder a los elementos de git hacemos uso del módulo simple-git. Existirán dos formas de clonar repositorios, la primera será haciendo click en la

opción de clonado en cada uno de los repositorios de la lista. La segunda será mediante el botón Clone All.

Para esta funcionalidad creamos una nueva clase, *GhcrtaRepo*, que contiene las funcionalidades de los repositorios. Al hacer click en uno de los repositorios, llamaremos a *cloneRepo*, que tomando como atributo el nombre del repositorio, creará una instancia de *GhcrtaRepo* que recibirá como atributos el nombre del repositorio y la organización a la que pertenece.

Una vez creado el elemento, el método *clone* comprueba la existencia del directorio de clonado y comprueba la existencia del repositorio en ese directorio. Al realizar esto, pueden pasar varias cosas:

- El directorio de clonado existe, pero no el repositorio. En este caso, se llama al método *clone* de simple-git y se realiza la clonación.

- Existen tanto el directorio de clonado como el repositorio. En esta situación se realizará un pull de la información del repositorio con el método *pull* de simple-git. En caso de que se produzca algún error, como errores de conflicto, será mostrado en la vista que lista los repositorios.

- No existe el directorio de clonado. Utilizando el módulo File System de node llamamos al método *mkdir*, que creará el directorio de clonado. En caso de que la dirección donde debe estar el directorio de clonado tampoco exista, se devolverá un error y se cancelará la clonación.

3.10 Directorio de trabajo.

Hasta esta fase, todas las funcionalidades habían sido desarrolladas utilizando directorio definidos de forma estática, pero una vez finalizadas, es el momento de definir un directorio de trabajo por defecto.

Para la elección de este directorio hay que tener en cuenta muchos factores, ya que hay que prestar atención a los distintos sistemas en los que se inicie, y tener en cuenta si dentro de Atom ya hay un directorio de trabajo iniciado. Por tanto se ha definido un orden de prioridad.

1. Directorio del proyecto. En Atom, se puede realizar la apertura de proyectos, de forma que solo aparezcan los ficheros incluidos dentro de un directorio en concreto. Siempre que este directorio exista, será tomado como el directorio por defecto para la clonación en el plug-in. Esto se realizará gracias a la función del objeto *atom* que el propio editor nos aporta.

   atom.project.getDirectories()

2. Escritorio del sistema. En caso de que no exista este directorio del proyecto, se tomará como directorio de trabajo el escritorio del sistema.

Una vez resuelto este conflicto, abordamos otro factor a tener en cuenta, el tipo de sistema operativo en el que trabajamos. El sistema de directorios en windows es distinto al de linux, por lo que debemos controlar esta situación. La librería *path* de node resuelve este problema por completo. Al usar direcciones, *path* es capaz de identificar el sistema en el que trabajamos y las maneja de forma correcta. Por tanto, para el movimiento entre directorios podemos utilizar su método:

path.join()

Por otro lado, mediante la vista Modal Auxiliar se incorporará la posibilidad de modificar la dirección de trabajo según lo desee.

3.11 Vista Modal Auxiliar.

Esta vista es un elemento div de tipo modal creado desde la ejecución del plug-in. Para esta vista, creamos una nueva clase a la que llamaremos GhcrtaDir, cuya función principal es la gestión de los directorios e informar a los usuarios de los errores producidos durante la ejecución del programa. Utiliza varios métodos estáticos mediante los cuales podremos realizar comprobaciones sobre la información que contiene de forma simple. Esta clase contiene nuestro directorio de trabajo, que es registrado la primera vez que entramos dentro de la vista de una organización.

Como se comentó anteriormente, este elemento se mostrará mediante el botón *DIR* situado en la vista del listado de repositorios y dispondrá de tres secciones:

- Sección central. Aquí se sitúan dos botones, uno para aceptar los cambios realizados en la vista, y otro para cancelarlos.

- Sección superior. En esta parte se mostrará un input con la dirección de trabajo actual. Al ser modificado y aceptados, al realizar una clonación se tomará como dirección de clonado la dirección indicada.

- Sección inferior. En esta sección aparecerán todos los errores producidos al intentar realizar una clonación o un PULL.

3.12 Control de errores.

Cuando utilizamos una función puede ocurrir que se produzcan errores que corten el curso normal del programa. En estos caso debemos interceptar estos fallos y definir un camino alternativo con el fin de que el plug-in siga funcionando correctamente.

- GhcrtaView. En esta clase podemos encontrar varios casos de este tipo de errores. En su constructor, haciendo uso del token de github y del módulo Octonode, creamos un objeto *client* para realizar cada uno de las peticiones a la API. Esta función puede fallar por varias situaciones:

    - Token inválido. El token puede ser inválido tanto por algún fallo a la hora de guardarlo, como porque haya sido eliminado en la página de Github. En este caso, se redirigirá al login con instrucciones para volver a solicitar el token.

    - Token inexistente. En caso de que no se haya guardado el token, o que haya sido eliminado del sistema de almacenamiento de Atom, se dirigirá al login para solicitar uno nuevo.

- GhcrtaClient. La comprobación de las credenciales es realizada en esta clase. En caso de que las credenciales sean erróneas, se devolverá a la vista de login con el mensaje :

Usuario o contraseña incorrecto

Por otro lado, también puede devolver error si ya existe un token en github para esta aplicación. En este caso, se mostrará información sobre cómo realizar una nueva petición de token.

- GhcrtaOrgs. Para realizar la solicitud de las organizaciones hemos de crear una instancia de *org* encontrada en Octonode. Al realizar la solicitud en la función *getRepos* haciendo uso de este objeto, puede ocurrir que devuelva un error. En este caso, mostraremos en pantalla el error devuelto por Github.

- GhcrtaRepo. Esta clase se encarga de realizar la clonación en el sistema, por lo que hay que controlar los errores tanto en la creación de directorios, como en la clonación y PULL de repositorios. En caso de que ocurra uno de estos errores, se enviarán a la vista Modal Auxiliar, donde será mostrado.

Capítulo 4

Guia de uso

A continuación, se realizará un pequeña guía sobre el uso de la aplicación desarrollada, abarcando desde la instalación del plug-in hasta el clonado de repositorios.

4.1 Instalación y ejecución.

La instalación del plug-in es un proceso rápido y sencillo. Haciendo uso de la página de atom, realizamos la búsqueda del paquete ghcrta. Una vez encontramos el paquete, podemos instalarlo mediante el botón *install.*

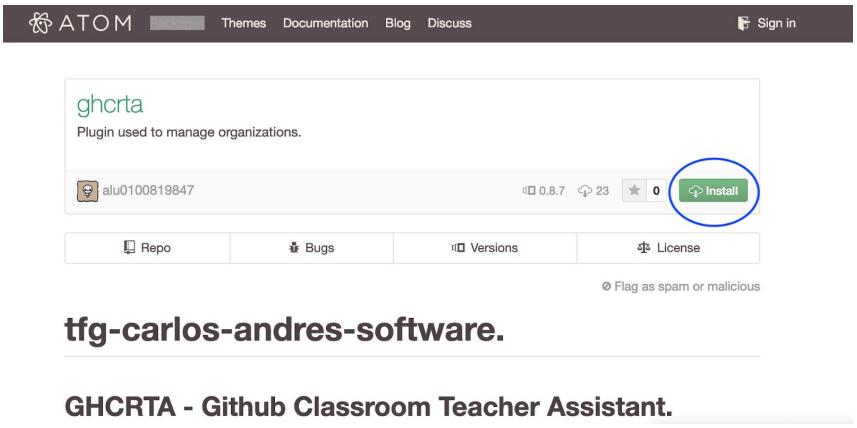


Figura 4.1: Dirección de descarga del paquete ghcrta.

Tras concluir la instalación, el paquete aparecerá en la ventana de paquetes de Atom. En caso de que no aparezca, puede ser necesario reiniciar el editor de texto.

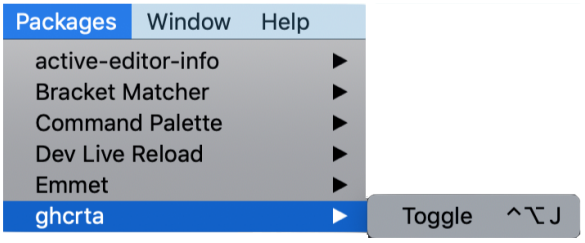


Figura 4.2: Menú de ejecución de paquetes de Atom.

Una vez en este menú, lo ejecutaremos haciendo click en Toggle.

4.2 Login.

La primera vista que tenemos al iniciar el plug-in es la de login. Para acceder, usaremos nuestras credenciales de github y pulsamos el botón *Login.*

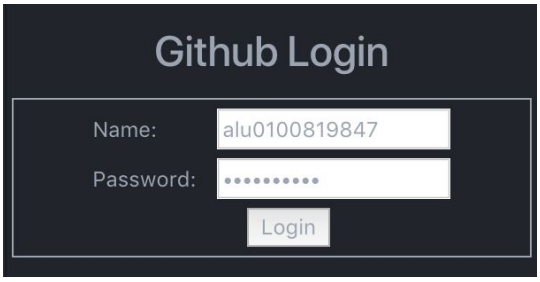


Figura 4.3: Login plug-in.

4.3 Organizaciones.

Si nuestras credenciales son correctas, nos redirigirá a la vista de organizaciones. En esta vista obtendremos una lista de todas nuestras organizaciones.

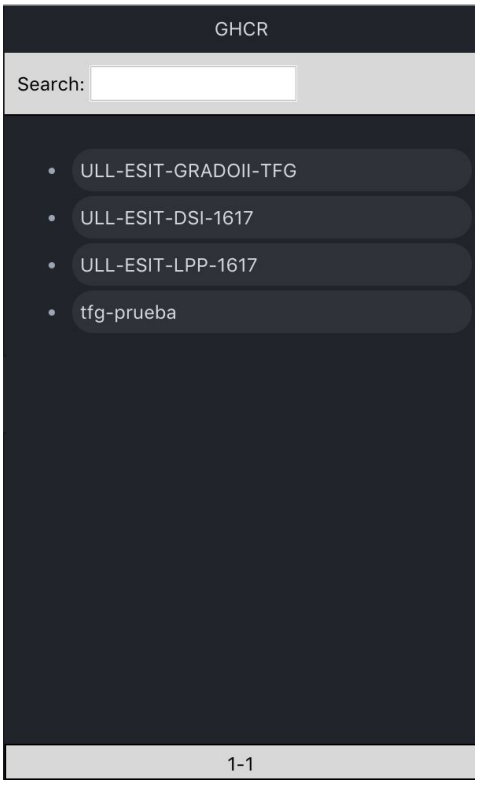


Figura 4.4: Lista de organizaciones.

Además, podemos filtrar nuestras organizaciones mediante el input Search.

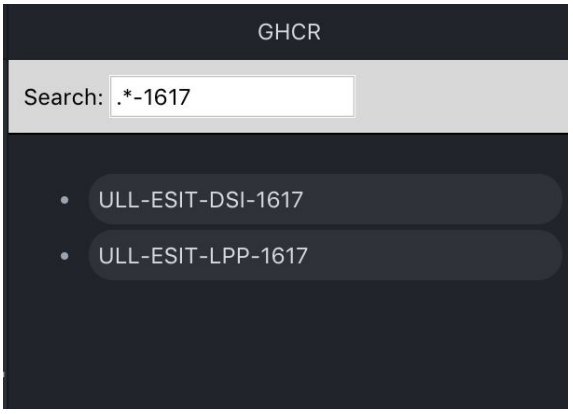


Figura 4.5: Búsqueda por expresión regular.

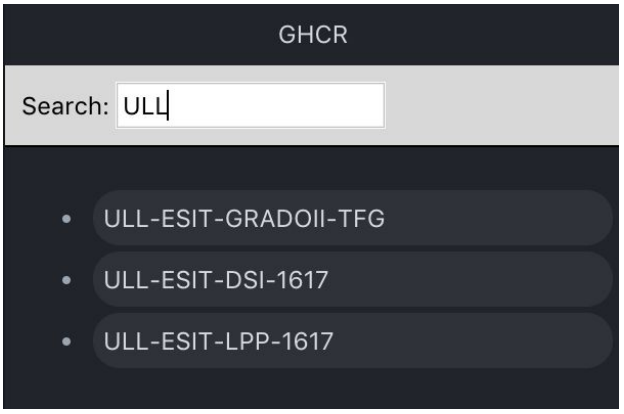


Figura 4.6: Búsqueda por nombre.

Para entrar en una de las organizaciones, hacemos click en su nombre.

4.4 Clonación.

Tras entrar en una organización, encontramos la lista de repositorios de la organización.

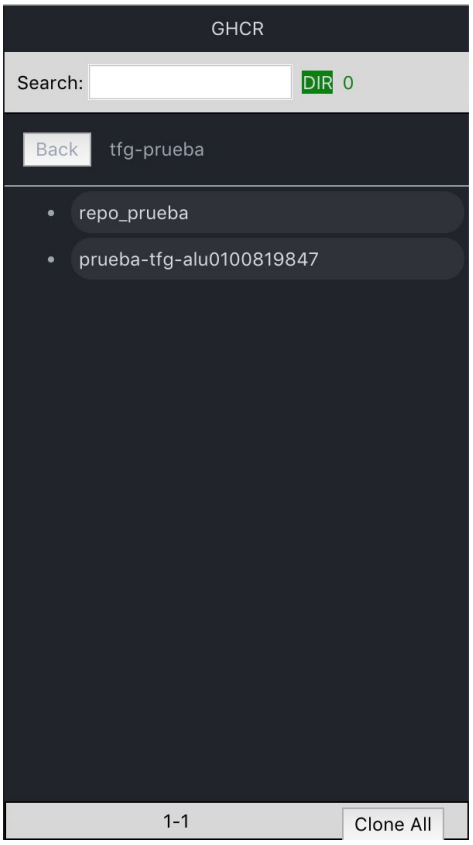


Figura 4.7: Lista de repositorios.

El primer paso es comprobar que la dirección de clonado es la deseada. Para ello, hacemos

click en el botón DIR situado en la sección superior.

Figura 4.8: Botón DIR.

Al pulsar el botón, se abrirá el Modal Auxiliar, donde comprobaremos la dirección del directorio de trabajo, y la modificaremos si es necesario.

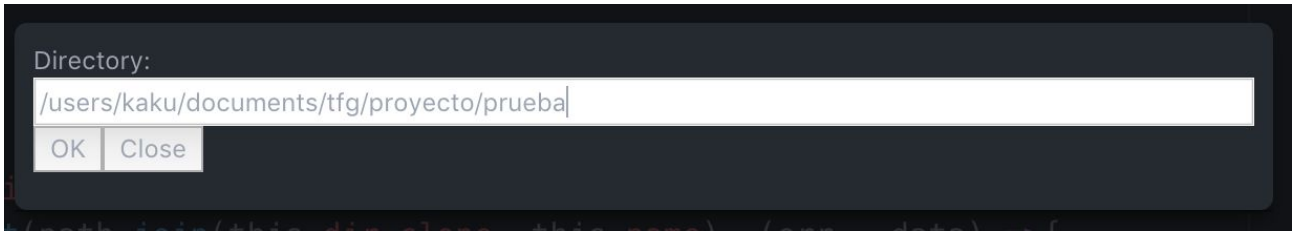


Figura 4.9: Input directorio de trabajo.

Una vez realizada esta comprobación, podemos comenzar a hacer clonaciones. Para ello tenemos 2 métodos:

- Clonar todos los repositorios listados mediante el botón *Clone All* situado en la franja inferior del panel.

Figura 4.10: Botón Clone All.

- Hacer click en el nombre del repositorio que deseas clonar.

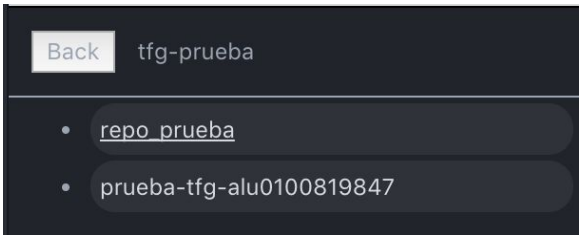


Figura 4.11: Clonación de 1 repositorio.

```
MacBook-Pro-de-Carlos:prueba kaku$ pwd
/Users/kaku/Documents/TFG/proyecto/prueba
MacBook-Pro-de-Carlos:prueba kaku$ ls -la
total 16
drwxr-xr-x  4 kaku  staff   128  9 mar 11:39 .
drwxr-xr-x  6 kaku  staff   192  9 mar 11:39 ..
-rw-r--r--@ 1 kaku  staff  6148  9 mar 11:40 .DS_Store
drwxr-xr-x  5 kaku  staff   160  9 mar 11:39 repo_prueba
```

Figura 4.12: Terminal. Repositorio Clonado.

En caso de que los repositorios seleccionados para clonar ya existan en el directorio de destino, se intentará realizar un pull de los datos.

4.5 Filtrado.

Se puede realizar el filtrado de los repositorios de la organización, tanto para buscar un repositorio en concreto, como para realizar la clonación de un conjunto reducido de ellos. En la siguiente ejemplo se intenta clonar todos los repositorios que contengan la cadena *carlos-andres.*

Como se puede ver en la siguiente imagen, estamos en una organización como múltiples páginas de repositorios, pero si añadimos la cadena *carlos-andres* al input de búsqueda, sólo aparecerán los repositorios deseados.

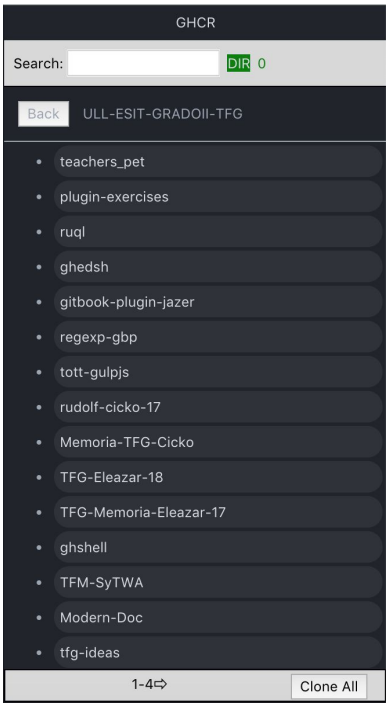


Figura 4.13: Lista de repositorios extensa.

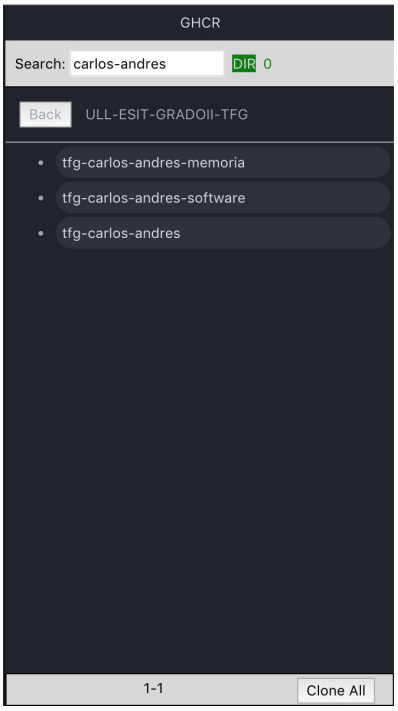


Figura 4.14: Lista de repositorios filtrada.

Si en este momento usamos el botón Clone All, solo se clonarán los repositorios filtrados.

```
MacBook-Pro-de-Carlos:prueba kaku$ pwd
/Users/kaku/Documents/TFG/proyecto/prueba
MacBook-Pro-de-Carlos:prueba kaku$ ls -la
total 16
drwxr-xr-x  6 kaku  staff   192  9 mar 11:53 .
drwxr-xr-x  6 kaku  staff   192  9 mar 11:53 ..
-rw-r--r--@ 1 kaku  staff  6148  9 mar 11:40 .DS_Store
drwxr-xr-x  9 kaku  staff   288  9 mar 11:53 tfg-carlos-andres
drwxr-xr-x  5 kaku  staff   160  9 mar 11:53 tfg-carlos-andres-memoria
drwxr-xr-x  7 kaku  staff   224  9 mar 11:53 tfg-carlos-andres-software
```

Figura 4.15: Terminal. Lista de repositorios filtrada clonada.

4.6 Errores.

En el proceso de manejo de repositorios, frecuentemente encontramos errores al realizar la actualización de la información o incluso para clonar el repositorio. Algunos de los errores más frecuentes son los siguientes:

- Conflicto al actualizar los ficheros.
- Cambios en el repositorio del equipo no commiteados.
- El directorio de clonación no existe.
- No está instalado git en el equipo.

Cualquier fallo que se produzca sobre este proceso aparecerá en el Modal Auxiliar, accesible desde el botón *DIR* dentro de una organización.

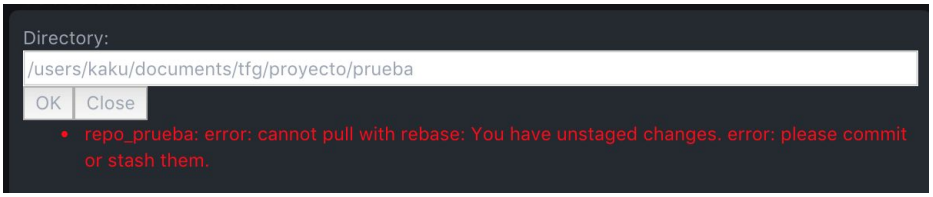


Figura 4.16: Lista de errores de clonado.

Capítulo 5 Conclusiones y líneas futuras

5.1 Conclusiones.

Las tecnologías de control de versiones actuales han tenido un gran crecimiento en los últimos años, siendo utilizadas en casi todos los proyectos. Además, presentan un amplio abanico de posibilidades para ser ampliadas. En este proyecto, hemos tratado el tema desde el ámbito educativo y hemos cogido un inconveniente que presentan estas herramientas y lo hemos analizado para encontrar una solución. Luego, hemos implementado ésa solución, concluyendo así con una herramienta alternativa para evitar ese problema.

Este proyecto consigue reducir el número de herramientas empleadas para el seguimiento de una organización en github a únicamente el editor de texto. Github Classroom se sustenta en estas organizaciones, por lo que podremos realizar la clonación de las asignaciones enteras en un tiempo muy reducido.

Hemos logrado responder al problema planteado cumpliendo todos los objetivos planteados en un principio, por lo tanto, podemos determinar que el proyecto ha sido finalizado.

5.2 Líneas Futuras.

En este proyecto, hemos abordado un problema detectado dentro del ámbito educativo, y el resultado es una herramienta que responde directamente a ello, permitiendo realizar la funcionalidad deseada de la manera más sencilla y rápida posible. Por tanto, este plug-in tiene muchos apartados en los que se puede avanzar, completando así una herramienta mucho más profunda y compleja.

- Repositorios. Actualmente, la única interacción que tenemos con los repositorios es la posibilidad de clonarlo o actualizarlo. Añadir una vista a partir de la cual se pueda obtener más información de cada uno de los repositorios sería un buen avance. Además, incluir datos como el estado del repositorio en el equipo sería una actualización muy interesante.

- Organizaciones. El uso de las organizaciones en este proyecto es muy importante, pero realmente solo son usadas para la obtención de los listados de repositorios. Incluir la

posibilidad de crear o borrar organizaciones, así como de obtener información adicional como el número de repositorios que contiene, son aspectos que se podrían mejorar.

- Directorio de trabajo. La forma de indicar el directorio de trabajo en el plug-in es a través de un input y un string. Una mejora significativa a este sistema sería añadir un explorador de directorios.

- Input. El sistema de input se encuentra muy limitado en Atom, impidiendo la modificación, copia o eliminación de los valores que contiene. En este proyecto, se han realizado funciones externas que lo controlan superficialmente, pero aún así, existen muchas mejoras que se le pueden aplicar.

- Velocidad. Durante todo el proyecto se ha intentado reducir los tiempos de carga debidos a las peticiones realizadas a Github, pero en ocasiones, algunas de las vistas tardan más de lo esperado en responder.

Capítulo 6

Summary and Conclusions

Current version control technologies have had a great growth during the last years, being used in almost all the projects. They have a wide range of possibilities to be expanded. In this project, we have addressed the issue from the educational area and we have taken a disadvantage of these tools to be analyzed and to find a solution. Then, we have implemented that solution, concluding with an alternative tool to resolve that problem.

This project achieves to reduce the number of tools used to manage a github organization to only one text editor. Github Classroom is based on these organizations, so we can do the cloning of the entire assignments in a very reduced time.

We have achieved to respond to the problem completing all the the objectives set out in the beginning, therefore, we can determine that the project has been completed.

Capítulo 7

Presupuesto

En este último capítulo, se presentará el presupuesto determinado para la realización de este proyecto.

7.1 Presupuesto

Actividad	Precio(€)/Horas	Horas
Análisis del problema.	12 €/h	10 h
Diseño.	15 €/h	25 h
Elección de herramientas.	15 €/h	5 h
Estudio de la documentación.	10 €/h	120 h
Implementación.	10 €/h	75 h
Total	2.520 €	235 h

Tabla 7.1: Presupuesto.

Bibliografía

[1] Atom Documentation. GitHub Inc. https://atom.io/docs, 2014.

[2] W3schools. Refsnes Data. https://www.w3schools.com/, 1999.

[3] NPM. npm Inc. https://www.npmjs.com, 2014.

[4] Octonode. Pavan Kumar Sunkara. https://www.npmjs.com/package/octonode, 2012.

[5] Simple-Git. Steveukx https://www.npmjs.com/package/simple-git, 2013.

[6] React. Facebook Inc. https://www.npmjs.com/package/react, 2013.

[7] React Dom. Facebook Inc. https://www.npmjs.com/package/react-dom, 2013.

[8] Github. GitHub Inc. https://github.com, 2010.

[9] Node.js Api.Node.js Foundation. https://nodejs.org/api, 2009.