Escuela Superior
de Ingeniería y Tecnología
Universidad de La Laguna

Trabajo de Fin de Grado

Grado en Ingeniería Informática

Extensión para el editor Atom. Web Bookmarks

Extension for the Atom editor. Web Bookmarks

Raúl Martín Morales

La Laguna, 7 de Junio de 2019

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

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

Que la presente memoria titulada:

“Extensión para el editor Atom. Web Bookmarks”

ha sido realizada bajo su dirección por D. **Raúl Martín Morales**,

con N.I.F. 79.082.293-J.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 7 de junio de 2019

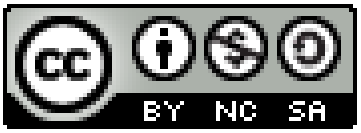
Agradecimientos

En primer lugar, dar las gracias a mis amigos y familiares que siempre han estado animándome y apoyándome durante el transcurso de la carrera y durante toda mi vida.

Por otro lado agradecer a todos los profesores y compañeros con los que he compartido conocimientos y han contribuido en mi formación. También a José Ginés Venazco Cabrera y Elena Robayna Duque, fundadores de creaTáctil S.L., empresa donde realicé las prácticas externas del Grado en Ingeniería Informática y donde se me trató desde el primer día como un miembro más del equipo, teniendo una primera experiencia laboral muy satisfactoria.

Por último, agradecer a Casiano Rodríguez León, tutor en este proyecto, por su ayuda durante todo el desarrollo de este Trabajo de Fin de Grado y por ser un gran profesor y persona.

Licencia



Resumen

El objetivo de este Trabajo de Fin de Grado ha sido aplicar los conocimientos adquiridos durante el desarrollo del Grado en Ingeniería Informática, para elaborar una extensión para un programa, que lo dote de nuevas funcionalidades con el fin de dar soporte en sus tareas a los profesores y alumnos de asignaturas de programación.

Para ello, se ha creado un plug-in para el editor de código Atom, que permita crear una lista de marcadores web dentro del propio editor.

En este plug-in se podrán crear marcadores y carpetas, para tener una mayor organización, pudiendo crear subcarpetas. También se podrán editar y borrar los elementos creados mediante un fichero json donde se guardan los marcadores.

Palabras clave: Atom, marcador, plug-in, extensión, paquete

Abstract

The objective of this Final Degree Project has been to apply the knowledge acquired during the development of the Degree in Computer Engineering, to develop an extension for a program that provides new functionalities in order to support teachers and students in their tasks of programming subjects.

For this, a plug-in has been created for the Atom code editor, which allows creating a list of web bookmarks within the editor itself.

This plug-in allows the creation of bookmarks that can be grouped into folders. You can also edit and delete the elements created using a json file where the bookmarks are saved.

Keywords: Atom, bookmark, plug-in, extension, package

Índice general

Índice de figuras

Índice de tablas

Índice de códigos

Capítulo 1 Introducción

1.1 Antecedentes y estado actual del arte

Atom [1] es un editor de código fuente abierto para macOS, Linux y Windows con soporte para plug-ins escritos en Node.js [2]. Ha sido construido en Electron [3], un entorno de desarrollo de aplicaciones Desktop que utiliza tecnologías web.

Actualmente existen varias extensiones de Atom que facilitan la labor del educador en la enseñanza práctica de la programación, tanto en la fase de diseño de la práctica, como en su resolución. Citemos por ejemplo las relacionadas con git y github como GitHub package [4], el cual permite gestionar el control de versiones directamente desde Atom. También nombrar Remote-atom [5] que permite usar Atom sobre un directorio remoto sobre SSH y por lo tanto poder acceder desde una instancia de Atom que se ejecuta en una máquina de la Escuela al sistema de archivos de las máquinas virtuales del servicio iaas.ull.es.

1.2 Objetivos

En este proyecto nos proponemos el análisis, diseño, implementación y pruebas de una extensión para Atom, que facilite la labor del educador en la enseñanza práctica de la programación, tanto en la fase de diseño de la práctica, como en su resolución, así como a los alumnos en el desarrollo de éstas.

El objetivo principal de este Trabajo de Fin de Grado, como se comentó en el resumen de esta memoria, es el desarrollo de este plug-in, que nace debido a que a la hora de estar trabajando con el editor Atom en ciertos proyectos o trabajos, es habitual tener que visitar alguna página web, lo que requiere cambiar de programas y pestañas para abrirla.

Por tanto, este plug-in pretende ahorrar tiempo y esfuerzo a la hora de realizar estas labores, de manera que dentro del propio editor, se proporcione una lista de marcadores que, al hacer clic ellos, abran el navegador por defecto en la página web seleccionada.

1.3 Actividades a realizar

Para llevar a cabo los objetivos descritos es la sección anterior, se han definido las actividades que deben ser realizadas:

T1. Revisión bibliográfica
  T1.1. Estudio de tutoriales para la creación de plug-ins para Atom
T2. Diseño e implementación de prototipo
  T2.1. Elegir el nombre del plug-in
  T2.2. Crear vista del plug-in
T3. Implementación del sistema
  T3.1. Abrir enlaces en el navegador
  T3.2. Crear marcadores
  T3.3. Crear carpetas

T3.4. Crear subcarpetas
T3.5. Editar y/o eliminar elementos creados

T4. Validación y evaluación del sistema

T5. Redacción de la memoria y difusión de los resultados

Tarea	Duración
T1: Revisión bibliográfica	15 días (febrero)
T2: Diseño e implementación de prototipo	15 días (febrero)
T3: Implementación del sistema	1 mes y medio (marzo, abril)
T4: Validación y evaluación del sistema	1 mes y medio (abril, mayo)
T5: Redacción de la memoria y difusión de los resultados	1 mes (mayo, junio)

Tabla 1.1: Plan de trabajo

Capítulo 2 Desarrollo

En el capítulo anterior se ha introducido de manera simple y sencilla el tema sobre el que está enfocado este proyecto, situándose en torno a antecedentes y el estada actual del tema, así como los objetivos y tareas a realizar para desarrollar el proyecto.

En este capítulo se va a describir el desarrollo del proyecto, en cuanto a las tecnologías, herramientas y metodología de trabajo utilizada de manera extendida.

2.1 Tecnologías y herramientas usadas

2.1.1 Electron

Electron es una biblioteca de código abierto desarrollada por GitHub [6] para construir aplicaciones de escritorio multiplataforma con HTML [7], CSS [8], y JavaScript [9]. Electron logra esto combinando Chromium y Node.js en un mismo ejecutable, así las aplicaciones pueden empaquetarse para Mac, Windows y Linux.

Electron comenzó en 2013 como el entorno de trabajo sobre el que se construiría Atom, del que se hablará después.

Desde entonces se ha convertido en una herramienta popular utilizada por desarrolladores de código abierto, nuevas empresas y compañías establecidas. Para hacerse una idea, algunas de las aplicaciones de escritorio que han sido desarrolladas usando Electron son Discord, GitHub Desktop, Twitch, WhatsApp, entre otras.

Figura 2.1: Electron

2.1.2 Atom

Atom es un editor de código de fuente abierta con soporte para plug-ins escritos en Node.js y control de versiones Git integrado, desarrollado por GitHub. Está escrito en CoffeeScript [10] y Less [11] y también puede ser utilizado como un entorno de desarrollo integrado (IDE).

Utilizando los complementos predeterminados, Atom es compatible con muchos lenguajes, pero puede añadir soporte para otros lenguajes de programación mediante el sistema de paquetes, así como mejorar el soporte para los paquetes existentes mediante mejoras como intérpretes, debuggers o pipelines que conecten software de terceros a Atom.

Se puede obtener funcionalidades extra gracias al “package manager”, con el que se puede instalar y desinstalar fácilmente multitud de paquetes, ya que están en continuo desarrollo.

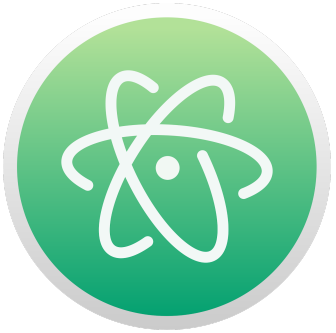


Figura 2.2: Atom

2.1.3 NodeJs

Node es un intérprete Javascript del lado del servidor que cambia la noción de cómo debería trabajar un servidor. Su meta es permitir a un programador construir aplicaciones altamente escalables y escribir código que maneje decenas de miles de conexiones simultáneamente en sólo una máquina física.

Está basado en el lenguaje de programación ECMAScript [12], es asíncrono, con I/O de datos en una arquitectura orientada a eventos y basado en el motor V8 de Google.

Figura 2.3: NodeJs

2.1.4 NPM

Node Package Manager [13] es el gestor de paquetes por defecto de Node.js. Gracias a él, se puede tener casi cualquier librería disponible a tan solo una línea de comando de distancia. Npm va a ayudar a administrar nuestros módulos, distribuir paquetes y agregar dependencias de una manera sencilla.

Este gestor de paquetes, va a utilizar el fichero “package.json”, un fichero que guarda cierta información como el nombre del proyecto, la versión, el repositorio o las dependencias que se van a instalar con este gestor de paquetes, entre otras cosas.

Figura 2.4: Npm

2.1.5 Javascript

JavaScript es un lenguaje ligero e interpretado, orientado a objetos con funciones de primera clase. Es un lenguaje script multi-paradigma, basado en prototipos, dinámico, soporta estilos de programación funcional, orientada a objetos e imperativa.

Se utiliza principalmente en su forma del lado del cliente, implementado como parte de un navegador web permitiendo mejoras en la interfaz de usuario y páginas web dinámicas aunque existe una forma de Javascript del lado del servidor.

El estándar de JavaScript es ECMAScript. desde el 2012, todos los navegadores modernos soportan completamente ECMAScript 5.1. Los navegadores más antiguos soportan por lo menos ECMAScript 3. En 2015 ECMAScript International publicó la sexta versión de ECMAScript, la cual es oficialmente llamada ECMAScript 2015, y fue inicialmente nombrada como ECMAScript 6 o ES6. Desde entonces, los estándares ECMAScript están en ciclos de lanzamiento anuales.

Figura 2.5: JavaScript

2.1.6 HTML

HTML, que significa Hyper Text Markup Language, es posiblemente el más famoso lenguaje de marcado. Es el elemento de construcción más básico de una página web y se usa para crear y representar visualmente una página web. Determina el contenido de la página, pero no su funcionalidad. Esta tecnología suele venir acompañada de otras tecnologías para describir la apariencia/presentación de una página web, como CSS, o su funcionalidad, como JavaScript.

El primer documento formal con la descripción de HTML se publicó en 1991 con el nombre de HTML Tag, pero no fue hasta 1995 que se publica el primer estándar oficial, HTML 2.0. La versión HTML 3.2 publicada en 1997 es la primera recomendación de HTML publicada por el W3C [14]. La última especificación oficial de HTML se publica en 1999, denominada HTML 4.01. El Web Hypertext Application Technology Working Group (WHATWG) ha desarrollado el estándar HTML5 y ha sido adoptado por el World Wide Web Consortium (W3C). El 28 de mayo de 2019 el Consorcio WWW declinó a favor de WHATWG en cuanto al control de las futuras normas acerca de HTML y DOM.

Figura 2.6: HTML

2.1.7 LESS CSS

Cascading Style Sheets (CSS) u hojas de estilo, llamadas comúnmente, es el lenguaje utilizada para describir la presentación de documentos HTML o XML. CSS describe como debe ser renderizado el elemento estructurado en pantalla.

CSS es uno de los lenguajes base de la Open Web y posee una especificación estandarizada por parte del W3C. Desarrollado en niveles, CSS1 es ahora obsoleto, CSS2.1 es una recomendación y CSS3, ahora dividido en módulos más pequeños, está progresando en camino al estándar.

En este proyecto se utiliza LESS CSS que es una ampliación a las hojas de estilo CSS, pero a diferencia de éstas, funcionan como un lenguaje de programación, permitiendo el uso de variables, funciones, operaciones aritméticas, entre otras cosas, para acelerar y enriquecer los estilos en un sitio web.

La principal diferencia entre LESS y otros precompiladores CSS es que LESS permite la compilación en tiempo real vía less.js por el navegador. LESS se puede ejecutar en el lado del cliente y del lado del servidor, o se puede compilar en CSS sin formato.

Figura 2.7: CSS

Figura 2.8: LESS

2.2 Metodología de Trabajo

La metodología de trabajo va a describir el procedimiento que se ha llevado a cabo para realizar el proyecto. Es decir, de qué manera se ha llegado a las conclusiones que presenta nuestro trabajo.

Existen varias metodologías ya definidas y se podría decir que entre ellas la eXtreme Programming o XP [15] es la que se ha usado en este proyecto.

XP es una metodología ágil y flexible, que pone el énfasis en la retroalimentación continua entre cliente y equipo de desarrollo y es idónea para proyectos con requisitos imprecisos o muy cambiantes. Por ello, se ajusta al proyecto, ya que a lo largo de éste, se han hecho reuniones entre el tutor y el alumno para revisar los avances, añadir nuevas

funcionalidades, hacer cambios en la forma de realizar cierta función del plug-in, etc.

Esta metodología se basa en varios factores:

- Integración continua

  Durante todo el desarrollo de la extensión, las distintas funcionalidades de ésta, se realizan de manera progresiva. Haciendo modificaciones a medida que se van desarrollando todas, ya que al añadir nuevas funcionalidades ha habido que hacer cambios en otras.

- Simplicidad

  El hecho de simplificar el diseño, agiliza el desarrollo y facilita el mantenimiento. Pero siempre prestando atención a que se cumplan las necesidades del proyecto.

- Retroalimentación

  El desarrollo del plug-in se ha hecho de manera que cada vez que se añadían funcionalidades nuevas o se corregían ciertos bugs se comunicaba y se hacía una reunión con el tutor, que hacía de sujeto de pruebas. Por lo que en cada reunión recibía un feedback.

- Rápida respuesta a cambios

  Como se comentó anteriormente, el proyecto ha estado sujeto a cambios continuos, ya sean por problemas en el funcionamiento correcto del plug-in, por el diseño o por un cambio de idea en el funcionamiento.

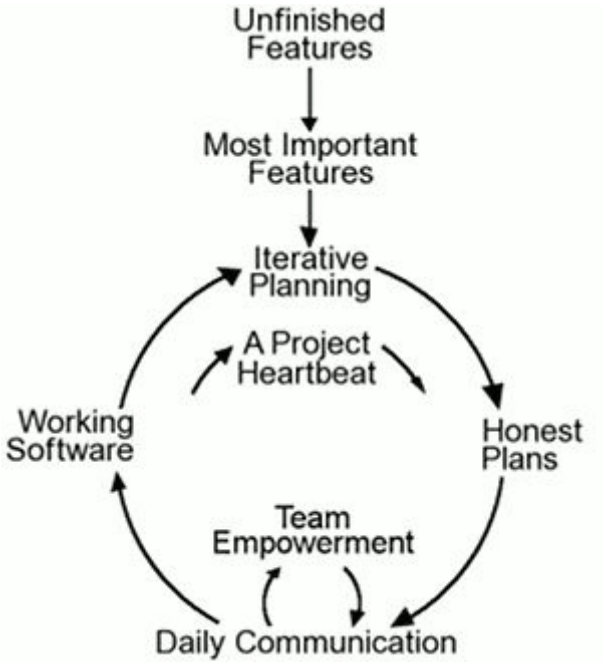


Figura 2.9: Extreme Programming

Capítulo 3 Web Bookmarks

Este capítulo describe el plug-in desarrollado.

Web Bookmarks [29] es una extensión creada para el editor de código Atom, que pretende servir de ayuda en el desarrollo de las labores académicas tanto de los profesores como de los alumnos.

Como se ha comentado en varias puntos de este documentos, Web Bookmarks va a permitir crear una lista de marcadores web que estarán integrados en el propio editor, para tener un fácil y rápido acceso.

3.1 Apariencia y diseño

Al activar el plug-in se abrirá un tab en el lateral derecho del editor, del mismo estilo al que lo hacen los de Git y GitHub, esto es debido a que resulta una manera cómoda y accesible su uso.

Web Bookmarks está dividido en dos secciones:

- La primera contiene un pequeño título seguido de los formularios para crear los marcadores y carpetas, además de un pequeño botón en la esquina superior derecha que se usará para refrescar la vista.

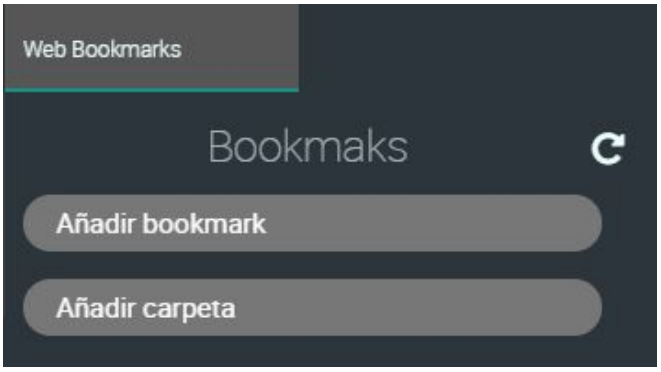


Figura 3.1: Primera sección, formularios

Ambos formularios están ocultos y se utiliza un botón para cada uno para desplegarlos. Esto permite tener un rápido acceso a los formularios en todo momento, evitando crear cuadros emergentes u otras posibles soluciones que seguramente sean más tediosas. Pero también se consigue una mejora en cuanto a la organización y disposición del contenido, ya que en un principio los formularios estaban siempre a la vista pero esto desaprovecha mucho espacio.

Además, para evitar que ocupe mucho espacio, cuando ya hay un formulario abierto y se pulsa en el botón del otro para desplegarlo, el primero que ya estaba abierto se contraerá.

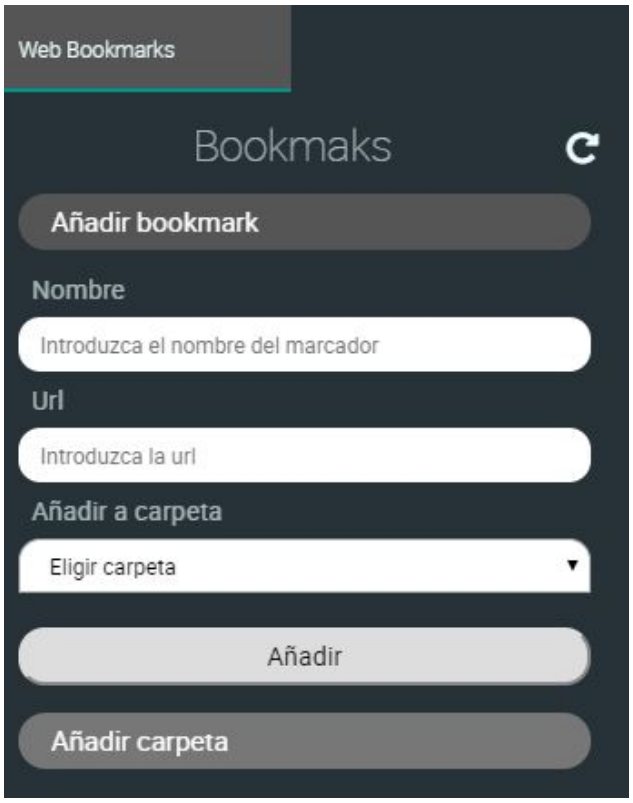


Figura 3.2: Formulario marcadores

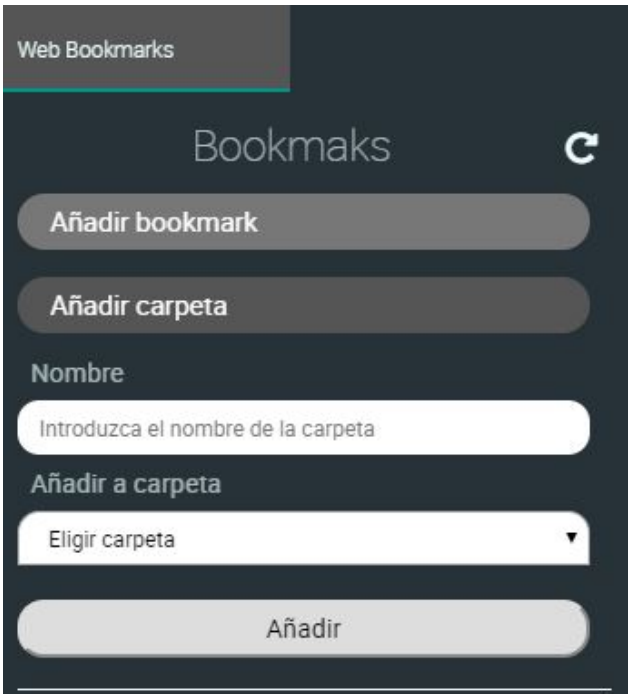


Figura 3.3: Formulario carpetas

- La segunda sección tendrá la lista de marcadores y carpetas que se han creado. Para agrupar los marcadores en las carpetas se utiliza el mismo procedimiento que con los formularios, se utiliza un botón que al hacer clic despliegue o contraiga los elementos que contiene.

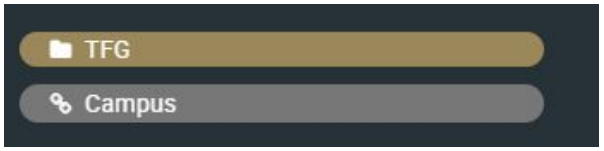


Figura 3.4: Ejemplo marcadores contraído

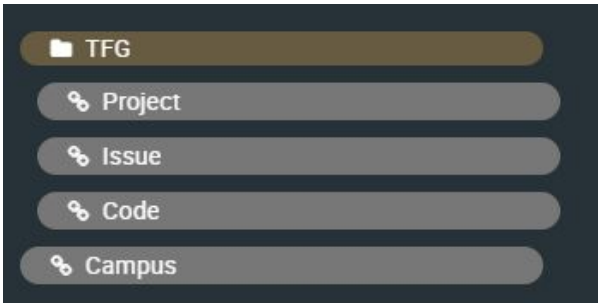


Figura 3.5: Ejemplo marcadores desplegado

En el caso de que la lista de marcadores y carpetas no quepa en el espacio visible, aparecerá una barra lateral con la que hacer scroll.

Para implementar estos botones expandibles se utiliza javascript y css. Al crearlos se añade también a continuación un *div* que, por defecto, tendrá el atributo *display: none*, para que no sea visible. En este div estarán los campos del formulario en el caso de la primera sección y en el caso de la segunda, tendrá marcadores y/o carpetas. En cada botón se añade un *event listener [16]* que intercambia el valor de la propiedad *display* entre "*block*" y "*none*" según el valor que tenga antes.

expandBtns() es la función que va a realizar todo este proceso. Primero guarda en una variable todos los botones expansibles, a continuación guarda en otra variable una función que alterna la propiedad display y también se encarga de evitar que se expandan los dos formularios a la vez. Por último se recorre la primera variable que tiene guardada todos los botones y les añade un *event listener* que al hacer clic, ejecute la función guardada anteriormente. En la *Figura 3.6*, que se muestra a continuación se puede ver el código.

```
  expandBtns() {
    var coll = document.getElementsByClassName("collapsible");
    var listener = function() {
      this.classList.toggle("active");
      var content = this.nextElementSibling;
      if (content.style.display === "block") {
        content.style.display = "none";
      } else {
        content.style.display = "block";
      }
      if((this.id == "btn-form-bookmark") && (this.classList.contains("active")) &&
(document.getElementById("btn-form-folder").classList.contains("active"))){
        document.getElementById("btn-form-folder").classList.toggle("active");
        var content = document.getElementById("btn-form-folder").nextElementSibling;
        if (content.style.display === "block") {
          content.style.display = "none";
        }
      }
      else if((this.id == "btn-form-folder") && (this.classList.contains("active"))
&& (document.getElementById("btn-form-bookmark").classList.contains("active"))){
        document.getElementById("btn-form-bookmark").classList.toggle("active");
        var content =
document.getElementById("btn-form-bookmark").nextElementSibling;
        if (content.style.display === "block") {
          content.style.display = "none";
        }
      }
    };

    for (var i = 0; i < coll.length; i++) {
      coll[i].addEventListener("click", listener, false);
    }
  }
```

Código 3.1: Expandir botones

Al expandir los formularios se ocupa más espacio y para evitar que se puedan ocultar algunos marcadores o carpetas, cada vez que se hace clic en uno de ellos se calcula la altura que ocupa la primera sección donde se encuentran éstos y se redimensiona la segunda sección para que en caso de sobrepasar el límite, se active el scroll vertical, que se ha definido como automático.

3.2 Crear marcadores y carpetas

Para crear los marcadores y carpetas se utilizan los dos formularios mostrados en el punto anterior.

Los marcadores requieren rellenar los campos *Nombre*, *Url* y de manera opcional, mediante un *select*, elegir si se desea añadir el marcador a una carpeta existente. Respecto a las carpetas hay que rellenar los campos *Nombre* y como en el caso anterior, si se quiere añadir a una carpeta existente para crear una subcarpeta, elegir la opción en el *select*.

newBookmark() guarda los datos rellenados en el formulario, llama a otra función que crea el botón del marcador y actualiza el fichero json donde están guardados los marcadores. Las dos últimas tareas se explicarán más adelante.

En el caso de que seleccione o no en el formulario una carpeta a la que añadir el marcador, habrá que llamar antes a una función que se encarga de obtener la ruta de dicha carpeta para poder insertar correctamente el nuevo marcador en el fichero json.

```
newBookmark() {
  var bookmark = document.getElementById("bookmark").value;
  var url = document.getElementById("url").value;
  var folder = "";

  if(document.getElementById("myfolders").options.length > 0) {
    folder = document.getElementById("myfolders").value;
  }

  if(folder) {
    this.insertElement(folder, false);
    this.newButton(bookmark, url, prefix + folder);
  }
  else {
    this.newButton(bookmark, url, "btns-creados");
    savebookmarks[bookmark] = url;
  }

  var datos1 = JSON.stringify(savebookmarks, null, '\t');
  fs.writeFile(ruta, datos1, function(err) {
    if (err)
      return console.error(err);
  });

 ...
}
```

Código 3.2: Nuevo marcador

newFolder() por otro lado va a crear las carpetas. El procedimiento es similar al de los marcadores, destacando que al tratarse de una carpeta hay que añadirla como una nueva opción a los *select* de los formularios y crear el *event listener* para hacer que la carpeta se expanda. Para esto último se hace uso de la función ***addEventFolder()***.

```
newFolder() {
```

```
    var folder = document.getElementById("folder").value;
    var fatherfolder = "";

    if(document.getElementById("myfolders-form-folder").options.length > 0) {
      fatherfolder = document.getElementById("myfolders-form-folder").value;
    }

    if(fatherfolder) {
      this.insertElement(fatherfolder, true);
      this.newFolderButton(folder, prefix + fatherfolder);
    }
    else {
      this.newFolderButton(folder, "btns-creados");
      savebookmarks[folder] = {};
    }

    var datos1 = JSON.stringify(savebookmarks, null, '\t');
    fs.writeFile(ruta, datos1, function(err) {
      if (err)
        return console.error(err);
    });

    var newoptionfolder = document.createElement('option');
    newoptionfolder.textContent = folder;
    newoptionfolder.value = folder;
    document.getElementById('myfolders').appendChild(newoptionfolder);
    var newoptionfolder1 = document.createElement('option');
    newoptionfolder1.textContent = folder;
    newoptionfolder1.value = folder;
    document.getElementById('myfolders-form-folder').appendChild(newoptionfolder1);

    this.addEventFolder(document.getElementById(folder));

   ...
  }
```

Código 3.3: Nueva carpeta

```
  addEventFolder(elementevent) {
    elementevent.addEventListener("click", function() {
      document.getElementById("btns-creados").style.height =
$('#web-bookmarks').height() - $('#container').height() - 27 + "px";
    });

    elementevent.addEventListener("click", function() {
      this.classList.toggle("active");
      var content = this.nextElementSibling;
      if (content.style.display === "block") {
        content.style.display = "none";
      } else {
        content.style.display = "block";
      }
    });
```

```
  }
```

Código 3.4: Añadir eventos carpetas nuevas

3.3 Guardar y cargar marcadores

Al usar Web Bookmarks hay que usar algún método para que los marcadores y carpetas creadas se guarden y no se pierda la lista de marcadores creada.

3.3.1 Fichero JSON

JSON [17] es un formato basado en texto estándar para representar datos estructurados en la sintaxis de objetos de JavaScript.

Para la estructura primero se pensó en utilizar vectores pero se comprobó que dificulta el acceso a los elementos y a diferenciar entre marcador y carpeta. La estructura que se adoptó al final en este proyecto es la siguiente:

```
{
  "Nombre-marcador": "Url-marcador",
  "Nombre-carpeta": {
    "Nombre-marcador": "Url-marcador",
    "Nombre-carpeta": {}
  },
  "Nombre-marcador": "Url-marcador"
}
```

Código 3.5: Estructura JSON

El fichero que se utiliza en Web Bookmarks es ***atomBookmarks.json***. Este fichero se va a crear por defecto dentro del directorio de Web Bookmarks que Atom crea donde están todos los paquetes instalados.

La ruta de este fichero puede variar un poco según el sistema operativo, pero en todos la ruta final sería "*/packages/web-bookmarks/atomBookmarks.json*", por lo que haciendo uso de la ruta del fichero de configuración de Atom, se puede obtener la ruta completa.

```
import { dirname } from 'path';
var configPath = atom.config.getUserConfigPath();
var ruta = dirname(configPath) + "/packages/web-bookmarks/atomBookmarks.json";
```

Código 3.6: Ruta fichero JSON

3.3.2 Paquete fs

El fichero se creará vacío la primera que se ejecute el plug-in y para trabajar con él se utiliza el paquete "*fs*" *[18]*, instalado mediante npm. Este paquete va a permitir leer y escribir el fichero json.

Se utilizan dos métodos en concreto: *readFileSync* y *writeFileSync*. El primero para leer el fichero y el segundo para escribir en él y para crearlo si no existe. Ambos métodos se utilizan es su versión síncrona, para evitar que continúe la ejecución del código hasta que no se hayan leído y/o escrito los datos y de esta manera acceder a datos que puedan ser erróneos.

```
var fs = require("fs");
```

```
...

  loadBookmarks() {
    var readbookmarks = "";

    try {
      readbookmarks = fs.readFileSync(ruta, "UTF8");
      savebookmarks = JSON.parse(readbookmarks);
    } catch (err) {
      var json = JSON.parse('{}');
      var datos = JSON.stringify(json, null, '\t');
      fs.writeFileSync(ruta, datos);

      savebookmarks = json;
    }
  }
```

Código 3.7: Leer datos/crear fichero JSON

En el código anterior se hace uso de los dos métodos nombrados.

loopBookmarks() se podría decir que es la función que se llama primero una vez se crea el tab de nuestro plug-in. Ya que es el encargado de leer y guardar la lista de marcadores del fichero json y en caso de no existir, crearlo para así poder trabajar con él. La variable *savebookmarks* es una variable global donde estarán guardados los marcadores.

3.3.3 Parse

Para poder trabajar con la lista de marcadores una vez leído el fichero json, se guarda la cadena en una variable y se parsea para que nos devuelva un objeto JavaScript.

Se hace uso del método ***JSON.parse()*** *[19]* que se puede ver en el punto anterior. Donde lo único necesario es pararle el texto a parsear, en nuestro caso el leído del fichero json.

Se puede utilizar la notación de punto o corchete para acceder a los elementos. Por ejemplo, para acceder a un valor sería de la forma: *savebookmarks[miMarcador]*, *savebookmarks[miCarpeta][miMarcador]*, etc. Para modificar el valor de algún elemento: *savebookmarks[miMarcador] = valor [20]*.

3.3.4 Stringify

JSON.stringify() *[21]* trabaja de la forma opuesta al anterior. Convierte un objeto JavaScript en una cadena JSON, para poder escribir la lista de marcadores actualizada en nuestro fichero json.

3.4 Editar y eliminar marcadores y carpetas

Una vez añadidas varias funcionalidades hubo que resolver un problema fundamental como es poder editar y/o eliminar un marcador o carpeta creado, ya sea porque se introduzcan mal los datos o porque ya no sea necesario.

Al final el método más práctico, sencillo y funcional, debido a algunas “limitaciones” que encontramos en Atom es abrir el fichero ***atomBookmarks.json*** en el propio programa para editarlo manualmente.

Para ello se ha añadido una opción en el menú del paquete, tanto en el menú superior como en el menú contextual al hacer clic derecho, dentro de ***Packages/Web bookmarks*** y dentro de la pestaña ***Web bookmarks*** respectivamente.

Una vez activada la opción se abrirá el fichero json, donde sólo hay que modificar los elementos. Cabe destacar que hay que tener cuidado con la estructura fijándose en que cada elemento esté entre comillas y que tengan una coma al final si no son el último elemento de la lista. A continuación se muestra un ejemplo:

```
{
  "TFG": {
    "Project": "https://github.com/ULL-ESIT-GRADOII-TFG/tfg-raul-martin-software/projects",
    "Issue": "https://github.com/ULL-ESIT-GRADOII-TFG/tfg-raul-martin-software/issues",
    "Code": "https://github.com/ULL-ESIT-GRADOII-TFG/web-bookmarks"
  },
  "Campus": "https://campusvirtual.ull.es/1819/"
}
```

Código 3.8: Ejemplo fichero JSON

Se puede observar como el elemento *Code* no tiene coma al final ya que es el último dentro de *TFG* y por último está *Campus* que tampoco tendría coma por ser el elemento final.

3.4.1 Refrescar vista

Cuando se realiza algún cambio en el fichero anterior y se guarda, no se va a apreciar el cambio en la vista a menos que se reinicie el plug-in. Para solucionar esto, en la primera sección del plug-in, descrita en el punto 3.1, hay un botón en la esquina superior derecha que servirá para refrescar la vista. Se puede observar en la Figura 3.1. Al hacer clic en él llamará a la función ***refreshBookmarks()***.

```
$('#icon-refresh').click(function(e){
  element.refreshBookmarks(element);
});

...

refreshBookmarks(element) {
  element.cleanData();
  element.loadBookmarks();
  element.createButtons(element);
  element.loadSelectFolder(savebookmarks);
  element.expandBtns();
  element.changeHeight();
  document.getElementById("btns-creados").style.height = $('#web-bookmarks').height()
- $('#container').height() - 27 + "px";
}
```

Código 3.9: Actualizar vista

Esta función lo que hará es llamar a una serie de funciones que también se llaman al iniciar el plug-in normalmente:

- ***cleanData()***: limpia los campos de los formularios y elimina de la vista todos los botones que se han creado en la segunda sección del plug-in.

```
cleanData() {
  $('#bookmark').val('');
  $('#url').val('');
  $('#myfolders').find("option:gt(0)").remove();
  $('#folder').val('');
  $('#myfolders-form-folder').find("option:gt(0)").remove();
  $("#btns-creados").empty();
  var old_element = document.getElementsByClassName("collapsible");
  for (var i = 0; i < old_element.length; i++) {
    var new_element = old_element[i].cloneNode(true);
    old_element[i].parentNode.replaceChild(new_element, old_element[i]);
  }
}
```

Código 3.10: Limpiar vista

- ***loadBookmarks()***: lee la lista de marcadores del fichero json. Comentado en el punto 3.3.2.
- ***createBookmarks()***: esta función va a controlar el tipo de elemento que se crea, un marcador o una carpeta. Dependiendo del tipo que sea, llamará a una función o a otra.

```
createButtons(element) {
  for (var i in savebookmarks) {
    if(typeof savebookmarks[i] === 'object'){
      var folder = i;
      element.newFolderButton(folder, "btns-creados");
      element.loopBookmarks(folder, savebookmarks[i], prefix + folder);
    }
    else {
      var bookmark = i;
      var url = savebookmarks[i];
      element.newButton(bookmark, url, "btns-creados");
    }
  }
}
```

Código 3.11: Controlador tipo de elemento

- ***loadSelectFolder()***: carga las carpetas existentes en los *select* de los formularios.

```
loadSelectFolder(thisbookmarks) {
  for (var i in thisbookmarks) {
    if(typeof thisbookmarks[i] === 'object') {
      var newoptionfolder = document.createElement('option');
      newoptionfolder.textContent = i;
      newoptionfolder.value = i;
      document.getElementById('myfolders').appendChild(newoptionfolder);
                        var newoptionfolder1 = document.createElement('option');
      newoptionfolder1.textContent = i;
      newoptionfolder1.value = i;

document.getElementById('myfolders-form-folder').appendChild(newoptionfolder1);
                        this.loadSelectFolder(thisbookmarks[i]);
```

```
      }
    }
  }
```

Código 3.12: Cargar opciones select

- ***expandBtns***: se encarga de permitir que los formularios y carpetas obtengan el efecto de contraerse y expandirse. Se comentó en el punto 3.1.
- ***changeHeight()***: esta función aunque no se mostró, se comentó también en el punto 3.1 y se encarga de redimensionar la segunda sección del plug-in cuando se expanden o contraen los formularios.

```
  changeHeight() {
    var changeDim =
document.getElementById("container").getElementsByClassName("collapsible");
    for (var i = 0; i < changeDim.length; i++) {
      changeDim[i].addEventListener("click", function() {
        document.getElementById("btns-creados").style.height =
$('#web-bookmarks').height() - $('#container').height() - 27 + "px";
      });
    }
  }
```

Código 3.13: Redimensionar tamaño segunda sección

3.5 Abrir marcadores

La funcionalidad principal de este plug-in no es otra que la de tener una lista de marcadores web. Por tanto, debe haber un mecanismo que permita abrir la url guardada del marcador en el navegador.

Para realizar esta tarea se hace uso del módulo *shell [22]* que proporciona funciones relacionadas con la integración de escritorio. Concretamente se utiliza el método ***shell.openExternal()***, que abre la URL del protocolo externo dada de la manera predeterminada del escritorio. En nuestro caso, al tratarse de páginas web, se abrirán en el navegador por defecto.

El proceso que sigue el plug-in para llamar a este método consiste en capturar los elementos donde se hace clic dentro de la segunda sección del plug-in y si el elemento es un botón, se obtiene el valor que corresponde con la url. Por último se llama al método que abre la url en el navegador.

En el caso de que el botón cliqueado sea una carpeta, no se ejecutaría dicho método ya que el valor del atributo *value* es nulo y no llega a ser llamado.

```
  var shell = require("electron").shell;

  ...

  $('#btns-creados').click(function(e){
    if((e.target.tagName == 'BUTTON') && (e.target.value)){
      var id = e.target.id;
      var value = e.target.value;
      element.openUrl(value);
    }
```

```
});

...

openUrl(link) {
  shell.openExternal(link);
}
```

Código 3.14: Abrir marcadores

Capítulo 4 Preparación y publicación

4.1 Preparación y estudio de tutoriales

Antes de empezar con el desarrollo del plug-in es conveniente realizar algunos tutoriales y revisar alguna documentación que Atom proporciona. Una herramienta que va a ayudar en todo el desarrollo es el *Manual de Vuelo de Atom [23]*. Este manual proporciona toda la información necesaria sobre el uso e implementación de multitud de recursos para Atom, desde la instalación del propio editor, hasta una guía para crear varios paquetes.

Por tanto, si es la primera vez que vamos a desarrollar un plug-in para Atom, es recomendable comenzar creando algún paquete de prueba para ver el funcionamiento de estos.

Hay varios tutoriales interesantes sobre cómo crear tu primer plug-in para Atom [24], [25], pero quizás el más útil para este proyecto es *Active Editor Info [26]*. Este paquete va a mostrar cierta información sobre un fichero que tengamos abierto. Pero lo interesante no es esto si no la manera y lugar en la que muestra la información. En vez de usar un *modal panel* como en otros casos, se muestra la vista en un elemento del espacio de trabajo. Para ello se registra un *opener* con Atom. Un *opener* es una función que acepta una *URI* y devuelve una vista. Este ha sido el método utilizado para mostrar la vista de Web Bookmarks.

```
'use babel';

import WebBookmarksView from './web-bookmarks-view';
import { CompositeDisposable, Disposable } from 'atom';
import { dirname } from 'path';
var configPath = atom.config.getUserConfigPath();
var ruta = dirname(configPath) + "/packages/web-bookmarks/atomBookmarks.json";

export default {

  subscriptions: null,

  activate(state) {
    // Events subscribed to in atom's system can be easily cleaned up with a
CompositeDisposable
    this.subscriptions = new CompositeDisposable(
      atom.workspace.addOpener(uri => {
        if(uri === 'atom://web-bookmarks') {
          return new WebBookmarksView();
        }
      }),
```

```
      // Register command that toggles this view
      atom.commands.add('atom-workspace', {
        'web-bookmarks:toggle': () => this.toggle()
      }),

      atom.commands.add('atom-workspace', {
        'web-bookmarks:edit-bookmarks': () => atom.workspace.open(ruta)
      }),

      new Disposable(() => {
        atom.workspace.getPaneItems().forEach(item => {
          if (item instanceof WebBookmarksView) {
            item.destroy();
          }
        })
      })
    );
  },

  deactivate() {
    this.subscriptions.dispose();
  },

  toggle() {
    atom.workspace.toggle('atom://web-bookmarks');
  }

};
```

Código 4.1: Eventos y comandos registrados

En este fichero también se van a registrar los comandos que se quieran registrar para el plug-in. Para este, se observan dos:

- ***toggle***: este comando será el encargado de mostrar y ocultar el plug-in o la vista de éste.
- ***edit-bookmarks***: al ejecutar este comando se va a abrir en el editor el fichero json donde se han guardados los marcadores .

4.2 Publicar plug-in

Lo última tarea a realizar una vez desarrollado el plug-in será publicarlo para que sea accesible, junto con los numerosos paquetes ya existentes [27].

Atom proporciona una guía sencilla y rápida en su manual [28]. Hay que cumplir con una serie de requisitos antes de publicarlo:

- El fichero *package.json* tiene los campos *name*, *description* y *repository*.
- El fichero *package.json* tiene el campo *version* con un valor de "0.0.0".
- El fichero *package.json* tiene el campo *engines* que contenga una entrada para Atom como: *“engines”: {“atom”: “>=1.0.0 <2.0.0”}*.
- El paquete tiene un fichero *README.md* en la raíz.
- La URL del repositorio en el fichero *package.json* tiene la misma URL que la de nuestro repositorio.

- El paquete está en un repositorio Git que ha sido pusheado a GitHub.

Una vez se cumplen estos requisitos, también es recomendable comprobar que no exista ya un paquete con el mismo nombre al que se desea poner, para ello, se puede comprobar buscándolo en [27].

Por último, sólo faltaría situarnos en la raíz del plug-in y ejecutar el siguiente comando:

```
C:\> cd path-to-your-package
C:\> apm publish minor
```

Código 4.2: Publicar plug-in

Capítulo 5 Guía de usuario

En este capítulo se va a detallar una guía de usuario para explicar lo necesario sobre este plug-in, desde la instalación hasta el uso.

Para realizar las tareas que se describen a continuación se da por hecho que el usuario tiene instalado el editor Atom. También debería tener instalado *Atom package manager*, *apm*, en caso de instalar el plug-in manualmente.

5.1 Instalación

Hay dos formas de instalar un paquete o plug-in en Atom:

1. Introducir el comando *apm install package-name* en la terminal

```
C:\> apm install web-bookmarks
```

Código 5.1: Instalar Web Bookmarks

2. Abrir Atom, ir a *File* > *Settings* > *Install* y buscar el paquete que se quiere instalar y pulsar en *install*.

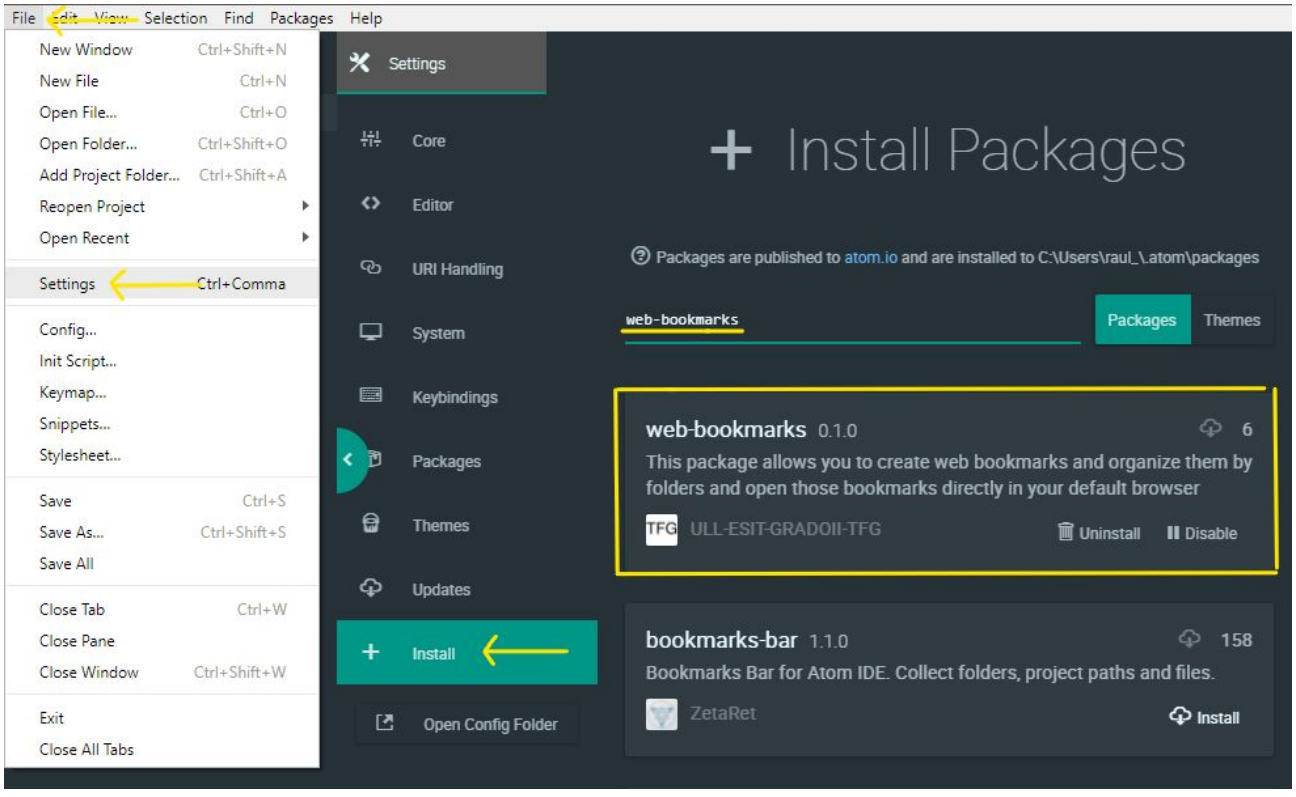


Figura 5.1: Instalar Web Bookmarks

5.2 Uso

El plug-in tiene tres maneras de iniciarse:

1. Mediante el menú superior accediendo a *Packages* > *Web Bookmarks* > *Toggle*.
2. Mediante el menú contextual haciendo clic derecho con algún fichero abierto, accediendo a *Web Bookmarks* > *Toggle*.
3. A través de la combinación de teclas *Alt+Ctrl+O*

Una vez iniciado, en la parte superior hay dos botones que contienen los formularios con los que añadir marcadores y carpetas (revisar punto 3.1). Estos elementos se podrán organizar por carpetas y subcarpetas.

Para editar y/o eliminar algún elemento de la lista se hace a partir de un fichero json (revisar punto 3.3.1 y 3.4). El acceso a este fichero se hace a través de los menús, del mismo modo que para activar el plug-in, pero esta vez con la opción *Edit/Delete Bookmarks*.

El fichero se va a crear en la carpeta de nuestro plug-in que está en la ruta donde atom ubica por defecto los paquete instalados. El nombre de este fichero será *atomBookmarks.json*.

../.atom/packages/Web-bookmarks/atomBookmarks.json

En este fichero podremos cambiar el nombre, eliminar u ordenar los marcadores. Pero hay que tener cuidado con las comillas y las comas, para evitar errores de sintaxis. Una vez realizadas las modificaciones hay que guardar el archivo y para que estos cambios surjan efecto en la vista, hay que pulsar el botón de refrescar, situado en la esquina superior derecha.

Capítulo 6 Conclusiones y líneas futuras

El uso de editores de código u otras herramientas similares están a la orden del día durante el desarrollo de las labores docentes tanto de profesores como de alumnos. Esto hace que acabemos teniendo preferencias sobre qué herramientas usar.

Por tanto, un editor de código como Atom ofrece muchas posibilidades debido al potencial que tiene gracias a los paquetes que se pueden instalar. Estos paquetes pueden ayudar tanto a la hora de organizar los ficheros, de trabajar en equipo, autocompletar o resaltar código, entre otras muchas cosas, ya que hay una gran multitud de paquetes, más de 8000 actualmente, aunque muchos de estos ofrecen características similares.

Otro punto a favor de Atom es que al igual que ofrece la posibilidad de instalar paquetes, también podremos crear uno para añadir alguna funcionalidad nueva o mejorar alguna que ya exista.

De todo esto nace Web Bookmarks, que como se ha comentado anteriormente, pretende facilitar el trabajo tanto a profesores como a alumnos.

Hacer uso del navegador web es muy común y frecuente en el desarrollo de las tareas académicas, ya sea realizando una práctica en el caso de un alumno o corrigiendo prácticas en el caso de un profesor. El caso, es que en muchas ocasiones hay que estar cambiando entre programas y abriendo pestañas nuevas en el navegador para acceder a sitios que usamos con frecuencia. Lo lógico es que esos sitios estén guardados como marcadores en nuestro navegador. Pero esto también puede ser tedioso, ya que la lista de marcadores puede ser extensa al tener los marcadores personales en ella.

Por tanto, una buena solución es tener listas de marcadores separadas. De esta manera, dentro del propio editor podemos tener un fácil y rápido acceso a todos esos sitios web que usamos con frecuencia al realizar estas tareas, como puede ser el campus virtual, GitHub, documentación de algún lenguaje o herramienta, etc.

Durante el propio desarrollo de este trabajo ha sido muy útil para abrir documentación, el repositorio, la página del campus de la asignatura, etc, y ahorra mucho tiempo y esfuerzo.

Por último, cabe destacar que debido a que Atom es de código abierto y está desarrollado por GitHub, hay una gran comunidad detrás donde encontrar ayuda y obtener muchas referencias para el desarrollo de nuestros paquetes. Esto puede ayudar mucho a la hora de resolver errores o buscar soluciones para nuestras tareas.

Capítulo 7 Summary and Conclusions

The use of code editors or other similar tools are the order of the day during the development of the teaching tasks of teachers and students. This makes us end up having preferences about what tools to use.

Therefore, a code editor such as Atom offers many possibilities due to the potential it has thanks to the packages that can be installed. These packages can help when organizing files, working as a team, autocomplete or highlight code, among many other things, since there are a large number of packages, more than 8000 currently, although many of these offer similar characteristics.

Another point in favor of Atom is that, as it offers the possibility of installing packages, we can also create one to add some new functionality or improve one that already exists.

From all this, Web Bookmarks was born, which, as previously mentioned, aims to facilitate the work of both teachers and students.

Making use of the web browser is very common and frequent in the development of academic tasks, either performing a practice in the case of a student or correcting practices in the case of a teacher. The case is that on many occasions you have to be switching between programs and opening new tabs in the browser to access sites that we use frequently. The logical thing is that these sites are saved as bookmarks in our browser. But this can also be tedious, since the list of bookmarks can be extensive by having personal bookmarks in it.

So, a good solution is to have lists of bookmarks separate. In this way, inside the editor we can have an easy and fast access to all those websites that we use frequently when performing these tasks, such as the virtual campus, GitHub, documentation of a language or tool, etc.

During the development of this work it has been very useful to open documentation, the repository, the campus page of the subject, etc, and it saves a lot of time and effort.

Finally, it should be noted that because Atom is open source and developed by GitHub, there is a large community behind where to find help and obtain many references for the development of our packages. This can help a lot at the time of solving errors or finding solutions for our tasks.

Capítulo 8 Presupuesto

En este capítulo se describe el presupuesto del proyecto.

8.1 Coste hardware

Para el desarrollo de este trabajo se necesita un ordenador con sistema operativo Windows, cuya configuración no requiere que sea de componentes muy caros y de gama alta. Por lo que el presupuesto se ajusta a 500 euros.

8.2 Coste humano

El desarrollo de este proyecto ha tenido una duración de 4 meses, con un coste de 1600 euros al mes, que implicaría un total de 6400 euros.

8.3 Coste total

El presupuesto total para este proyecto sería de un total de 6900 euros.

Bibliografía

[1] J. Cabana, “Drauta,” Drauta agencia digital, 14-Aug-2017. [Online]. Available: https://www.drauta.com/atom-un-ide-para-el-desarrollador-web.

[2] F. de Node.js, “Acerca,” Node.js. [Online]. Available: https://nodejs.org/es/about/.

[3] “About Electron,” Electron. [Online]. Available: https://electronjs.org/docs/tutorial/about.

[4] “github,” Atom. [Online]. Available: https://atom.io/packages/github.

[5] “remote,” Atom. [Online]. Available: https://atom.io/packages/remote-atom.

[6] “What is GitHub?,” What is GitHub. [Online]. Available: https://www.w3schools.com/whatis/whatis_github.asp.

[7] “HTML,” Documentación web de MDN. [Online]. Available: https://developer.mozilla.org/es/docs/Web/HTML.

[8] “CSS,” Documentación web de MDN. [Online]. Available: https://developer.mozilla.org/es/docs/Web/CSS.

[9] “JavaScript,” Documentación web de MDN. [Online]. Available: https://developer.mozilla.org/es/docs/Web/JavaScript.

[10] “CoffeeScript,” CoffeeScript. [Online]. Available: https://coffeescript.org/.

[11] T. C. L. Team, “Overview,” Getting started | Less.js. [Online]. Available: http://lesscss.org/.

[12] “ECMAScript 2015,” ECMAScript 6. [Online]. Available: https://www.w3schools.com/js/js_es6.asp.

[13] “About npm,” About npm | npm Documentation. [Online]. Available: https://docs.npmjs.com/about-npm/.

[14] “W3C,” W3C España. [Online]. Available: https://www.w3c.es/Consorcio/.

[15] “Extreme programming,” Wikipedia, 05-May-2019. [Online]. Available: https://en.wikipedia.org/wiki/Extreme_programming.

[16] “element.addEventListener,” Documentación web de MDN. [Online]. Available: https://developer.mozilla.org/es/docs/Web/API/EventTarget/addEventListener.

[17] “Trabajando con JSON,” Documentación web de MDN. [Online]. Available: https://developer.mozilla.org/es/docs/Learn/JavaScript/Objects/JSON.

[18] “Node.js v12.3.1 Documentation,” File System | Node.js v12.3.1 Documentation. [Online]. Available: https://nodejs.org/api/fs.html#fs_file_system.

[19] JSON.parse(). [Online]. Available:

https://www.w3schools.com/js/js_json_parse.asp.

[20] JSON Syntax. [Online]. Available: https://www.w3schools.com/js/js_json_syntax.asp.

[21] JSON.stringify(). [Online]. Available: https://www.w3schools.com/js/js_json_stringify.asp.

[22] “shell,” Electron. [Online]. Available: https://electronjs.org/docs/api/shell.

[23] “Atom,” Atom. [Online]. Available: https://flight-manual.atom.io/.

[24] J. Britton, “Building your first Atom plugin,” The GitHub Blog, 16-Jan-2019. [Online]. Available: https://github.blog/2016-08-19-building-your-first-atom-plugin/.

[25] Vcomposieux, “Create your first Atom package,” Blog Eleven Labs, 05-Dec-2016. [Online]. Available: https://blog.eleven-labs.com/en/create-atom-package/.

[26] Package: Active Editor Info. [Online]. Available: https://flight-manual.atom.io/hacking-atom/sections/package-active-editor-info/.

[27] “Packages make Atom do amazing things,” Atom. [Online]. Available: https://atom.io/packages.

[28] Publishing. [Online]. Available: https://flight-manual.atom.io/hacking-atom/sections/publishing/.

[29] “web-bookmarks,” Atom. [Online]. Available: https://atom.io/packages/web-bookmarks.