Escuela Superior
de Ingeniería y Tecnología
Universidad de La Laguna

Trabajo de Fin de Grado

Herramientas para una navegación más eficiente: CVULL y Web Bookmarks

Tools for a more efficient navigation: CVULL and Web Bookmarks.

Alejandro González Alonso

La Laguna, 5 de junio de 2019

D. **Casiano Rodríguez León**, con N.I.F. 42.020.072-S profesor Titular de Universidad adscrito al Departamento de Nombre del Departamento de la Universidad de La Laguna, como tutor

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

Que la presente memoria titulada:

"*Herramientas para una navegación más eficiente: CVULL y Web Bookmarks*"

ha sido realizada bajo su dirección por D. **Alejandro González Alonso**, con N.I.F. 79.060.537-S.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 5 de junio de 2019

Agradecimientos

Después de un intenso período de 4 años, hoy es el día: el día en el que escribo este apartado para finalizar mi Trabajo de Fin de Grado. Ha sido un período tanto de desarrollo a nivel profesional como a nivel personal, y es por ello que quiero dar las gracias a:

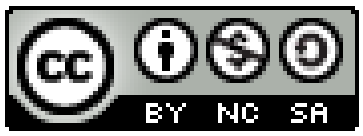
A mi pareja, por apoyarme en todo momento y darme fuerzas cuando lo necesitaba.

A mi tutor, Casiano Rodríguez León, por guiarme y ayudarme en el desarrollo de este Trabajo de Fin de Grado.

A mis padres y a mi hermano, por prestarme su ayuda siempre que lo necesitara.

Y a todos los compañeros de carrera con los que he tenido el lujo de compartir esta experiencia y de los que he aprendido infinidad de cosas.

Licencia



Resumen

El objetivo de este Trabajo de Fin de Grado consiste en aplicar los conocimientos adquiridos durante los años de formación en el Grado de Ingeniería Informática, para codificar dos herramientas que proporcione una mejora en la eficiencia de las labores de los docentes de las asignaturas de programación.

La primera herramienta desarrollada ha sido una extensión para el editor de código Visual Studio Code, que permite añadir marcadores dentro del entorno del editor. Con esta extensión se podrá añadir, editar, borrar, importar y exportar marcadores, y organizarlos en carpetas si así se desea.

La segunda herramienta permitirá a los docentes automatizar la obtensión de la información de sus alumno, y procesarla para dejarla en un formato compatible con la extensión desarrollada. Esta herramienta permitirá añadir las diferentes asignaturas, a través de su dirección web en el moodle, y a través del uso de la línea de comandos se descargará la información deseada.

Palabras clave: Visual Studio Code, TypeScript, Scraper, Crawler, Puppeteer, Marcador

Abstract

The objective of this Final Degree Project is to apply the knowledge acquired during the years of learning in the Degree in Computer Engineering, to codify two tools that provide an improvement in the efficiency of the tasks of the teachers of the programming subjects.

The first developed tool has been an extension for the Visual Studio Code code editor, which allows adding bookmarks within the editor environment. With this extension you can add, edit, delete, import and export bookmarks, and organize them in folders if desired.

The second tool will allow teachers to automate the retrieval of information from their students, and process it and leave it in a format compatible with the developed extension. This tool will allow you to add the different subjects, through their web address in the moodle, and through the use of the command line you will download the desired information.

Keywords: *Visual Studio Code, TypeScript, Scraper, Crawler, Puppeteer, Bookmark*

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

La programación ha cambiado en los últimos años gracias a la revolución de los IDEs que nos ayudan tanto a la sintaxis como nos simplifican de manera notable el proceso de depuración de nuestras aplicaciones. Sin embargo, desde no hace mucho han empezado a aparecer aplicaciones gratuitas y de código abierto que fomentan el uso de extensiones para ampliar las funcionalidades de los editores de código, como es el caso de Microsoft Visual Studio Code.

1.1. Antecedentes

Microsoft Visual Studio Code [6] es un editor de texto gratuito y de código abierto totalmente personalizable, de manera que los usuarios puedan adaptarlo a sus necesidades; desarrollado por Microsoft. La principal característica de este editor se encuentra en el amplio abanico de extensiones que nos proporcionan todo tipo de funcionalidades, así como la posibilidad de programar tu propia extensión si no existe alguna que satisfaga su necesidad.

Existen numerosas extensiones de Visual Studio Code que simplifica la labor del educador en la enseñanza práctica de la programación. Entre ellas podemos nombrar, GitLens[4], que nos proporciona una integración de Git dentro de Visual Studio Code; agregándonos funcionalidades como:

- Historial de Archivos
- Búsqueda directa de Commits
- Gestión de Ramas
- Comparación de Archivos
- Gestión de Conflictos
- Gestión de Stash

Un Web Crawler es un programa diseñado para explorar páginas Web de forma metódica y automática. Un ejemplo de Web Crawler es GoogleBot[12] encargado de indexar las nuevas páginas web en el motor de búsqueda de Google.

La plataforma Moodle[1] es un sistema de enseñanza open source, permitiendo su distribución e implementación de forma libre; diseñado para crear y gestionar espacios de aprendizaje online adaptados a las necesidades de profesores, estudiantes y administradores. Una de las implementaciones más cercanas es la del Campus Virtual de la Universidad de la Laguna[8], pensado para que los alumnos puedan gestionar las asignaturas desde una plataforma virtual.

1.2. Objetivos

En este proyecto se propone el análisis, diseño, implementación y pruebas de una extensión para Visual Studio Code que facilite al profesor la navegación web hacia los perfiles de sus alumnos. Además para automatizar la obtención de la lista de alumnos se propone una aplicación de tipo CLI (Command-line Interface) que genere un archivo con toda la información necesaria.

El objetivo de este Trabajo de Fin de Grado es el desarrollo de estas dos herramientas que, usadas en conjunto, permitirán ahorrar tiempo y esfuerzo al educador en el desarrollo de sus tareas, proporcionándole una forma de guardar una lista de marcadores dentro del editor de código, y que al clicar sobre ellos se abra el navegador por defecto en el marcador seleccionado; así como automatizar la obtención de los perfiles y libros de calificaciones de sus alumnos en las asignaturas del Campus Virtual.

1.3. Actividades a realizar

Para cumplir con los objetivos descritos, se han de realizar las siguientes actividades:

1. Revisión Bibliográfica para el desarrollo de la extensión
    - Estudio y análisis de la documentación disponible
2. Prototipado de la extensión
    - Elección nombre para la extensión
    - Mockup de la extensión
3. Codificación de la extensión
    - Visualización de los marcadores
    - Función de Edición
    - Función de Importación
    - Función de Exportación
4. Testing de la extensión
5. Revisión bibliográfica para el desarrollo de la segunda herramienta
    - Estudio y análisis de la documentación disponible
6. Codificación de la segunda herramienta
    - Función de configuración
    - Función de Participants
    - Función de Grades
    - Función de Fetch
7. Redacción de la memoria y obtención de resultados

La siguiente tabla representará el tiempo asignado para las diferentes tareas definidas:

Tareas	**Duración**
Tarea 1: Revisión Bibliográfica para el desarrollo de la extensión	15 días (Febrero)
Tarea 2: Prototipado de la extensión	7 días (Febrero)
Tarea 3: Codificación de la extensión	1 mes (Febrero - Marzo)
Tarea 4: Testing de la extensión	7 días (Marzo)
Tarea 5: Revisión bibliográfica para el desarrollo de la segunda herramienta	15 días (Abril)
Tarea 6: Codificación de la segunda herramienta	1 mes y 7 días (Abril - Mayo)
Tarea 7: Redacción de la memoria y obtensión de resultados	15 días (Mayo)

Tabla 1.1: Tabla de duración de las actividades

Capítulo 2

Desarrollo

A continuación se explicará el procedimiento realizado en el desarrollo de las herramientas, así como las diferentes tecnologías usadas.

2.1. Tecnologías y herramientas usadas

2.1.1. Visual Studio Code

Visual Studio Code[6] es un editor de programación multiplataforma desarrollo por Microsoft. Es un proyecto de software libre que se distribuye bajo la licencia MIT, aunque los ejecutables se distribuyen bajo una licencia gratuita no libre. Está basado en Electron[10] y está escrito en TypeScript junto con CSS.

Por defecto, Visual Studio Code es compatible con gran variedad de lenguajes de programación. Además trae un conjunto de características que facilitan el proceso de codificación como puede ser el uso de Snippets, Resaltado de sintaxis, Autocompletado, Debugger, etc. Todas estas características pueden ser extendidas a través de complementos, disponibles en un repositorio central.

Figura 2.1: Visual Studio Code

2.1.2. Node.js

Node.js[11] es un entorno JavaScript del lado del servidor, que sigue el paradigma de programación basado en eventos, y de programación asíncrona. Node.js ejecuta JavaScript utilizando el motor V8 desarrollado por Google, proporcionando un entorno de ejecución del lado del servidor que compila y ejecuta JavaScript a alta velocidad.

El objetivo de Node.js es generar una base para construir programas de red que sean escalables. Para ello, resuelve el problema de la gestión de conexiones a través de eventos dentro del proceso del motor de Node, evitando la ejecución de código bloqueante.

Figura 2.2: Node.js

2.1.3. NPM

NPM o **Node Package Manager**[14] es un gestor de paquetes que nos facilitará trabajar con Node, ya que centraliza los paquetes en un repositorio de acceso global, permitiendo obtener cualquier librería a través de una línea de código.

Todos los paquetes instalados serán registrados en el archivo de configuración de npm, package.json, que además nos proporcionará cierta información sobre el desarrollo de nuestro proyecto, por ejemplo, la versión, scripts para facilitar procesos de compilación, entre otros.

Figura 2.3: NPM

2.1.4. Electron y VSCODE API

Puesto que Visual Studio Code está basado en Electron, vamos a explicar que es. Electron[10] es un framework de código abierto desarrollado por GitHub que facilita el desarrollo de aplicaciones gráficas de escritorio usando componentes del lado del cliente y del servidor, utilizando Chromium como interfaz.

Figura 2.4: Electron

Debido a que Visual Studio Code no permite una conexión directa con la API de Electron, debemos usar la propia del Editor para poder extender las funcionalidades del mismo. De ésta forma, la API[7] de Visual Studio Code será la encargada de proporcionarnos la base para modificar la interfaz de Visual Studio Code.

2.1.5. Bootstrap

Bootstrap[9] es un framework desarrollado y liberado por Twitter que tiene como objetivo facilitar el diseño web. Permite crear de forma sencilla webs de diseño responsive, es decir que se adapte al tamaño de pantalla del dispositivo en el que se este viendo. Además proporciona una serie de componentes base que disponen de una apariencia atractiva y animaciones, que liberan la carga de trabajo al programador.

Figura 2.5: Bootstrap

2.1.6. TypeScript

TypeScript[5] es un lenguaje de programación de código abierto desarrollado por Microsoft. TypeScript, es llamado también Superset de JavaScript, y por tanto, convierte su código en JavaScript puro. TypeScript integra las últimas funcionalidades de las versiones de ECMAScript, manteniéndose siempre actualizado.

La principal ventaja de TypeScript frente a JavaScript es el tipado estático, que nos proporciona más control y estabilidad en nuestras aplicaciones.

Figura 2.6: TypeScript

2.1.7. Puppeteer

Puppeteer[13] es una librería de Node que proporciona una API de alto nivel para controlar una instancia de Chrome o Chromium en modo headless, es decir, sin interfaz gráfica; sobre el protocolo DevTools. De ésta forma, permite automatizar acciones sobre las páginas web en las que navegamos.

2.1.8. Open

Open[2] es una librería de Node que nos permite abrir cualquier tipo de archivo con el programa por defecto asignado en cualquier sistema operativo. De esta forma, se nos simplifica el proceso de apertura de archivos en las diferentes plataformas.

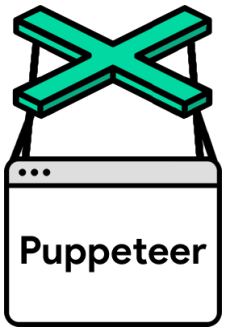


Figura 2.7: Puppeteer

2.1.9. Yargs

Yargs[3] es una librería de Node que nos permite parsear los parámetros recibidos en la línea de comandos. Además nos permite generar una estructura de subcomandos para que, a través de éstos, la utilización de la herramienta sea más intuitiva.

Figura 2.8: Yargs

2.2. Metodología de Trabajo

Este proyecto de Fin de Grado se ha desarrollado usando la metodología de desarrollo de software **XP** o **eXtreme Programming**. Esta metodología es una de las metodologías más utilizadas en cuanto al desarrollo de software y se diferencia de las demás dado que se centra en la retroalimentación entre el cliente (en este caso, el tutor académico) y el equipo de desarrollo (en este caso, el alumno).

Como cualquier otra metodología de desarrollo de software, su objetivo es mejorar la productividad de cualquier proyecto y reducir los riesgos y tiempos extras derivados de cambios en los requisitos del cliente. Para cumplir este objetivo, esta metodología define varios conceptos:

- **Planificaciones**

Se debe planificar diferentes plazos en el proyecto basándose en las exigencias del cliente, y en base a éstas se hará una estimación del coste y la dificultad y se definirán las prioridades.

- **Pruebas**

Continuamente se han de efectuar pruebas en base a los requisitos del cliente para comprobar que todo funciona correctamente. Éstas deben hacerse de forma periódica y automática, de forma que al final de cada planificación, el software esté probado y funcionando correctamente.

- **Simplicidad**

Al realizar reuniones con el cliente, se simplificará el desarrollo del proyecto, puesto que solo se desarrolla aquello que el cliente quiere, y además, éste participa en el.

Capítulo 3

Web Bookmarks

Web Bookmarks es una extensión para el editor de código Visual Studio Code. Ésta ofrece la posibilidad de guardar marcadores dentro del propio editor de código.

3.1. Logo, apariencia y diseño

El logo principal de la aplicación guarda un diseño sencillo y minimalista, esbozando la silueta de un navegador en negro y una banda en rojo haciendo referencia a los marcadores.

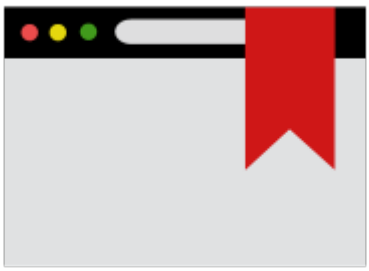


Figura 3.1: Web Bookmarks Logo

El diseño principal también guarda una apariencia sencilla, siguiendo el patrón **Material Design** gracias al framework de bootstrap.

Figura 3.2: Web Bookmarks Mockup

3.2. Funcionalidades

3.2.1. Generación de la vista

La vista de la WebView es generada dinámicamente mediante los marcadores leídos desde el archivo de configuración. Para ello, se han codificado las siguientes funciones:

- **loadJSON()**

A esta función se le pasa la ruta del archivo JSON a leer y, si el archivo se lee correctamente se retorna el objeto construido, y sino se genera un error que provocará una alerta dentro de Visual Studio Code.

```
function loadJSON(filePath: string) {
    try {
        let json = JSON.parse(fs.readFileSync(filePath, 'utf8'))
        return json;
    }
    catch (err) {
        throw new JSONLoadError(i18n.__('errorLoadJson'), filePath);
    }

}
```

Figura 3.3: Función loadJSON()

Figura 3.4: Alerta de error de lectura de JSON

- **html()**

Una vez leído los marcadores, se llama a la función **html()** que construirá la vista. A esta función se le pasan los marcadores, de la extensión, que servirá para poder localizar los diferentes assets del framework Bootstrap.

```
export function html(bookmarks: Folder, context: vscode.ExtensionContext): string {
    const bootstrapJsSrc = vscode.Uri.file(path.join(context.extensionPath, 'assets/js', 'bootstrap.min.js')).with({ scheme: 'vscode-resource' });
    const bootstrapCssSrc = vscode.Uri.file(path.join(context.extensionPath, 'assets/css', 'bootstrap.min.css')).with({ scheme: 'vscode-resource' });
    const jquerySrc = vscode.Uri.file(path.join(context.extensionPath, 'assets/js', 'jquery-3.3.1.slim.min.js')).with({ scheme: 'vscode-resource' });
    const popperSrc = vscode.Uri.file(path.join(context.extensionPath, 'assets/js', 'popper.min.js')).with({ scheme: 'vscode-resource' });

    let table: string = recursiveExplore(bookmarks);

    return `
    <!DOCTYPE html>
    <html>
        <head>
            <link rel="stylesheet" href="${bootstrapCssSrc}">
            <meta charset="UTF-8">
            <title>Web Bookmarks</title>
        </head>
        <body class="bg-dark">
            <div class="container">
                <div class="row">
                    <div class="col">
                        <div class="d-flex flex-row justify-content-center align-items-center">
                            <h1 class="text-white">Web Booksmarks</h1>
                            <button type="button" class="ml-2 btn btn-secondary" id="edit"> <svg
                                    xmlns="http://www.w3.org/2000/svg" width="24" height="24" viewBox="0 0 24 24">
                                    <path
                                        d="M3 17.25V21h3.75L17.81 9.94l-3.75-3.75L3 17.25zM20.71 7.04c.39-.39.39-1.02 0-1.41l-2.34-2.34c-.39-.39-1.02-.39-1.41 0l-1.83 1.83 3.75 3.75 1.83-1.83z"
                                        fill="white" />
                                    <path d="M0 0h24v24H0z" fill="none" /></svg> </button>
                        </div>
                    </div>
                </div>
                <div class="row">
                    <div class="col">
                        <div class="d-flex flex-row justify-content-center align-items-center">
                            <button type="button" class="btn btn-secondary" id="import">${i18n.__('impButton')}</button>
                            <button type="button" class="btn btn-secondary ml-2" id="export">${i18n.__('expButton')}</button>
                        </div>
                    </div>
                </div>
                <div class="row align-items-center justify-content-center mt-2">
                    <div class="col-sm-12 col-lg-10">
                        ${table}
                    </div>
                </div>
            </div>
            <script>
                window.onload = function(){
                    const vscode = acquireVsCodeApi();
                    let link = document.querySelectorAll(".bookmark");
                    for(let i of link){
                        i.addEventListener('click', function(){
                            vscode.postMessage({
                                command: 'open',
                                url: i.innerText
                            });
                        },false);
                    }
                    let edit = document.querySelector("#edit");
                    edit.addEventListener('click',function(){
                        vscode.postMessage({
                            command: 'edit'
                        },false);
                    });
                    let impbutton = document.querySelector("#import");
                    impbutton.addEventListener('click', function(){
                        vscode.postMessage({
                            command: 'import'
                        }, false);
                    });
                    let expbutton = document.querySelector("#export");
                    expbutton.addEventListener('click', function(){
                        vscode.postMessage({
                            command: 'export'
                        }, false);
                    });

                }
            </script>
            <script src="${jquerySrc}"></script>
            <script src="${popperSrc}"></script>
            <script src="${bootstrapJsSrc}"></script>
        </body>
    </html>`;
}
```

Figura 3.5: Función html()

Puesto que la organización en carpetas es recursiva, se ha desarrollado la función **recursiveExplore()** que permitirá recorrer recursivamente el objeto JSON de marcadores y generar elementos HTML para su representación.

```
let level = 0;
function recursiveExplore(data: Folder) {
    let table = "";
    for (const [key, value] of Object.entries(data)) {
        if (typeof (value as Bookmark) == 'string') {
            table += `
                <ul class="mt-2 mb-2 list-group list-group-horizontal text-white">
                    <li class="col-2 d-flex align-items-center justify-content-center list-group-item bg-secondary">${key}</li>
                    <li class="list-group-item bg-secondary flex-fill"><button class="btn btn-link bookmark text-warning">${value}</but
                </ul>`;
        }
        else {
            level++;
            let dummy = recursiveExplore(value as Folder);
            level--;
            let formattedKey = key.replace(' ', '-')
            table += `
                <div class="accordion mb-2" id="accordion-${level}-${formattedKey}">
                    <div class="card rounded bg-secondary text-white">
                        <div class="card-header" id="header-${level}-${formattedKey}">
                            <h3 style="cursor:pointer;" data-toggle="collapse" data-target="#folder-${level}-${formattedKey}" aria-expa
                                <svg xmlns="http://www.w3.org/2000/svg" width="24" height="24" viewBox="0 0 24 24"><path d="M11 5c-1.62
                            </h3>
                        </div>
                        <div id="folder-${level}-${formattedKey}" class="collapse" data-parent="#accordion-${level}-${formattedKey}">
                            <div class="card-body">
                                ${dummy}
                            </div>
                        </div>
                    </div>
                </div>`;
        }
    }
    return table;
}
```

Figura 3.6: Función recursiveExplore()

La función **html()**, gracias al string de elementos generados por **recursiveExplore()**, construirá la página html necesaria para que el componente WebView proporcionado por la API de Visual Studio Code. Para ello, se instancia un Panel de tipo WebView al que se le pasarán las opciones:

- **enableScripts**: Permite la ejecución de código JavaScript dentro de la WebView.
- **localResourceRoots**: Permite la carga de recursos, tales como archivos js, css, imágenes, etc.

```
panel = vscode.window.createWebviewPanel('webBookmarks', 'Web Bookmarks', vscode.ViewColumn.One, {
    enableScripts: true,
    localResourceRoots: [vscode.Uri.file(path.join(context.extensionPath, 'assets'))]
});

panel.webview.html = indexView.html(bookmarks, context);
```

Figura 3.7: Llamada a función html()

3.2.2. Refresco de la vista

Para que la vista esté siempre actualizada, se ha utilizado la función **OnChangeViewState** del panel instanciado. Está función es ejecutada cuando el panel WebView lanza un evento al obtener el focus en el editor, y produce una nueva lectura del archivo JSON de marcadores, y una nueva renderización de la página HTML del panel.

```
panel.onDidChangeViewState(e => {
    try {
        const panel = e.webviewPanel;
        bookmarks = loadJSON(bookmarksFile);
        refresh(context, panel, bookmarks);
    }
    catch (e) {
        if (e instanceof JSONLoadError) {
            vscode.window.showErrorMessage(e.message);
            vscode.window.showTextDocument(vscode.Uri.file(e.path));
        }
    }
}, null, context.subscriptions);
```

Figura 3.8: Función de refresco

3.2.3. Abrir

Para que la WebView sea capaz de ejecutar código y provocar cambios en el editor es necesario conectar la WebView con la API de Visual Studio Code. La comunicación entre la WebView y el editor se realiza a través de **mensajes Post**.

Para ello se utilizará la función **OnDidReceiveMessage()** del panel WebView, donde se procesará el mensaje Post y se ejecutará la acción codificada, que en este caso, es la apertura del navegador en la dirección web, trabajo que nos simplifica la Librería Open de Node.

```
panel.webview.onDidReceiveMessage(async message => {
    switch (message.command) {
        case 'open':
            vscode.window.showInformationMessage(i18n.__('open'));
            opn(message.url);
            return;
        case 'edit':
            vscode.window.showInformationMessage(i18n.__('edit'));
            vscode.window.showTextDocument(vscode.Uri.file(bookmarksFile));
            return;
        case 'import': {
            try {
                let file: vscode.Uri[] | undefined = await vscode.window.showOpenDialog({
                    canSelectFiles: true, canSelectFolders: false, canSelectMany: false, filters: {
                        "JSON (.json)": ["json"]
                    }
                });
                let content = loadJSON((file as vscode.Uri[])[0].fsPath)
                writeJSON(content, bookmarksFile);
                bookmarks = content;
                refresh(context, panel as vscode.WebviewPanel, bookmarks);
                vscode.window.showInformationMessage(i18n.__('import'));
                return;
            }
            catch (e) {
                if (e instanceof JSONLoadError) {
                    vscode.window.showErrorMessage(e.message);
                    vscode.window.showTextDocument(vscode.Uri.file(e.path));
                }
                return;
            }
        }
        case 'export': {
            let file: vscode.Uri | undefined = await vscode.window.showSaveDialog({
                filters: {
                    "JSON (.json)": ["json"]
                }
            })
            writeJSON(bookmarks, (file as vscode.Uri).fsPath)
            vscode.window.showInformationMessage(i18n.__('export'));
            return;
        }
    }
}, undefined, context.subscriptions)
```

Figura 3.9: Función OnDidReceiveMessage()

Además, cuando se genera el html se añade un event de click a los hipervínculos, que provocarán el envío del mensaje hacia el editor.

```
window.onload = function(){
    const vscode = acquireVsCodeApi();
    let link = document.querySelectorAll(".bookmark");
    for(let i of link){
        i.addEventListener('click', function(){
            vscode.postMessage({
                command: 'open',
                url: i.innerText
            });
        },false);
    }
```

Figura 3.10: Generación del Mensaje Post para Abrir Marcador

```
case 'open':
    vscode.window.showInformationMessage(i18n.__('open'));
    opn(message.url);
    return;
```

Figura 3.11: Captación y ejecución del comando Abrir Marcador

3.2.4. Importar

Al igual que en la funcionalidad de abrir, para importar se genera un mensaje post que será enviado al editor, provocando que se abra el diálogo de seleccionar un archivo, y una vez seleccionado, se actualizará el archivo de marcadores y se renderizará la vista con los marcadores importados.

```
let impbutton = document.querySelector("#import");
impbutton.addEventListener('click', function(){
    vscode.postMessage({
        command: 'import'
    }, false);
});
```

Figura 3.12: Generación del Mensaje Post para Importar Marcadores

```
case 'import': {
    try {
        let file: vscode.Uri[] | undefined = await vscode.window.showOpenDialog({
            canSelectFiles: true, canSelectFolders: false, canSelectMany: false, filters: {
                "JSON (.json)": ["json"]
            }
        });
        let content = loadJSON((file as vscode.Uri[])[0].fsPath)
        writeJSON(content, bookmarksFile);
        bookmarks = content;
        refresh(context, panel as vscode.WebviewPanel, bookmarks);
        vscode.window.showInformationMessage(i18n.__('import'));
        return;
    }
    catch (e) {
        if (e instanceof JSONLoadError) {
            vscode.window.showErrorMessage(e.message);
            vscode.window.showTextDocument(vscode.Uri.file(e.path));
        }
        return;
    }
}
```

Figura 3.13: Captación y ejecución del comando Importar Marcadores

3.2.5. Exportar

Para la funcionalidad de exportar también se genera un mensaje post que es recogido por el editor y procesado en la función OnDidReceiveMessage(). En este caso, se abrirá un diálogo para seleccionar la ruta y el nombre del archivo, y se escribirá en disco el objeto JSON gracias a la función **writeJSON()** y al uso de la librería estándar de Node: **fs**.

```
let expbutton = document.querySelector("#export");
expbutton.addEventListener('click', function(){
    vscode.postMessage({
        command: 'export'
    }, false);
});
```

Figura 3.14: Generación del Mensaje Post para Exportar Marcadores

```
case 'export': {
    let file: vscode.Uri | undefined = await vscode.window.showSaveDialog({
        filters: {
            "JSON (.json)": ["json"]
        }
    })
    writeJSON(bookmarks, (file as vscode.Uri).fsPath)
    vscode.window.showInformationMessage(i18n.__('export'));
    return;
}
```

Figura 3.15: Captación y ejecución del comando Exportar Marcadores

```
function writeJSON(data: Object, filePath: string) {
    fs.writeFileSync(filePath, JSON.stringify(data));
}
```

Figura 3.16: Función writeJSON()

3.2.6. Editar

Por último para la función editar, se generará su mensaje post correspondiente, y tras recibirlo, se abrirá en el editor una pestaña en el archivo de marcadores.

```
let edit = document.querySelector("#edit");
edit.addEventListener('click',function(){
    vscode.postMessage({
        command: 'edit'
    },false);
});
```

Figura 3.17: Generación del Mensaje Post para Editar Marcadores

```
case 'edit':
    vscode.window.showInformationMessage(i18n.__('edit'));
    vscode.window.showTextDocument(vscode.Uri.file(bookmarksFile));
    return;
```

Figura 3.18: Captación y ejecución del comando Editar Marcadores

Capítulo 4

CVULL

CVULL es un programa de interfaz de comandos que permite al usuario obtener y procesar la información de los alumnos matriculados en las asignaturas del campus virtual. Esta herramienta complementa a la extensión desarrollada, puesto que la información obtenida será compatible con ésta.

4.1. Logo, apariencia y diseño

El logo principal de ésta herramienta guarda un diseño minimalista que, como en la extensión, busca explicar la funcionalidad de la misma a través del logo. En él se puede observar la silueta de un navegador junto con el color de fondo de la página siendo el color corporativo de la ULL y una lupa que hace referencia a la extracción de información.

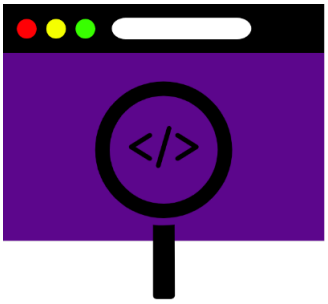


Figura 4.1: CVULL Logo

En relación al diseño de la herramienta, no se dispone de un diseño visual al ser una aplicación de interfaz de comandos.

4.2. Funcionalidades

En este apartado se expondrán las diferentes funcionalidades de la herramienta desarrollada, así como las diferentes funciones codificadas para su ejecución.

4.2.1. Configuración

En primer lugar, la herramienta CVULL se debe configurar para poder hacer uso de ella. Para ello, se han desarrollado varios comandos:

- **Establecer Usuario**

Lo primero de todo, nuestra herramienta en algún momento tendrá que autenticarse dentro del campus virtual, por lo tanto, se ha desarrollado un comando para configurar éste usuario de forma que no sea necesario pedir repetidamente éste en ejecuciones repetidas.

Esta función simplemente comprueba la existencia del archivo de configuración. Si este existe, es cargado y transformado a un objeto JavaScript, al que se le añade el atributo *username*. Si no existiese, se instancia un objeto JavaScript al que se le agregará ésta propiedad. Por último, el archivo de configuración es sobrescrito con la información del objeto JavaScript.

```
export const handler = (argv: UsernameArguments) => {
    try {
        const cfgFile = path.join(os.homedir(), '.cvull', 'config.json');
        const cfgDir = path.dirname(cfgFile);
        const cfgFound: boolean = fs.existsSync(cfgFile);
        const cfgDirFound: boolean = fs.existsSync(cfgDir);

        let cfg: ConfigOptions = {};

        if (cfgFound) cfg = JSON.parse(fs.readFileSync(cfgFile, 'utf-8'));

        cfg.username = argv.username;

        if (!cfgDirFound) fs.mkdirSync(cfgDir, { recursive: true });
        fs.writeFileSync(cfgFile, JSON.stringify(cfg));
        console.log('Username Added Succesfully!');
    } catch (err) {
        console.log(err.message);
    }
};
```

Figura 4.2: Función de establecer usuario

- **Añadir y Borrar Asignaturas**

Además de configurar el usuario, por cuestiones de usabilidad, se ha permitido al usuario añadir y borrar de la configuración las asignaturas de las que se quiere obtener la información de los alumnos.

En la función de añadir, se comprueba si existe el archivo de configuración, para importarlo si es necesario. Si no existe, se generará un nuevo objeto JavaScript, donde se añadirá la información de la asignatura.

Una vez generado o importado el objeto donde se almacenará la información, se añadirá al objeto el nombre de la asignatura como propiedad, y como valor su url, recibiendo estos datos en la línea de comandos.

```
export const handler = (argv: AddClassroomArguments) => {
    try {
        const cfgFile = path.join(os.homedir(), '.cvull', 'config.json');
        const cfgDir = path.dirname(cfgFile);
        const cfgFound: boolean = fs.existsSync(cfgFile);
        const cfgDirFound: boolean = fs.existsSync(cfgDir);

        let cfg: ConfigOptions = {};

        if (cfgFound) cfg = JSON.parse(fs.readFileSync(cfgFile, 'utf-8'));

        cfg[argv.name] = argv.url;

        if (!cfgDirFound) fs.mkdirSync(cfgDir, { recursive: true });
        fs.writeFileSync(cfgFile, JSON.stringify(cfg));
        console.log('Classroom Added Succesfully!');
    } catch (err) {
        console.log(err.message);
    }
};
```

Figura 4.3: Función de añadir asignatura

En la función de borrado, al igual que en la de añadir, se comprueba la existencia del archivo de configuración. Y se elimina del objeto la propiedad con el nombre de la asignatura y se sobrescribe el archivo.

```
export const handler = (argv: RemoveClassroomArguments) => {
    try {
        const cfgFile = path.join(os.homedir(), '.cvull', 'config.json');
        const cfgDir = path.dirname(cfgFile);
        const cfgFound: boolean = fs.existsSync(cfgFile);
        const cfgDirFound: boolean = fs.existsSync(cfgDir);

        let cfg: ConfigOptions = {};

        if (cfgFound) {
            cfg = JSON.parse(fs.readFileSync(cfgFile, 'utf-8'));
            delete cfg[argv.name];
        }

        if (!cfgDirFound) fs.mkdirSync(cfgDir, { recursive: true });
        fs.writeFileSync(cfgFile, JSON.stringify(cfg));
        console.log('Classroom Removed Succesfully!');
    } catch (err) {
        console.log(err.message);
    }
};
```

Figura 4.4: Función de borrar asignatura

- **Visualizar Configuración**

Para mejorar la usabilidad, también se ha codificado un comando para mostrar la configuración de la herramienta. Esta función lee el archivo de configuración, lo formatea y lo imprime por la salida estándar.

```
export const handler = (argv: Argv) => {
    const cfgFile = path.join(os.homedir(), '.cvull', 'config.json');

    let cfg: ConfigOptions = {};
    const cfgFound: boolean = fs.existsSync(cfgFile);

    try {
        if (cfgFound) cfg = JSON.parse(fs.readFileSync(cfgFile, 'utf-8'));
        else throw new Error('No configuration found!');

        const { username, ...classroomList } = cfg;
        const user = `\n\tUser: ${username || 'Username not configured!'}`;
        const classroom =
            Object.entries(classroomList).length !== 0
                ? `\n\tClassrooms: {${Object.keys(classroomList).map(
                      name => `\n\t\t${name}: ${classroomList[name]}`,
                  )}\n\t}`
                : '\n\tClassrooms: No Classrooms defined!';

        console.log('Crawler Configuration: %s %s', user, classroom);
    } catch (err) {
        console.log(err.message);
    }
};
```

Figura 4.5: Función de visualizar configuración

4.2.2. Login

Las siguientes funcionalidades estarán encapsuladas en una clase Crawler, que será la encargada de gestionar todo el procedimiento de la obtención de la información.

La primera función de desarrollada ha sido la de autenticación, puesto que será necesario para poder acceder al campus virtual. Esta función realizará click en el botón *1*, rellenará input *2* con el usuario configurado y la contraseña es requerida por consola, completando el input *3*. Por último, se expandirá el menú de navegación pulsando el botón *4*.

```
private login = async (username: string) => {
    ((await (this.page as puppeteer.Page).$(
        'a[href="index.php?authCAS=CAS"]',
    )) as puppeteer.ElementHandle).click();
    await (this.page as puppeteer.Page).waitForNavigation();
    const password = prompt.question(`Password for ${username}: `, {
        hideEchoBack: true,
    });
    const inputUsername = (await (this.page as puppeteer.Page).$(
        '#username',
    )) as puppeteer.ElementHandle;
    const inputPassword = (await (this.page as puppeteer.Page).$(
        '#password',
    )) as puppeteer.ElementHandle;
    await inputUsername.type(username);
    await inputPassword.type(password);
    await inputPassword.press('Enter');
    await (this.page as puppeteer.Page).waitForNavigation();
    await this.checkSideBar();
};
```

Figura 4.6: Función de autenticación en el campus virtual

Figura 4.7: Pantalla de login 1

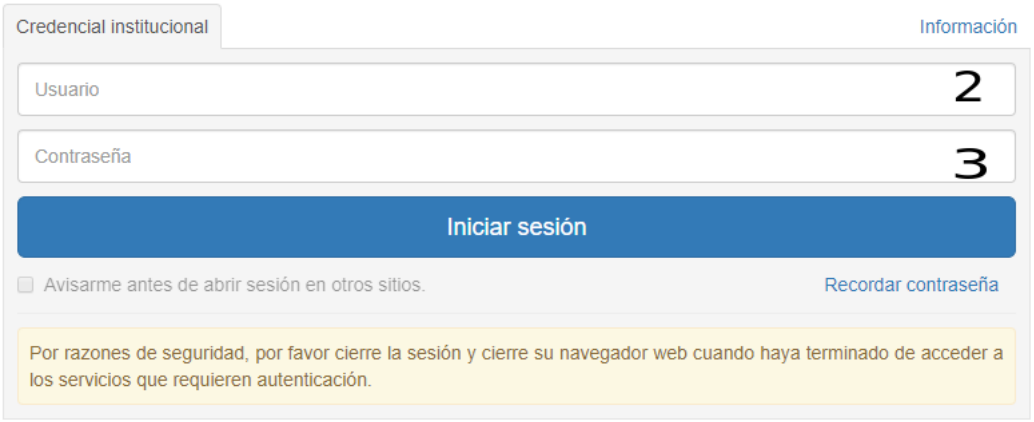


Figura 4.8: Pantalla de login 2

Figura 4.9: Pantalla de login 3

4.2.3. Participants

Para la funcionalidad de *participants*, se han codificado dos funciones: una pública que será la que se pueda llamar desde el objeto y una privada que es la encargada de la obtención de la información de los alumnos.

La función privada accederá al listado de participantes navegando hasta él desde el menú de navegación. Además formateará los datos para dejarlos con la estructura idónea para la importación en la extensión.

```
private getParticipants = async (): Promise<Bookmarks> => {
    ((await (this.page as puppeteer.Page).$(
        'a[data-key="participants"]',
    )) as puppeteer.ElementHandle).click();
    await (this.page as puppeteer.Page).waitForNavigation();
    ((await (this.page as puppeteer.Page).$(
        '#showall>a',
    )) as puppeteer.ElementHandle).click();
    await (this.page as puppeteer.Page).waitForNavigation();
    const users = await (this.page as puppeteer.Page).$$eval(
        '#participants>tbody>tr:not(.emptyrow)',
        (nodes: Element[]): string[][] =>
            nodes.map(
                (n: Element): string[] => [
                    n.children[1].textContent as string,

                    n.children[1].children[0].getAttribute(
                        'href',
                    ) as string,
                ],
            ),
    );
    const bookmarks = users.reduce(
        (result: Bookmarks, i: string[]): Bookmarks => {
            const a = result;
            const [name, type] = i;
            a[name] = type;
            return a;
        },
        {},
    );

    return bookmarks;
};
```

Figura 4.10: Función Privada de Participants

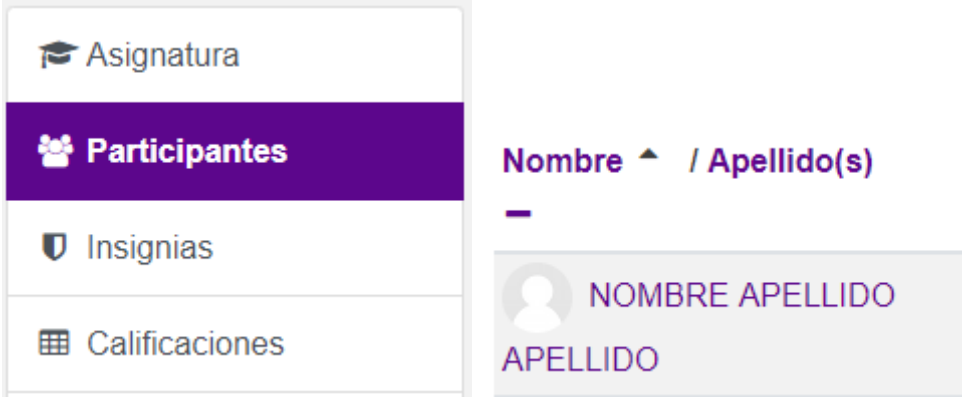


Figura 4.11: Pantalla de Participantes

La función pública recorrerá la lista de asignaturas de la que se quiere obtener la información y escribirá la información en el archivo de salida.

```
public participants = async (username: string, classrooms: Classrooms) => {
    const result: Bookmarks = {};
    this.page = await (this.browser as puppeteer.Browser).newPage();

    /* eslint-disable */
    const keys = Object.keys(classrooms);
    for (let name of keys) {
        await (this.page as puppeteer.Page).goto(classrooms[name], {
            waitUntil: 'load',
        });
        if ((this.page as puppeteer.Page).url() !== classrooms[name])
            await this.login(username);

        console.log(`fetching participants ${name} classroom ... `);

        result[name] = {};

        (result[name] as Bookmarks)[
            'Participantes'
        ] = await this.getParticipants();
    }
    /* eslint-enable */
    await (this.browser as puppeteer.Browser).close();
    return result;
};
```

Figura 4.12: Función Pública de Participants

4.2.4. Grades

Para la funcionalidad de *grades*, se ha codificado de forma similar a participants, dividiéndola en dos funciones: una pública y una privada.

La función privada accederá al libro de calificaciones de cada alumno, mediante una tabla que contiene toda la información, ubicada en la pestaña de calificaciones. Además formateará los datos para dejarlos con la estructura idónea para la importación en la extensión.

```
private getGrades = async (): Promise<Bookmarks> => {
    ((await (this.page as puppeteer.Page).$(
        'a[data-key="grades"]',
    )) as puppeteer.ElementHandle).click();
    await (this.page as puppeteer.Page).waitForNavigation();
    const users = await (this.page as puppeteer.Page).$$eval(
        '#user-grades>tbody>tr[data-uid]',
        (nodes: Element[]): string[][] =>
            nodes.map(
                (n: Element): string[] => [
                    n.children[0].textContent as string,

                    n.children[1].children[0].getAttribute(
                        'href',
                    ) as string,
                ],
            ),
    );
    const bookmarks = users.reduce(
        (result: Bookmarks, i: string[]): Bookmarks => {
            const a = result;
            const [name, type] = i;
            a[name] = type;
            return a;
        },
        {},
    );
    return bookmarks;
};
```

Figura 4.13: Función Privada de Grades

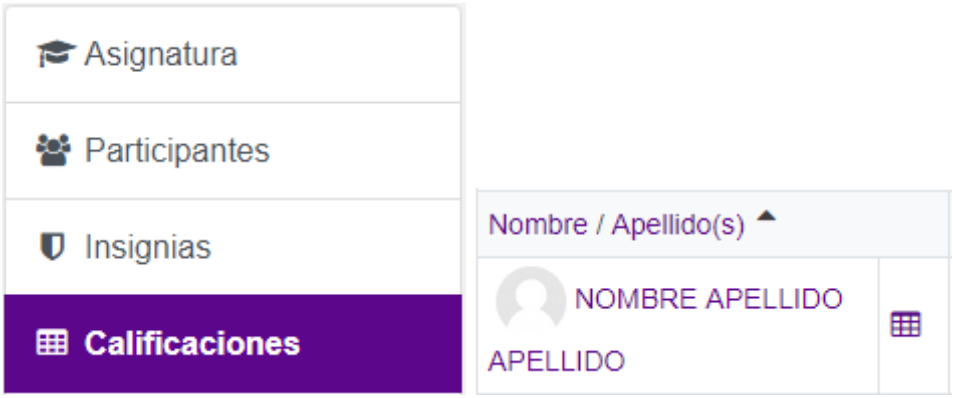


Figura 4.14: Pantallas de Grades

La función pública recorrerá la lista de asignaturas de la que se quiere obtener la información y escribirá la información en el archivo de salida.

```
public grades = async (username: string, classrooms: Classrooms) => {
    const result: Bookmarks = {};
    this.page = await (this.browser as puppeteer.Browser).newPage();

    /* eslint-disable */
    const keys = Object.keys(classrooms);
    for (let name of keys) {
        await (this.page as puppeteer.Page).goto(classrooms[name], {
            waitUntil: 'load',
        });
        if ((this.page as puppeteer.Page).url() !== classrooms[name])
            await this.login(username);

        console.log(`fetching grades ${name} classroom ... `);

        result[name] = {};

        (result[name] as Bookmarks)[
            'Calificaciones'
        ] = await this.getGrades();
    }
    /* eslint-enable */
    await (this.browser as puppeteer.Browser).close();
    return result;
};
```

Figura 4.15: Función Pública de Grades

4.2.5. Fetch

La función fetch se encargará de recoger toda la información, es decir, los perfiles de los alumnos y el libro de calificaciones de cada uno. Para ello, se recorren las asignaturas solicitadas y se llaman a las funciones privadas de **getParticipants()** y **getGrades()**.

```
public fetch = async (username: string, classrooms: Classrooms) => {
    const result: Bookmarks = {};
    this.page = await (this.browser as puppeteer.Browser).newPage();

    /* eslint-disable */
    const keys = Object.keys(classrooms);
    for (let name of keys) {
        await (this.page as puppeteer.Page).goto(classrooms[name], {
            waitUntil: 'load',
        });
        if ((this.page as puppeteer.Page).url() !== classrooms[name])
            await this.login(username);

        console.log(`fetching ${name} classroom ... `);

        result[name] = {};

        (result[name] as Bookmarks)[
            'Participantes'
        ] = await this.getParticipants();

        (result[name] as Bookmarks)[
            'Calificaciones'
        ] = await this.getGrades();
    }
    /* eslint-enable */
    await (this.browser as puppeteer.Browser).close();
    return result;
};
```

Figura 4.16: Función de Fetch

Capítulo 5

Conclusiones y Líneas Futuras

En conclusión, estas dos herramientas pueden ser usadas por los docentes de prácticas para ahorrar tiempo en el proceso de calificación de sus alumnos.

En términos generales, se han cumplido con los objetivos principales de este proyecto. Sin embargo, por falta de tiempo, se podrían realizar ciertas mejoras que incrementarían el rendimiento de las herramientas y la legibilidad del código.

En el futuro se buscará aplicar estas mejoras de forma que la calidad global de las herramientas aumenten, y su modificación se simplifique.

Capítulo 6

Summary and Conclusions

In conclusion, these two tools can be used by teachers of programming related subjects to save time in the process of grading their students.

In general terms, the main objectives of this project have been met. However, due to lack of time, certain improvements could be made that would increase the performance of the developed tools and the readability of the code.

In the future we will seek to apply these improvements so that the overall quality of the tools increases, and simplify the updating process.

Capítulo 7

Presupuesto

Puesto que en este Trabajo de Fin de Grado no es necesaria ninguna infraestructura en donde se despliguen las herramientas desarrolladas, el coste total de este proyecto constaría del sueldo al programador durante el tiempo de desarrollo.

Teniendo en cuenta el sueldo promedio para programadores en España es de **1628** euros al mes, y el desarrollo de este proyecto ha durado 4 meses, la cifra total de dicho proyecto alcanzaría los **6514** euros.

En este presupuesto no se tiene en cuenta el mantenimiento de la aplicación puesto que dependería del tiempo que se quiera contratar el mantenimiento. Sin embargo, la segunda herramienta desarrollada es muy dependiente de él, puesto que cualquier cambio estructural en la página del campus virtual provocaría que ésta dejara de funcionar.

Capítulo 8

Guía de Uso

8.1. Instalación

A continuación se mostraran los pasos a seguir para la instalación de las herramientas.

8.1.1. Web Bookmarks

Para instalar la extensión en Visual Studio Code, debemos ir a la sección de extensiones en Visual Studio Code, y buscar Web Bookmarks:
Una vez dentro, pulsamos el botón de instalar y listo, se nos instalará la extensión.

8.1.2. CVULL

Para instalar la herramienta CVULL, se ejecutará el comando:

```
npm install -g ull-crawler
```

8.2. Configuración

8.2.1. CVULL

8.2.1.1. User

Para configurar el usuario que usara la aplicación para conectarse al campus virtual se ejecutará el siguiente comando:

```
cvull config user <username>
```

donde <username> será el usuario del campus virtual.

8.2.1.2. Classrooms

Para configurar las diferentes asignaturas de las que se quieren extraer la información de los alumnos, se usan los siguientes comandos:

8.2.1.2.1. Add El comando Add añadirá al archivo de configuración una asignatura. Se ejecuta de la siguiente forma:

```
cvull config classroom add <name> <url>
```

donde <name> es el nombre que queramos asignarle a la asignatura y <url> la dirección web de la asignatura.

8.2.1.2.2. Remove El comando Remove borrará del archivo de configuración la asignatura expuesta. Se ejecuta de la siguiente forma:

```
cvull config classroom remove <name>
```

donde <name> es el nombre de la asignatura configurada.

8.2.1.3. List

Este comando mostrará la configuración actual del crawler. Se ejecuta de la siguiente forma:

```
cvull config list
```

8.3. Ejecución

En esta sección se mostrará como se deben ejecutar las herramientas.

8.3.1. Web Bookmarks

Para ejecutar la extensión en Visual Studio Code, se podrá hacer de dos maneras:

- Haciendo uso del shortcut:
  - **Windows y Linux:** Ctrl + Shift + L
  - **Mac:** CMD + Shift + L
- A través de la paleta de comandos:
  1. Abrir la paleta de comandos.
  2. Buscar *Open Web Bookmarks* y pulsar.

8.3.2. CVULL

En esta sección explicaremos como ejecutar las diferentes funciones de la herramienta.

8.3.2.1. Fetch

Para ejecutar la función Fetch de la herramienta se ejecutará el siguiente comando:

```
cvull fetch [<classroom>...]
```

Si no se proporciona <classroom>, se obtendrá la información de todas las asignaturas configuradas. Para lanzar fetch con más de una asignatura, se tendrá que listar todas las asignaturas separándolas por un espacio:

```
cvull fetch <classroom1> <classroom2>
```

8.3.2.2. Participants

Para ejecutar la función *Participants* de la herramienta se ejecutará el siguiente comando:

```
cvull participants [<classroom>...]
```

Si no se proporciona <classroom>, se obtendrán los participantes de todas las asignaturas configuradas. Para lanzar *participants* con más de una asignatura, se tendrá que listar todas las asignaturas separándolas por un espacio:

```
cvull participants <classroom1> <classroom2>
```

8.3.2.3. Grades

Para ejecutar la función *Grades* de la herramienta se ejecutará el siguiente comando:

```
cvull grades [<classroom>...]
```

Si no se proporciona <classroom>, se obtendrán los libros de calificaciones de los participantes de todas las asignaturas configuradas. Para lanzar Grades con más de una asignatura, se tendrá que listar todas las asignaturas separándolas por un espacio:

```
cvull grades <classroom1> <classroom2>
```

Bibliografía

[1] Moddle. https://moodle.org.

[2] Open. https://www.npmjs.com/package/open.

[3] Yargs. https://www.npmjs.com/package/yargs/.

[4] Eric Amodio. Gitlens. https://github.com/eamodio/vscode-gitlens.

[5] Microsoft Corporation. Typescript. https://www.typescriptlang.org.

[6] Microsoft Corporation. Visual studio code. https://code.visualstudio.com.

[7] Microsoft Corporation. Visual studio code api. https://code.visualstudio.com/api/references/vscode-api.

[8] Universidad de la Laguna. Campus virtual ull. https://campusvirtual.ull.es.

[9] Facebook Inc. Bootstrap. https://getbootstrap.com/.

[10] GitHub Inc. Electron. https://electronjs.org/.

[11] Node.js Developers Joyent. Nodejs. https://nodejs.org/es/.

[12] Google LLC. Googlebot.

[13] Google LLC. Puppeteer. https://www.npmjs.com/package/puppeteer/.

[14] npm Inc. Npm. https://nodejs.org/es/.