eman ta zabal zazu
Universidad del País Vasco
Euskal Herriko Unibertsitatea
INFORMATIKA FAKULTATEA
FACULTAD DE INFORMÁTICA

Grado en Ingeniería Informática

Ingeniería del Software

Trabajo de Fin de Grado

AC-Check: Una extensión de análisis, agregación y generación de informes de accesibilidad web

Autor

Mikel Iturria Silveti

Director

Juan Miguel López Gil

Resumen

En este documento está recogida la memoria del Trabajo de Fin de Grado titulado como "AC-Check: Una extensión de análisis, agregación y generación de informes de accesibilidad web" del alumno Mikel Iturria Silveti, estudiante del Grado en Ingeniería Informática, especialidad en Ingeniería del Software, para la Facultad de Informática de la Universidad del País Vasco / Euskal Herriko Unibertsitatea (UPV/EHU).

El tutor del proyecto ha sido Juan Miguel López Gil, profesor del departamento de *Lenguajes y Sistemas Informáticos* de la Facultad de Informática de la UPV/EHU.

El objetivo principal del proyecto consta de dos partes: La primera, crear una API que genere informes de accesibilidad automáticamente de distintos analizadores y los agregue sin perder información y una segunda parte, en la que se crea una extensión para *Google Chrome* que reciba los informes. Además, la extensión podrá mostrar los resultados, subir y almacenar en memoria informes manuales, agregar informes manuales, descargar los informes (ya sean automáticos o manuales) para almacenarlos/compartirlos y marcar los elementos en conflicto recibidos en los informes sobre la propia página web.

Índice general

Índice de figuras

Índice de tablas

1. CAPÍTULO

Introducción y antecedentes

En este capítulo se expondrán los antecedentes del proyecto. Concretamente, se hablará de la descripción del proyecto, el interés previo que había en el tema del proyecto y una pequeña investigación realizada sobre aplicaciones similares.

1.1. Descripción

Este proyecto consiste en crear una herramienta para ayudar a los usuarios a obtener informes de accesibilidad sobre una determinada página web. El proyecto tiene dos partes, una parte de agregación de informes y una parte de generación automática de informes.

La parte de la agregación es una extensión para el navegador dónde el usuario puede subir varios informes hechos de manera manual y la herramienta agregará dichos informes creando un nuevo informe. El nuevo informe, con la información agregada, tendrá un formato específico que permitirá que se visualice en la página web del *W3C*[1]. Si el usuario no quiere realizar la generación de informe de manera manual, podrá solicitar que se genere automáticamente.

La generación automática de informes se realiza mediante una herramienta de tipo aplicación de servidor creada por mí. Esta herramienta se encargará de, dado un enlace, solicitar a diversos analizadores de accesibilidad que generen un informe sobre el enlace dado. La herramienta realizará web scraping sobre esos resultados y generará un informe que será enviado a la extensión.

Una vez la extensión haya recibido el informe, se mostrarán los resultados del informe en la misma. La extensión, además de mostrar los resultados, mostrará los códigos en conflicto que generen esos resultados. Si el usuario clica en uno de esos códigos, la extensión modificará la página web para mostrar dónde se está dando ese error de accesibilidad, resaltándolos con un cambio de estilo y moviendo la vista del navegador hacia ese elemento. El usuario podrá también descargar el informe para enviarlo (podrá ser añadido de nuevo a la extensión por cualquier usuario) o visualizarlo en la web del *W3C*.

1.2. Interés previo al proyecto

Cuando Juan Miguel me presentó el proyecto me pareció una idea interesante sobre todo porque sería un proyecto útil y que podría ahorrar mucho tiempo a los desarrolladores web. Una herramienta que ayuda a agregar informes de más de un evaluador de manera automática simplifica mucho el proceso de generación de informes. Si hubiera cinco personas haciendo un análisis por su cuenta, cuando terminaran perderían mucho tiempo

[1] https://www.w3.org/WAI/eval/report-tool/

agregando los informes para que los resultados reflejaran los comentarios de las cinco personas. Ese proceso es instantáneo con mi herramienta,

Otro aspecto que me pareció interesante fue el de realizar informes de manera automática. Hay páginas web que realizan el análisis automáticamente pero luego muestran el resultado en la propia página, en HTML. Si alguien quiere enviar esos resultados a otra persona solo lo podrá realizar haciendo una captura de pantalla, porque no hay posibilidad de extraer los datos. Me pareció muy interesante crear una herramienta que pudiera extraer esos datos, analizarlos, procesarlos y almacenarlos.

1.3. Aplicaciones similares

No hay ninguna extensión que permita realizar agregación de informes de accesibilidad en JSON ni tampoco aplicaciones que extraigan los datos de los analizadores de accesibilidad automáticos. No obstante, si que hay algo parecido a cerca del mostrado de los errores sobre la propia página web mediante una extensión. Esa extensión [ext, 2022e] está creada por una página web que realiza análisis automáticos, la página de WAVE.[2]. Otro ejemplo sería la herramienta *AInspector* [ext, 2022a], aunque dicha extensión solo está disponible para Firefox y no para Chrome.

En aspectos generales, ambas extensiones realizan una función mejor que la que yo he creado en el apartado del mostrado de errores. Sin embargo, las extensiones mencionadas tiene menos funcionalidades que mi extensión, ya que no tienen la opción de subir informes, agregar informes, obtener un informe automático agregado de los resultados de diversos analizadores automáticos o descargar el informe de la página.

Además de los ya mencionados, se analizaron las siguientes extensiones:

- Axe Devtools [ext, 2022d]
- SiteImprove [ext, 2022g]
- ACheck Helper[ext, 2022c]
- Accessible web helper [ext, 2022b]
- LERA [ext, 2022f]

[2]Página web: `https://wave.webaim.org/`

2. CAPÍTULO

Planificación inicial del proyecto

Este capítulo recoge le planificación inicial que se planteó para el proyecto. El capítulo abordará tres aspectos. El primero, referente a los objetivos y al alcance del proyecto, el segundo, sobre las estimaciones de tiempo de las tareas donde veremos una estructura de descomposición del trabajo (EDT), una estimación horaria por tarea y un gráfico de Gantt. También se mencionarán las herramientas a utilizar y el porqué de su uso además de los riesgos y los interesados del proyecto. Por último, se hablará sobre los distintos sistemas de información que se han usado en el proyecto.

2.1. Objetivos del proyecto

El objetivo principal de este TFG es el de crear una herramienta que facilite la creación de informes de accesibilidad a las personas que les corresponda realizarlos. Para ello, debe de cumplir principalmente ciertos objetivos:

- Desarrollar una aplicación de tipo extensión de navegador que se encargue de:
  - —Agregar informes subidos por usuarios o obtenidos automáticamente.
  - —Recibir informes automáticos de la aplicación de servidor.
  - —Mostrar al usuario en la propia página web el informe con los resultados.
  - —Marcar y resaltar los elementos HTML que producen errores/advertencias en la propia página web al ser clicados por el usuario desde la vista de resultados.
- Desarrollar una aplicación de servidor que se encargue de:
  - —Realizar un análisis de páginas web de manera automática.
  - —Agregar resultados de diferentes analizadores para una determinada página.
  - —Generar un informe con formato W3C para que pueda ser importado directamente por la *Report Tool* de W3C[1].
- Realizar pruebas de ambas aplicaciones en local y comprobar su correcto funcionamiento.
- Que el proyecto sea extensible tanto en lo que a la extensión respecta como a lo que a la aplicación de servidor respecta, pudiendo ser añadidos en el futuro más páginas web de análisis automático.
- Documentar el código para ser entendido por otros desarroladores, así como el proceso de creación de la extensión en una memoria.

2.2. Estructura de descomposición del trabajo (EDT)

La EDT o estructura de descomposición del trabajo es un diagrama en la que se realiza una descomposición jerárquica del trabajo a realizar para cumplir con los objetivos y crear los entregables requeridos.

[1] https://www.w3.org/WAI/eval/report-tool/

El EDT de este proyecto cuenta con 5 secciones: formación, gestión, desarrollo, pruebas y documentación, de los que extraeremos los paquetes de trabajo para la planificación.

1. **Formación**: Esta sección incluye el paquete de trabajo para la formación previa al proyecto. Las competencias serían aprender el uso de *Flask*, *Beatiful Soup*, *Selenium* y aprender el uso más avanzado de *JavaScript* y *JQuery*, ya que en clase solo dimos aspectos superficiales.

   Paquetes de trabajo extraídos de esta sección:

   - H- Estudio de las herramientas.

2. **Gestión**: La sección de gestión tiene las competencias tanto de la planificación inicial y la definición de objetivos del proyecto como el seguimiento a lo largo del transcurso del mismo, con las reuniones del tutor y el Seguimiento y Control.

   Paquetes de trabajo extraídos de esta sección:

   - P- Planificación.
   - S- Seguimiento

3. **Desarrollo**: Esta sección es la sección a la que más tiempo se va a invertir ya que es la dedicada al desarrollo de la tecnología de este TFG. Incluye el desarrollo de la aplicación de servidor, de la aplicación de cliente, la creación de la extensión y la integración de las tres aplicaciones creadas para tener una extensión funcional.

   Paquetes de trabajo extraídos de esta sección:

   - DS - Desarrollo de la app de servidor
   - DJS - Desarrollo de la app JavaScript
   - EX- Creación de la extensión

4. **Pruebas**: Esta sección contendrá las pruebas para confirmar el correcto funcionamiento de la aplicación creada en la sección anterior. Posee dos competencias: El testeo de la generación de informes y el testeo del marcado de resultados en la página web.

   Paquetes de trabajo extraídos de esta sección:

   - P- Pruebas en webs reales

5. **Documentación**: Por último, el paquete documentación se encargará de dos competencias: la memoria del TFG y la documentación del código realizado.

   Paquetes de trabajo extraídos de esta sección[2]:

   - M- Memoria.

Definidas las secciones, el EDT resultante sería el siguiente:

Figura 2.1: EDT del proyecto

[2]Nota: La documentación del código realizado se refleja en los paquetes DS, DJS y EX respectivamente para cada código.

2.3. Paquetes de trabajo

Los paquetes de trabajo resultantes del apartado anterior se podrían agrupar de la siguiente manera:

Paquete de trabajo	Descripción
Formación	
H- Estudio de las herramientas.	Estudio de las herramientas necesarias para la realización de la tecnología.
Gestión	
P- Planificación	Realización de la planificación incial del proyecto
S- Seguimiento	Seguimiento y control del proyecto, incluye las tutorías con el profesor tutor.
Desarrollo	
DS - Desarrollo de la aplicación de servidor	Desarrollo de la aplicación del servidor para producir informes automáticos.
DJS - Desarrollo de la aplicación JavaScript	Desarrollo de la aplicación de lado cliente, encargada de enseñar los resultados del informe, subir y descargar informes, agregar informes y mostrar los resultados del informe sobre los elementos HTML de la página web.
EX- Creación de la extensión	Creación de la extensión e integración de las dos aplicaciones anteriores con la extensión
Pruebas	
P- Pruebas en webs reales	La realización de las pruebas para el correcto funcionamiento de la aplicación incluyendo tanto las pruebas de la realización de informes como las pruebas de marcado de elementos HTML.
Documentación	
M- Memoria.	La realización de la memoria del proyecto.

Tabla 2.1: Paquetes de trabajo del proyecto y su descripción

2.4. Tareas por paquete

Una vez tenemos definidos los paquetes de trabajo, vamos a extraer las tareas a realizar por cada paquete de trabajo.

- P - Planificación:
  - **P1- Definición del proyecto**: La definición del proyecto será la tarea en la que se definirá, con ayuda del tutor, el proyecto. Esto incluye la definición de objetivos, el alcance, el funcionamiento del proyecto y la finalidad del mismo.
  - **P2- Planificación**: La planificación es la tarea en la que se realiza una planificación inicial con un EDT, una definición de paquetes de trabajo y tareas, una estimación de dedicación y un diagrama de Gantt.
- H- Estudio de las herramientas:
  - **H1- Estudio de Flask**: Será la tarea en la que se realice un estudio de sintaxis y funcionamiento de la herramienta Flask.
  - **H2- Estudio de Beatiful Soup**: Será la tarea en la que se realice un estudio de sintaxis y funcionamiento de la herramienta Beatiful Soup.
  - **H3- Estudio de Selenium**: Será la tarea en la que se realice un estudio de sintaxis y funcionamiento de la herramienta Selenium.
  - **H4- Estudio de JavaScript avanzado**: Habrá que realizar un estudio más en profundidad de JavaScript, ya que tendrá un peso muy importante en el proyecto y en clase solo vimos aspéctos superficiales. Esta tarea incluye el estudio del árbol DOM de HTML así como la modificación dinámica del mismo.
  - **H5- Estudio de JQuery avanzado**: También, al igual que con JavaScript, habrá que realizar un estudio en profundidad de JQuery.
- S- Seguimiento:
  - **S1- Control del funcionamiento**: Controles periódicos del correcto funcionamiento de la aplicación.
  - **S2- Reuniones con el tutor**: Tiempo estimado para las reuniones con el tutor. Incluye el seguimiento y control sobre la planificación.
- DS - Desarrollo de la app de servidor: Este paquete de trabajo se dividirá en dos subpaquetes y una tarea.

  —Sub-paquete DS.AM - AccessMonitor: Obtención de resultados desde la web de *AccessMonitor*.

- **DS.AM1- Análisis de la página**: Análisis de la página web para saber donde se muestra la información y de cómo se pasa el parámetro de la URL a analizar, entre otros.
- **DS.AM2- Programación del scraping**: Programación de la obtención de resultados mediante scraping en dicha página para la URL que deseemos.
- **DS.AM3- Procesado de resultados**: Procesado de los resultados del scraping.

—Sub-paquete DS.AM - AChecker: Obtención de resultados desde la web de *AChecker*.

- **DS.AC1- Análisis de la página**: Análisis de la página web para saber donde se muestra la información y de cómo se pasa el parámetro de la URL a analizar, entre otros.
- **DS.AC2- Programación del scraping**: Programación de la obtención de resultados mediante scraping en dicha página para la URL que deseemos.
- **DS.AC3- Procesado de resultados**: Procesado de los resultados del scraping.

—Tarea independiente dentro del paquete DS:

- **DS1- Agregación de resultados de los dos analizadores**: Tarea encargada de la agregación de los resultados de ambos analizadores.

- DJS - Desarrollo de la aplicación JavaScript:

  - **DJS1- Creación de la estructura**: Creación de la estructura de la parte del cliente.
  - **DJS2- Implementación de tareas menores**: Implementación de tareas menores como subir un informe a la extensión o descargar un informe de la extensión, entre otros.
  - **DJS3- Implementación de la agregación**: El usuario podrá subir varios informes y se espera que la extensión los agregue. Habrá que realizar la implementación para esa funcionalidad.
  - **DJS4- Implementación del mostrado de resultados**: Ya sea porque se han agregado los informes subidos por el usuario o se ha obtenido un informe automático, hay que implementar el código para que la extensión muestre los resultados de forma clara, ordenada y legible.

    - **DJS5- Implementación de resaltar los resultados en la página web**: La implementación del marcado y el resaltado de los elementos en conflicto presentes en el resultado del informe sobre la página web (modificación del elemento HTML original).

- EX- Creación de la extensión:

    - **EX1- Análisis de extensiones Chrome**: Análisis de qué es una extensión de Chrome, cómo funcionan, dónde se crean, cómo se almacenan y sobre todo cómo interactuan con el navegador.
    - **EX2- Creación de la extensión**: Creación de la extensión con todo el código de *backgorund* para su correcto funcionamiento (e.g: Encendido/apagado de la extensión).
    - **EX3- Integración del código JS**: Integración del código del lado de cliente que incluye el almacenamiento y el muestreo de los resultados así como la agregación y el marcado de elementos HTML con la extensión.
    - **EX4- Conexión con la app de servidor**: Establecer una conexión correcta con la aplicación de servidor para que puedan mandar información entre las dos aplicaciones.
    - **EX5- Diseño de la extensión**: Relativo al diseño de la extensión: Logos, gama de colores, estilo de los botones...

- P- Pruebas en webs reales:

    - **P1- Pruebas de generación de informes**: Pruebas periódicas de que el informe se está generando de manera correcta.
    - **P2- Pruebas de marcado de elementos**: Pruebas periódicas de que los elementos HTML originales de los que aparecen en el resultado del informe se pueden marcar y resaltar dentro de la web.

- M- Memoria: El paquete es su propia tarea. La realización de la memoria del proyecto.

2.5. Estimación de dedicación

Una vez tenemos las tareas definidas podemos hacer una estimación de la dedicación para cada tarea. La dedicación estimada es la siguiente:

TAREA	HORAS ESTIMADAS
P - Planificación	11
P1- Definición del proyecto	4
P2- Planificación	7
H- Estudio de las herramientas	21
H1- Estudio de Flask	5
H2- Estudio de Beatiful Soup	5
H3- Estudio de Selenium	5
H2- Estudio avanzado JS	3
H2- Estudio avanzado JQuery	3
S- Seguimiento	4
S1- Control del funcionamiento	2
S2- Reuniones con el tutor	2
DS - Desarrollo de la aplicación de servidor	92
DS.AM - AccessMonitor	45
DS.AM1- Análisis de la página	5
DS.AM2- Programación del scraping	30
DS.AM3- Procesado de resultados	10
DS.AM - AChecker	45
DS.AC1- Análisis de la página	5
DS.AC2- Programación del scraping	30
DS.AC3- Procesado de resultados	10
DS1- Agregación de resultados de los dos analizadores	2
DJS - Desarrollo de la app JavaScript	85
DJS1- Creación de la estructura	5
DJS2- Implementación de tareas menores	15
DJS3- Impl. de la agregación	20
DJS4- Impl. del mostrado de resultados	25
DJS5- Impl. de resaltar los resultados en la página web	20
EX- Creación de la extensión	41
EX1- Análisis de extensiones Chrome	5
EX2- Creación de la extensión	10
EX3- Integración del código JS	15
EX4- Conexión con la app de servidor	3
EX5- Diseño de la extensión	8
P- Pruebas en webs reales	11
P1- Pruebas de generación de informe	5
P2- Pruebas de marcado de elementos	6
M- Memoria	50
Total	**315**

Tabla 2.2: Tabla de estimación de tiempos

2.6. Diagrama de Gantt

A continuación se podrá observar el inicio y fin de las tareas mediante el uso de un diagrama de Gantt.[3]

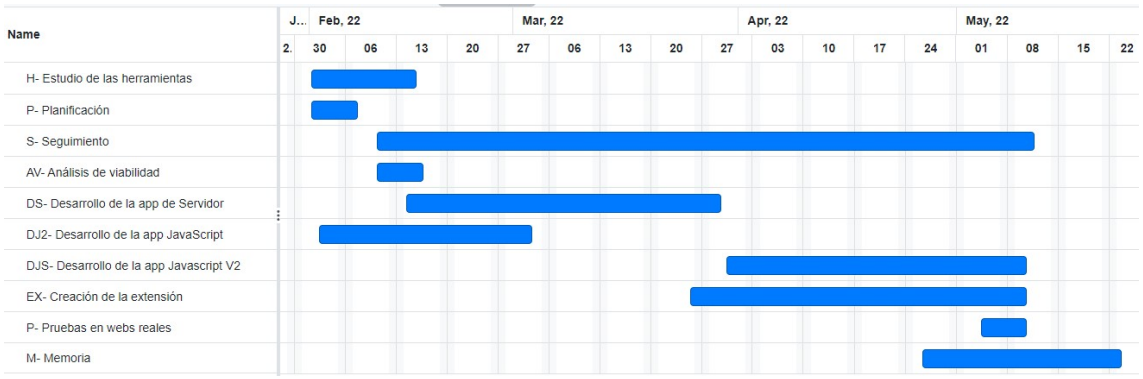

Figura 2.2: Diagrama de Gantt del proyecto

Se espera que la tecnología esté terminada para el día 7 de mayo y la memoria para el día 21.

2.7. Herramientas utilizadas

Para la realización de la tecnología, así como para la realización de la memoria se han usado muchas herramientas. Las herramientas usadas han sido las siguientes:

2.7.1. GlooMaps

GlooMaps [glo, 2018] es una página web completamente gratuita que permite la creación de diagramas EDT. Ha sido la web usada para crear la EDT de esta memoria (figura 2.1).

2.7.2. Online Gantt

Online Gantt [onl, 2022] es una página web gratuita con la que se pueden crear diagramas de Gantt de forma muy sencilla. Ha sido la herramienta usada para crear el diagrama de Gantt de esta memoria (figuras 2.2, 7.1).

[3]El diagrama de Gantt es una herramienta gráfica cuyo objetivo es exponer el tiempo de dedicación previsto para diferentes tareas o actividades a lo largo de un tiempo total determinado

2.7.3. Diagrams

Diagrams.net [dia, 2021], anteriormente *draw.io*, es una web gratuita y realmente potente que te permite realizar diagramas de todo tipo. Es la herramienta usada para hacer los diagramas de la arquitectura en la sección 4.1.

2.7.4. Google Fonts

Google Fonts [fon, 2020] es una librería de fuentes de texto y iconos que ofrece *Google* de forma completamente gratuita. De esa librería, se han obtenido los iconos usados para hacer los diagramas de arquitectura así como los iconos de las flechas que se usan en la extensión.

2.7.5. Overleaf

Overleaf [ove, 2011] es una página web que permite escribir documentos en Latex de forma sencilla y cómoda. Se podría considerar una herramienta gratuita ya que, si bien tiene una versión "PRO" que permite vincular el documento con *GitHub* o *Dropbox*, se puede escribir y descargar un documento sin ningún tipo de inconveniente. Ha sido la herramienta seleccionada para escribir la memoria.

2.7.6. Python

Python [pyt, 1991] es un lenguaje interpretado muy popular, siendo uno de los lenguajes de programación más utilizados en todo el mundo. Su popularidad se basa en la legibilidad del código, que hace que sea muy fácil de programar todo tipo de aplicaciones, en especial aplicaciones de servidor. Python ha sido mi elección para el lenguaje de la aplicación de servidor por las dos razones que he mencionado, la sencillez y la popularidad.

La popularidad es un aspecto importante porque, cuanto más popular es una herramienta, más soporte tiene y más herramientas adicionales se pueden usar sobre él. En mi caso, las herramientas adicionales, llamadas comúnmente librerías, son una pieza fundamental de mi proyecto, en concreto, las que menciono a continuación.

2.7.7. Flask

Flask [fla, 2010a] es una framework basado en Python, gratuito, que permite crear aplicaciones web de manera rápida y sencilla. Esto es algo que me viene perfectamente ya que la creación de la web para la aplicación de servidor no es más que un aspecto secundario del proyecto. Esto se debe a que lo importante es el trabajo de scraping que realiza la aplicación web, y no cómo se genera esa web. Además, la aplicación web no tiene un comportamiento web como tal, sería mas bien una API. Es necesario que sea una aplicación web para que se pueda conectar con la extensión, pero el usuario no abrirá en ningún momento dicha aplicación web.

2.7.8. Flask_cors

Flask_cors [cor, 2022] es una extensión para Flask, independiente de la aplicación, que sirve para que se pueda realizar el CORS [4] a la aplicación web. Esto será algo fundamental y necesario ya que mi extensión pedirá el recurso a mi aplicación web desde un dominio distinto. Si no activáramos el CORS, aparecería el mensaje de la figura 2.3 cuando la extensión pidiera el recurso.

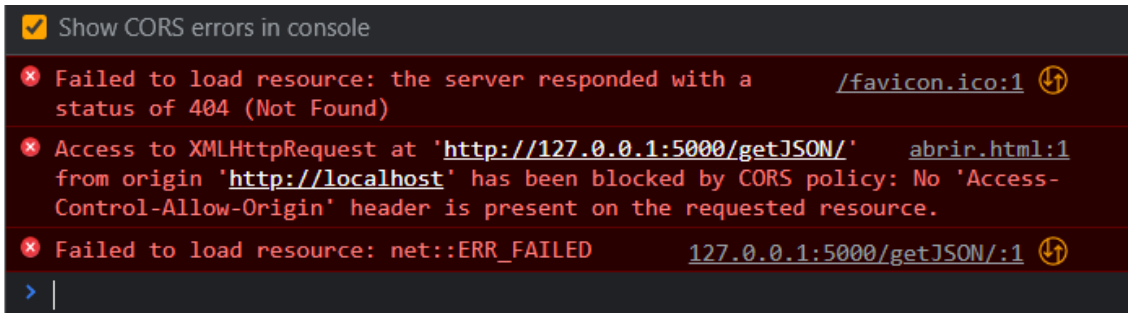

Figura 2.3: Error al cargar el JSON de resultado, al no estar activado el CORS

2.7.9. Concurrent.futures

Concurrent.futures [par, 2022] es un módulo de *Python* que permite ejecutar funciones de manera concurrente. El uso de este módulo hará que la eficiencia del programa mejore de manera notable en lo que al tiempo de procesado respecta.

[4] Cross Origin Resource Sharing (CORS): Es un mecanismo que permite que se puedan solicitar recursos restringidos en una página web desde un dominio diferente al propio

A la hora de devolver el JSON resultante a la extensión desde la aplicación de servidor, se realizarán dos análisis desde dos analizadores distintos, uno por cada analizador. Esos análisis se pueden realizar de forma paralela ya que no dependen el uno del otro. Esto es algo que interesa notablemente porque los tiempos de carga de las páginas web son elevados (unos 20s para *AccessMonitor*[5] y unos 40s para *AChecker*[6])[7].

2.7.10. BeautifulSoup

BeautifulSoup [bea, 2007] será una de las herramientas más importantes ya que será la encargada de hacer el scraping de las páginas web. BeautifulSoup es una biblioteca de Python para analizar páginas web HTML, creando un árbol con los elementos del documento y pudiendo navegar a través de él para obtener información de manera cómoda.

Será usado para obtener los resultados del análisis que realizan las páginas web *AccessMonitor* y *AChecker* para poder así procesar los resultados con el fin de crear posteriormente el fichero JSON que será devuelto desde la aplicación de servidor.

2.7.11. Selenium Webdriver

Selenium Webdriver [sel, 2022] es un software de automatización de pruebas que permite realizar pruebas sin necesidades de usar un lenguaje de scripting. Será la herramienta más importante para la parte de la aplicación de servidor ya que será la encargada de navegar por los analizadores de accesibilidad para poder obtener la información necesaria para la creación del JSON.

Además de Selenium, fue necesario el uso de *ChromeDriver* y *WebDriverWait*, detallados en las siguientes subsecciones 2.7.12 y 2.7.13.

2.7.12. Chromedriver

Selenium necesita un driver para simular el navegador. Ese driver puede ser de distintos navegadores (Chrome, Firefox, Edge...). En mi caso, como la extensión será de Google Chrome, he elegido el driver de Chrome. Selenium usará el módulo *Chromedriver* [chr, 2019] para navegar sobre los enlaces.

[5] https://accessmonitor.acessibilidade.gov.pt/
[6] https://achecker.achecks.ca/checker/index.php
[7] Tiempo de respuesta para web de ejemplo: https://www.ehu.eus/es/home

Es importante saber que ese driver que se descarga desde el repositorio oficial de Google [chr, 2022] tiene que estar ubicado en la carpeta del proyecto para poder ser cargado. Además, **la versión del driver dependerá directamente de la versión del navegador**, teniendo que coincidir para el correcto funcionamiento del driver. Aunque no habrá problemas de versión, ya que el driver se instalará de manera automática (ver sección 5.3.10, Funciones destacadas: Autoinstalación de *Chromedriver*).

2.7.13. WebDriverWait

La clase WebDriverWait [wai, 2022] de *Python* es fundamental porque permite al webdriver esperar a que un elemento se haya cargado completamente antes de empezar a navegar por la página. Esto es necesario porque no podemos saber cuánto tardará el analizador en realizar el análisis y mostrar los resultados, y no podemos navegar hasta que los resultados estén mostrados.

Al mostrar las páginas de análisis los resultados mediante JavaScript, el scraper creería que los resultados están en la página tan pronto como la página se haya cargado, algo lejos de la realidad. La página se carga completamente pero en el interior de la página aparece un gif de carga hasta que finalmente se añaden los datos mediante JavasScript.

2.7.14. Sphinx

Sphinx [sph, 2008] es un software gratuito que, usando documentación que ha realizado el usuario en las funciones, genera una documentación en HTML con la que un usuario podría ver un índice de las funciones, su documentación y código de manera cómoda y agradable. Ha sido la herramienta usada para realizar la documentación del código escrito en Python.

2.7.15. JavaScript

JavaScript [js, 1995] es un lenguaje de programación interpretado y orientado a objetos, Es uno de los lenguajes de programación más usados del mundo, siendo el séptimo más usado[8]. Es aún más usado en el entorno de la programación web ya que el uso de JavaS-

[8] https://www.tiobe.com/tiobe-index/

cript proporciona dinamismo a las webs haciendo que la experiencia del usuario sea más reconfortante. La extensión de Chrome se desarrollará en su totalidad en JavaScript.

2.7.16. JQuery

JQuery [jqu, 2006] es una biblioteca de *JavaScript* que simplifica la interacción con los documentos HTML, la manipulación del árbol DOM y sobre todo el manejo de eventos, algo que necesitamos para nuestro proyecto. Usaremos JQuery para la respuesta a eventos como click de botones o la subida de ficheros a la extensión.

2.7.17. Funciones de las extensiones de Chrome

Las extensiones de Chrome tienen funciones de JavaScript predefinidas por el navegador que permiten a la extensión tener variables "globales", que son independientes de la página web y que se guardan en el navegador. Serán necesarias para el correcto funcionamiento de la extensión.

2.8. Gestión de riesgos

Todos los proyectos pueden verse envueltos en problemas que impidan la realización del mismo. A esos problemas posibles se les denominan riesgos. Los riesgos que se han identificado que pueden hacer peligrar la finalización del proyecto son los siguientes:

- **R1- Modificación de la estructura de las webs analizadoras**: El generador automático de los informes obtiene datos de dos páginas web, *AccessMonitor* y *AChecker*, mediante el uso de scraping. El uso de scraping trae consigo un riesgo muy grande y es que el código que hace el scraping funciona únicamente para una estructura de web concreta. Esto se debe a que para hacer el scrapìng la aplicación se desplaza a través de elementos concretos y lee otros elementos únicos. Si la página web cambia la estructura (cambia la ubicación de los elementos en el árbol DOM, modifica el id de los elementos...) el scraper podría dejar de funcionar.

  Poco se puede hacer para mitigar este riesgo, ya que es un riesgo que viene intrínseco con el uso del scraping. Lo máximo que se puede hacer es tratar de fijar los elementos lo máximo posible (por uso de atributos 'id' y no por ubicación) y dejar

el código bien documentado para que en caso de materializarse el riesgo se pudiera solucionar lo antes posible.

- **R2- Perdida de información**: Uno de los riesgos más probables es que debido a la pérdida o rotura del dispositivo en el que se realiza el proyecto se perdiera información. Sería algo que causaría un retraso severo respecto a la planificación.

  Por suerte, la mitigación de este riesgo es muy fácil de realizar. Se realizará una copia periódica de todo el código cada vez que se realice un proceso significativo o hayan pasado 3 días desde la última copia de seguridad, lo que llegue antes. La copia de seguridad se realizará mediante el uso de *Git* y se almacenará en la web de *GitHub*. La copia de seguridad no tarda más de 20 segundos, haciendo que no resulte ningún inconveniente su realización periódica.

  La memoria del proyecto se realizará online, mediante el uso de la aplicación de *Overleaf*, con lo que estará a salvo de cualquier pérdida de información.

- **R3- Problemas de carácter personal**: Al tener el proyecto una duración tan larga, es posible que en esos meses tuviera algún problema de personal (e.g. un accidente de tráfico) que me imposibilitase la realización del proyecto con normalidad.

  Si se diera, se intentaría recuperar el tiempo perdido en las semanas venideras y si no fuera posible se realizaría una nueva planificación.

- **R4- Problemas académicos**: La realización de la planificación se ha realizado sin saber cómo serán las asignaturas en las que estoy matriculado en el 4º curso. Podría suceder que necesitara invertir más tiempo de lo esperado en asignaturas en las que estoy matriculado. Esto podría suponer un retraso en cascada en todas las previsiones de fechas de las tareas.

  El R4 no supone un riesgo notable, porque las asignaturas del 4º curso estarán terminadas para mediados de mayo y el TFG no se entrega hasta finales de Junio con lo que tendría, si fuera necesario, mes y medio para realizar un "plan intensivo" del proyecto.

- **R5- Problemas tecnológicos**: A lo largo del proyecto se van a usar muchísimas tecnologías y eso supone un riesgo porque puede ser que alguna de las tecnologías produzcan fallos. Eso podría suponer invertir más tiempo de lo planeado en alguna parte del desarrollo.

  Este riesgo viene mitigado desde la planificación en dos aspectos. El primero es que se han seleccionado herramientas muy populares, lo que supone que tendrá

mucha documentación así como soporte a problemas comunes en foros, con lo que no resultaría tedioso buscar la solución para algún fallo en concreto. El segundo, la planificación está hecha para terminar pronto el proyecto de modo que se podría aceptar que hubiera retrasos con alguna tecnología, porque seguiría entrando en plazos.

2.9. Sistemas de información

El sistema de información para el código será doble. Primero, se almacenará en el disco duro de mi ordenador en la carpeta "TFG_CODE". Además, tal y como se ha mencionado en el riesgo R2 del apartado 2.8, se realizará una copia de seguridad periódica mediante el uso del servicio *GitHub*.

Además de *GitHub*, periódicamente se harán copias de seguridad de versiones "estables" a otra carpeta del ordenador de manera que si se avanza en una versión y el programa deja de funcionar, se podrá recuperar la última versión estable.

El sistema de información para la memoria será, tal y como se ha mencionado en el riesgo R2, la famosa página de *Overleaf*. Además, cada cierto tiempo, se realizarán descargas de la memoria para llevar un control de versiones.

2.10. Interesados del proyecto

El principal interesado del proyecto sería yo, Mikel Iturria Silveti, ya que este proyecto será evaluado como mi trabajo de fin de grado y de él depende mi graduación como Ingeniero Informático especializado en Software.

Otro interesado sería Juan Miguel López Gil, director del proyecto, ya que ha sido él quién me propuso este proyecto como trabajo de fin de grado y querrá que el proyecto se finalice de manera correcta.

Por último, el tribunal que se encargue de evaluar mi proyecto también se considerará como interesado.

2.11. Alcance del proyecto

El alcance del proyecto es conseguir una herramienta funcional en forma de extensión de *Google Chrome*. La extensión tendrá funciones que ayudarán al desarrollador a comprobar la accesibilidad de la página web. Las funciones serán:

- Generación automática de informes mediante agregación de información de diferentes analizadores en la aplicación de servidor.
- Agregación manual de informes.
- Almacenamiento de informes y mostrado de resultados de los mismos.
- Marcado de resultados.
- Descarga del informe almacenado (agregación automática o manual) en el formato aceptado por la *Report Tool* de W3C[9].

Las **exclusiones**, características que no están incluidas en el alcance, serían:

- El usuario no podrá seleccionar cualquier analizador automático. Solo podrá elegir entre *AccessMonitor* y *AChecker* o escoger ambas.
- El usuario no podrá editar los resultados del informe. Si el usuario desea editarlo manualmente tendría que descargar el informe, editarlo y volver a subirlo.

Respecto a lo que concierne a los **entregables** del proyecto, realmente solo habrá un entregable obligatorio, la memoria del proyecto. También se podrían considerar entregables el código de la extensión y el código de la aplicación de servidor o API.

La licencia de la tecnología será, en un principio, **Creative Commons BY-CC-SA**.

[9] https://www.w3.org/WAI/eval/report-tool/

3. CAPÍTULO

Accesibilidad web

En este capítulo se relatará, de manera breve, qué es la accesibilidad web, los fundamentos de la evaluación de la accesibilidad, cómo se evalúa que una web sea accesible y los problemas que puedan surgir en la evaluación de una página web.

3.1. Definición de accesibilidad web

La accesibilidad web [W3C, 2022b] tiene como finalidad que el mayor número de personas puedan navegar a través de la web, independientemente de sus capacidades físicas y mentales. Un claro ejemplo de accesibilidad es que las imágenes de la web tengan texto alternativo para que así cuándo una persona invidente o de visión reducida use herramientas de ayuda como el *text-to-speech*[1], la herramienta narre, usando el sonido, lo que aparece en la imagen.

Como la tecnología no ha avanzado lo suficiente -al menos a día de la publicación de este TFG- como para saber a ciencia cierta que representa una imagen, es necesario que un ser humano lo especifique. Esto suele ser un trabajo realizado por el desarrollador web si la imagen es estática y siempre es la misma.

Pero no siempre es así, en webs de noticias por ejemplo, es el periodista el que especifica el texto alternativo de la foto cuando la selecciona para el artículo, que suele coincidir con el pie de foto. Incluso la red social *Twitter* ha realizado una actualización en la que permite que los propios usuarios, al añadir una foto, puedan poner texto alternativo.

Figura 3.1: Descripción de Twitter para añadir texto alternativo a imágenes.

El ejemplo del texto alternativo es uno de los criterios del *WGAC*, cuya definición se detalla en la siguiente sección. En concreto, se trata del criterio *1.1.1: Non-text Content*.

[1] *Text-to-speech*, también conocido como *TTS*, es una herramienta que convierte el lenguaje de texto normal en sonido para que el usuario del software pueda escuchar el texto en vez de tener que leerlo.

3.2. Evaluación de la accesibilidad web

Para saber si se cumple o no la accesibilidad en las páginas web, será necesario realizar una evaluación. La web de *W3C* [W3C, 2022a] recomienda que durante el desarrollo de la web se realicen análisis periódicos de la accesibilidad, ya que es cuando es más fácil arreglar los errores. Aún así, la accesibilidad es algo que se debe comprobar durante todo el ciclo de vida del proyecto, porque puede que haya novedades en los criterios o nuevos complementos, como el *TTS* mencionado anteriormente, que ayuden a hacer las webs más accesibles.

Existen herramientas de evaluación que ayudan a realizar dichos análisis de accesibilidad. Aunque resultan de muchísima ayuda, para afirmar que un sitio es accesible se requiere una evaluación humana con conocimiento de causa ya que ninguna herramienta puede determinar por sí sola si un sitio cumple las normas de accesibilidad. Esto es algo sobre lo que se hablará en la siguiente sección, Problemática de la evaluación.

Como hemos mencionado, hace falta una cooperación herramienta-persona para lograr realizar un análisis de manera correcta. Ambos necesitan un "diccionario" con en el que especificar qué es un error y qué no lo es y cómo deben llamar a un error. Por ejemplo, el humano podría decir que hay un error porque hay imágenes sin atributo `alt`, la herramienta decir que hay un error por falta de texto alternativo y ninguno de los dos ser conscientes de que ambos están hablando de lo mismo.

Por eso y por muchas más razones, se creó la *Web Content Accessibility Guidelines* (WCAG) [W3C, 2022d], cuya última versión, la 2.1 [W3C, 2022f] publicada en el 2018 es la que hemos usado de base para nuestra herramienta. En ella, se detallan qué criterios son considerados como necesarios para que se pueda afirmar que la web es totalmente accesible. Esos criterios están dentro de subsecciones que a su vez estarían agrupadas por secciones dependiendo de con qué estuvieran relacionados.

Siguiendo el ejemplo anterior, que las imágenes no tengan texto alternativo sería un error en el criterio *1.1.1 Non-text Content* (contenido sin texto), de la subsección *1.1 Text Alternatives* (texto alternativo) de la sección *1. Perceivable* (Perceptible), en el que se agrupan los criterios que garantizan que la información y los componentes de la interfaz se presentan a los usuarios de manera que puedan percibirlos.

Ahora que tenemos unos estándares universales que nos hacen saber qué cosas se pueden considerar como un error de accesibilidad y tenemos la forma de que las herramientas de accesibilidad nos den información que podamos clasificar, solo quedará definir una "conformancia" correcta, lo que incluye la definición del alcance, la elección de la muestra, la evaluación y la información de los resultados, entre otros [W3C, 2022e].

3.3. Problemática de la evaluación

Ya hemos visto lo que es el *WCAG* y en qué se basa la evaluación de accesibilidad. Ahora hablaremos del cómo evaluar y los diferentes inconvenientes que pueden surgir.

Un análisis manual de accesibilidad sería una de la formas posibles de realizar el informe de accesibilidad. Coger la *WCAG* e ir criterio a criterio comprobando si se cumplen o no, pero hoy en día es muy poco usual realizarlo manualmente por el excesivo número de horas necesarias. Además, es fácil que con tantos criterios y tantos elementos de la web que comprobar el evaluador pueda no percatarse de algún error. Para ser conscientes de la cantidad de criterios que contiene el *WCAG*, ver las tablas 3.1, 3.2 y 3.3 ubicadas unas páginas mas abajo. La información sobre el criterio/directriz/estándar se podrá consultar clicando el enlace del nombre.

Otra opción sería realizar un análisis puramente automático. Pero, la opinión del W3C es que, cito textualmente *"No tool alone can determine if a site meets accessibility standards. Knowledgeable human evaluation is required to determine if a site is accessible."* [W3C, 2022a]. Ello implica que ninguna herramienta puede determinar si una web cumple los estándares de accesibilidad y se necesitará la ayuda de una persona para determinar el resultado. Esto se debe a que un evaluador automático puede tener falsos positivos o incluso falsos negativos sobre criterios porque, pese a que el código diera a intuir que podía haber un error, el contexto de la página hace que no sea así.

Con lo que, como hemos mencionado en la sección anterior, se deberá realizar una cooperación herramienta-persona para realizar los análisis de accesibilidad. El uso de herramientas hace que el análisis sea más ameno para el desarrollador, que necesitará mucho menos tiempo para realizar un análisis y disminuirá la posibilidad de cometer descuidos en la lectura. A su vez, que haya un desarrollador detrás del informe hará que el evaluador automático no se aventure a tomar decisiones precipitadas y simplemente marque el código que sospecha que pueda provocar errores y que sea el desarrollador quien tome la decisión final.

Sabemos que es necesaria la cooperación persona-herramienta, pero, ¿hasta qué punto es necesaria? Numerosos estudios, tales como [Vigo et al., 2013], respaldan que cuanto mayor sea el número de analizadores, más completo será el informe, ya que lo que un analizador no haya comprobado lo ha podido comprobar el otro. No es la única referencia de dicha afirmación, citando al propio *W3C*, *"Use of a variety of evaluation tools for Web site accessibility to ensure that they have access to a broad range of evaluation tools across the group as a whole."* [W3C, 2022c]. Recomiendan la utilización de diversas herramientas de evaluación para que tengan acceso a más herramientas y poder así considerar al grupo como un conjunto de resultados mucho más fiable.

Es por la afirmación anterior por lo que cobra mucho sentido uno de los aspectos más importantes de este TFG, la **agregación de analizadores automáticos para garantizar la fiabilidad.** Pero la agregación no es solo importante para las herramientas automáticas, también vale para los informes manuales. La premisa es que, a más analizadores, ya sean herramientas automáticas o personas, más fiable será el resultado.

Por eso es importante también **la agregación manual de informes**, que también facilita la herramienta que se ha desarrollado en este proyecto. Que el trabajo de dos personas se pueda agregar -y en consecuencia, ganar fiabilidad- en cuestión de segundos, sin perder ningún tipo de información y actualizando datos, hace que la calidad de vida del desarrollador mejore notablemente.

Si además dicha herramienta facilita la colaboración herramienta-persona, que es la finalidad de este proyecto, facilitando el análisis, lectura y edición de los datos automáticos para ser manejados por el usuario, podríamos afirmar que la herramienta que ha desarrollado este proyecto es útil.

Nombre	Tipo
1. Perceptible	Estándar.
1.1. Texto alternativo	Directriz.
1.1.1 Contenido no textual	Criterio.
1.2. Contenido multimedia dependiente del tiempo	Directriz.
1.2.1 Sólo audio y sólo vídeo (grabado)	Criterio.
1.2.2 Subtítulos (grabados)	Criterio.
1.2.3 Audiodescripción o Medio Alternativo (grabado)	Criterio.
1.2.4 Subtítulos (en directo)	Criterio.
1.2.5 Audiodescripción (grabado)	Criterio.

Tabla 3.1: Criterios de la versión 2.1 de WCAG (parte 1)

Nombre	Tipo
1.3. Adaptable	Directriz.
1.3.1 Información y relaciones	Criterio.
1.3.2 Secuencia significativa	Criterio.
1.3.3 Características sensoriales	Criterio.
1.3.4 Orientación	Criterio.
1.3.5 Identificar propósito del input	Criterio.
1.4. Distinguible	Directriz.
1.4.1 Uso del color	Criterio.
1.4.2 Control del audio	Criterio.
1.4.3 Contraste (mínimo)	Criterio.
1.4.4 Cambio de tamaño del texto	Criterio.
1.4.5 Imágenes de texto	Criterio.
1.4.10 Reflow	Criterio.
1.4.11 Contraste del no texto	Criterio.
1.4.12 Espaciado del texto	Criterio.
1.4.13 Contenido en “Hover” o “Focus”	Criterio.
2. Operable	Estándar.
2.1. Teclado accesible	Directriz.
2.1.1 Teclado	Criterio.
2.1.2 Sin trampas para el foco del teclado	Criterio.
2.1.4 Atajos de teclado	Criterio.
2.2. Tiempo suficiente	Directriz.
2.2.1 Tiempo ajustable	Criterio.
2.2.2 Poner en pausa, detener, ocultar	Criterio.
2.3. Ataques epilépticos	Directriz.
2.3.1 Umbral de tres destellos o menos	Criterio.
2.4. Navegación	Directriz.
2.4.1 Evitar bloques	Criterio.
2.4.2 Titulado de páginas	Criterio.
2.4.3 Orden del foco	Criterio.
2.4.4 Propósito de los enlaces (en contexto)	Criterio.
2.4.5 Múltiples vías	Criterio.
2.4.6 Encabezados y etiquetas	Criterio.
2.4.7 Foco visible	Criterio.
2.5. Modalidades de entrada	Directriz.
2.5.1 Gestos del puntero	Criterio.
2.5.2 Cancelación del puntero	Criterio.
2.5.3 Etiqueta en nombre	Criterio.
2.5.4 Actuación del movimiento	Criterio.

Tabla 3.2: Criterios de la versión 2.1 de WCAG (parte 2)

Nombre	Tipo
3. Comprensible	Estándar..
3.1. Legible	Directriz.
3.1.1 Idioma de la página	Criterio.
3.1.2 Idioma de las partes	Criterio.
3.2 Previsible	Directriz.
3.2.1 Al recibir el foco	Criterio.
3.2.2 Al recibir entradas	Criterio.
3.2.3 Navegación coherente	Criterio.
3.2.4 Identificación coherente	Criterio.
3.3. Asistencia a la entrada de datos.	Directriz.
3.3.1 Identificación de errores	Criterio.
3.3.2 Etiquetas o instrucciones	Criterio.
3.3.3 Sugerencias ante errores	Criterio.
3.3.4 Prevención de errores (legales, financieros, datos)	Criterio.
4. Robustez	Estándar.
4.1 Compatible	Directriz.
4.1.1 Procesamiento	Criterio.
4.1.2 Nombre, función, valor	Criterio.
4.1.3 Mensajes de estado	Criterio.

Tabla 3.3: Criterios de la versión 2.1 de WCAG (parte 3)

4. CAPÍTULO

Arquitectura, casos de uso y diseño del proyecto

En este capítulo se detallará la arquitectura del proyecto y los casos de uso, ya sea para la obtención del informe como para el marcado de resultados o otras funciones secundarias. Además, se explicará el diseño del producto, que incluye el diseño de la propia extensión, la selección de nombre y logos, el diseño de los componentes, la paleta de colores escogida y el resultado final.

4.1. Arquitectura y casos de uso del proyecto

La arquitectura es un aspecto importante de este proyecto porque vamos a crear dos aplicaciones que se van a estar mandando información a través de la web.

4.1.1. Obtención de informe

Primero, veremos el diagrama de secuencia para la generación de un informe automático y posteriormente lo explicaremos.

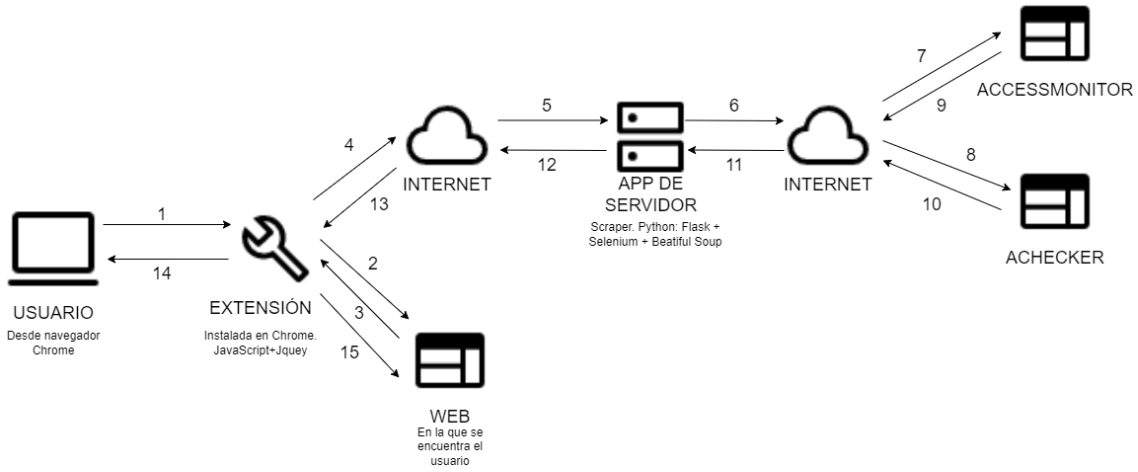

Figura 4.1: Diagrama de secuencia para la obtención de informes

Como se puede observar en la figura, por cada flecha hay un número. Ese número, además de estar ordenado temporalmente, expresa un proceso. El proceso es el siguiente.

1. El usuario selecciona los “checkbox” de los analizadores automáticos que quiere usar para realizar el informe y hace click en “get automatically generated report” desde la extensión.

2. La extensión solicita al navegador la URL de la web en la que está el usuario actualmente.

3. El navegador devuelve la URL.

4. La extensión solicita a través de internet un informe para la URL dada.

5. Internet remite ese mensaje a la aplicación de servidor.

6. La aplicación de servidor se comunica con internet para hacer un proceso de scraping.

7. Este apartado es opcional, solo si el usuario ha seleccionado que se use el analizador *AccessMonitor*. Si se cumple, se navega en *AccessMonitor* para obtener el informe de la URL dada.

8. Este apartado es opcional, solo si el usuario ha seleccionado que se use el analizador *AChecker*. Si se cumple, se navega en *AChecker* para obtener el informe de la URL dada.

9. Solo si se ha realizado el proceso 7. *AccessMonitor* devuelve el contenido resultante.

10. Solo si se ha realizado el proceso 8. *AChecker* devuelve el contenido resultante.

11. Se comunican desde internet con la aplicación de servidor. En la aplicación del servidor, se analizan los datos y se prepara un informe con formato W3C.

12. La aplicación de servidor manda el informe a internet para que llegue a la extensión,

13. Internet remite el informe a la extensión.

14. La extensión recibe el informe, lo analiza, crea los resultados y finalmente se los muestra al usuario.

15. Por último, la extensión se comunica con la web para guardar en la memoria local de la web los datos recibidos en bruto y los datos procesados.

4.1.2. Marcado de elementos

El marcado de elementos se produce cuándo el usuario quiere que se marque en la web el elemento HTML que coincide con el código de un error o advertencia en los resultados obtenidos. El diagrama (fig. 4.2) se encuentra en la página siguiente.

El proceso se detalla a continuación:

1. La extensión solicita a la web la información que se ha guardado en su memoria local.

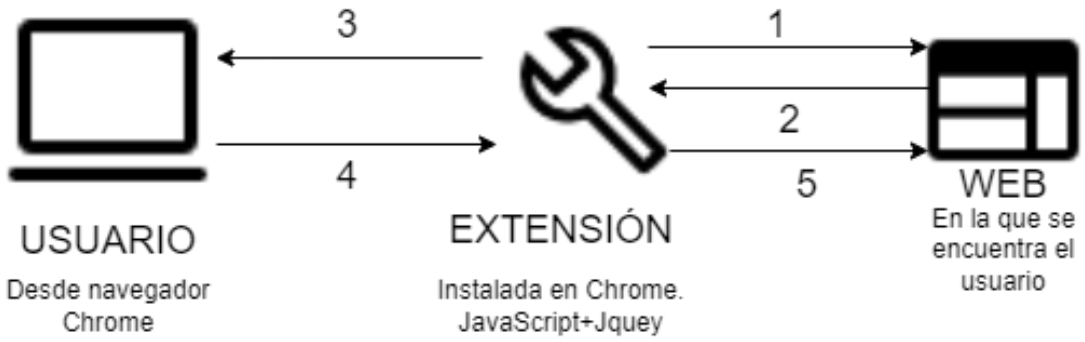

Figura 4.2: Diagrama de secuencia para el marcado de resultados

2. La web devuelve a la extensión la información que tenía almacenada en su memoria local.

3. La extensión muestra al usuario los resultados que estaban almacenados para la página web en la que se encuentra.

4. El usuario hace click en un trozo de código de los resultados.

5. La extensión busca el trozo de código en la página web original. Primero, comprueba si hay algún elemento marcado, si lo hay, lo desmarca. Después, una vez haya encontrado el elemento que desea marcar el usuario, lo modifica en la página web y lo marca añadiendole un borde rojo. Además, guarda el elemento como último elemento marcado para que se pueda desmarcar en el próximo proceso de marcado.

NOTA: Los pasos 1, 2 y 3 de esta subsección se repiten en todas las subsecciones que se explican más abajo, ya que la extensión pide siempre los datos almacenados al navegador nada más se haya cargado la página e independientemente de la funcionalidad que vaya a realizar el usuario.

4.1.3. Limpieza de datos

El usuario podrá borrar los datos almacenados sobre la web en la que se encuentra con hacer un click al botón. La figura se muestra en la siguiente página.

El proceso es el siguiente:

1. El usuario hace click en el botón de “Clean stored data”.

2. La extensión elimina los datos guardados en la memoria local de la web.

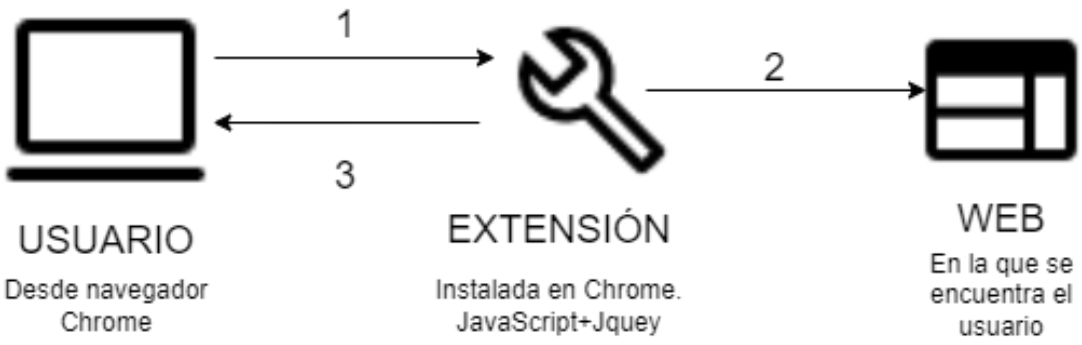

Figura 4.3: Diagrama de secuencia para la limpieza de datos

3. La extensión emite una alerta informando al usuario de que los datos han sido borrados y recarga la página.

4.1.4. Descargar informe

El usuario podrá también descargar el informe con formato W3C almacenado en la web en la que se encuentra haciendo click sobre un botón.

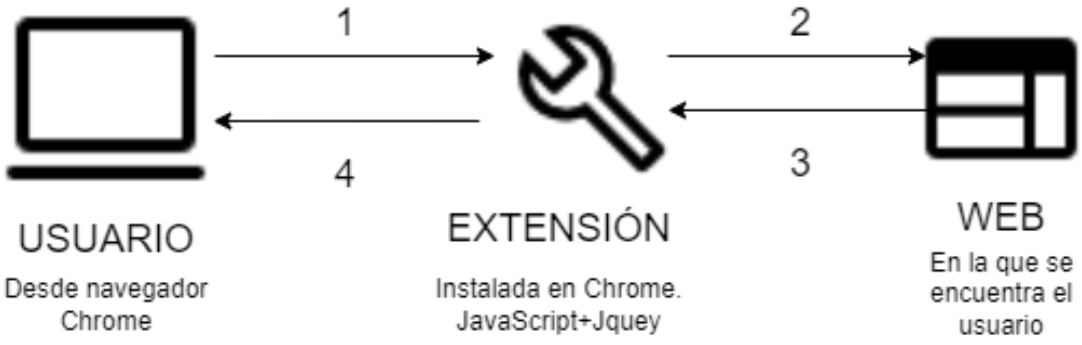

Figura 4.4: Diagrama de secuencia para la descarga del informe

El proceso es el siguiente:

1. El usuario hace click en el botón de “Download report”.
2. La extensión solicita el informe con formato W3C guardado en la memoria local de la página web.
3. La página web devuelve el informe a la extensión.
4. La extensión crea un fichero con la información recibida y lo descarga.

4.1.5. Subida y agregación de informe

El usuario podrá subir a la extensión un informe en formato W3C y la extensión agregará los datos de ese informe a los datos que estén almacenados, creando un nuevo informe con la información agregada.

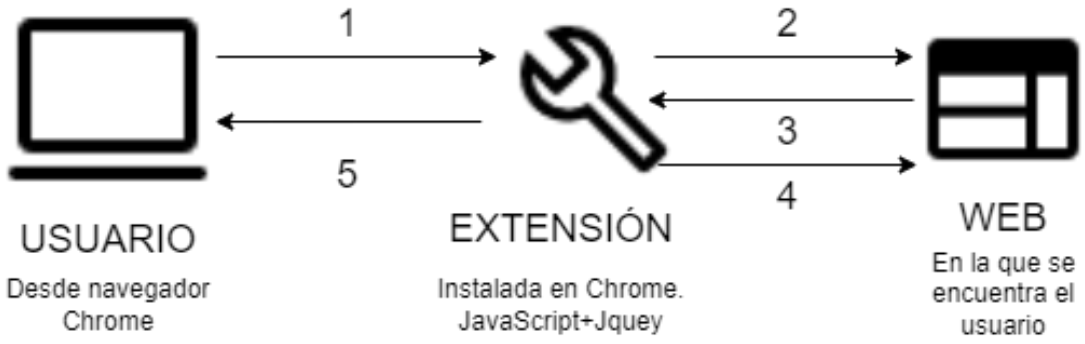

Figura 4.5: Diagrama de secuencia para la subida de informes

El proceso es el siguiente:

1. El usuario sube un informe en formato W3C a la extensión.
2. La extensión solicita el informe almacenado en memoria a la página web
3. La extensión recibe el informe almacenado. Una vez tiene los dos informes, analiza el informe subido, lo agrega al informe guardado y crea un nuevo informe con la información agregada.
4. La extensión guarda el nuevo informe en la web.
5. Por último, el usuario es notificado mediante una alerta de que se ha realizado la agregación.

4.2. Diseño del producto

La extensión, al ser algo con la que el usuario de la misma tendrá que interactuar, deberá tener una interfaz gráfica que sea agradable a la vista y sea sencilla para que cualquier usuario con conocimientos básicos de la informática pueda usarla. En esta sección se detallarán las decisiones de diseño.

4.2.1. Diseño de la extensión

La extensión tendrá un diseño de barra lateral o *sidenav*. Este diseño es comunmente usado en páginas de renombre como podrían ser en cualquier sección de *W3Schools*, como por ejemplo en la sección de HTML[1] o incluso la propia página de *Overleaf* en la que estoy realizando la memoria usa este tipo de barras laterales.

La extensión tendrá dos modos, el modo en el que no tiene ningún dato almacenado y el modo en el que tiene datos almacenados. La diferencia entre los modos se dará en la parte de abajo de la extensión. Si no hay datos, aparecerá un mensaje notificándolo y si los hubiera, se mostraría el resultado.

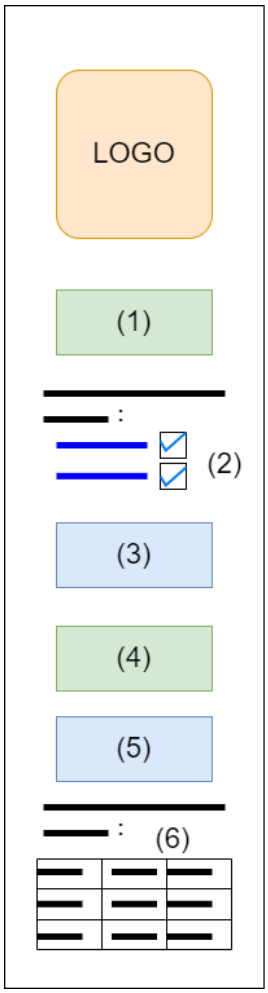

Figura 4.6: Boceto del diseño de la extensión

El boceto básico sería la figura 4.6. Como se pueden apreciar, en el boceto hay unos números que serán elementos de la extensión. Los elementos serán los siguientes:

1. Será el botón de borrar datos. Cuando el usuario haga click a ese botón la extensión borrará los datos que se encuentren almacenados en memoria para esa página web.
2. El elemento 2 serán los "checkbox" de los analizadores automáticos que el usuario quiera que se usen para realizar el análisis automático. En azul, estará los enlaces a los analizadores, para que el usuario pueda visitarlos.
3. Este elemento será el botón para la obtención automática del informe. Cogerá los datos de los "checkbox".
4. El cuarto elemento será el "input" de tipo "file" donde se podrá subir un informe de manera manual,
5. El quinto será un botón que, al hacer click sobre él, descargará el informe.
6. Por último, en el sexto elemento se mostrarán los datos almacenados en memoria. Si no los hubiera, se mostrará un mensaje indicando que no existe información almacenada.

[1] https://www.w3schools.com/html/default.asp

4.2.2. Selección de nombre y logos

Toda extensión tiene que tener un nombre y un logo para poder ser identificada y mi extensión no iba a ser una excepción.

Para el nombre, decidí llamarla **AC-Check**. Los motivos por los que la decidí fueron porque incluía la "A" mayúscula por ***A**ccessmonitor*, la "AC" de ***AC**hecker* y la palabra "check"[2] que representa lo que hace la función, *comprobar* si los estándares de accesibilidad se están cumpliendo. Además, es corto, conciso y, subjetivamente hablando, suena bien.

El logo fue obtenido de la web de *canva.com* y se llama "Blue Modern Letter C Tech Logo Design". La autora es *FlyBets* y la licencia del logo es de libre uso, edición y publicación. La elegí porque me gusto la combinación de colores, porque tiene un aire tecnológico y tiene una 'C', letra que incluye el nombre.

A continuación se encuentran el logo así como los iconos (tamaño 16x16) que se usarán para notificar si la extensión está apagada o está encendida.

Figura 4.7: Logo de la extensión

Figura 4.8: Icono de extensión encendida

Figura 4.9: Icono de extensión apagada

4.2.3. Diseño de los componentes

Los componentes que formarán parte de la extensión y que hemos visto en la figura del boceto (4.6) deberán tener un diseño. Las decisiones de diseño se relatan a continuación:

[2]check: (*verbo*) Comprobar en inglés

Logo

El logo no sufrirá muchos cambios para ser mostrado en la extensión. El único cambió destacable será que se le redondearán las esquinas para hacerlo más agradable para el usuario.

La clase de logo es la siguiente:

Figura 4.10: Versión mostrada del logo

```
.clase_logo{
  height: 128px;
  margin-left: 32px !important;
  border-radius: 30px !important;
  padding: 8px 12px 8px 12px !important;
  display: inline-flex !important;
}
```

Botones

Para el diseño de los botones se han tomado como referencia los botones del *Report Tool* la página web de W3C ya que será a la página a la que se podrán subir los informes una vez descargados. Habrá dos tipos de botones, un tipo de botón principal con fondo verde y letras blancas y un tipo de botón secundario con letras verdes y fondo blanco. Los botones son los siguientes:

Figura 4.11: Ejemplo de botón principal

Figura 4.12: Ejemplo de botón secundario

Las clases CSS de los botones son las siguientes:

Clase del botón principal:

```
.boton_principal_sn {
  cursor: pointer !important;
  margin-top: 30px !important;
  margin-bottom: 30px !important;
  position: absolute !important;
  margin-left: 30px !important;
  border-radius: 5px !important;
  border: 2px solid #005a6a
    !important;
  background-color: #005a6a
    !important;
  color: white !important;
  justify-content: center
    !important;
  padding: 8px 12px 8px 12px
    !important;
  align-items: center !important;
  display: inline-flex !important;
  text-align: center !important;
  font-size: 13px !important;
  font-weight: bold !important;
}
```

Clase del botón secundario:

```
.boton_secundario_sn {
  cursor: pointer !important;
  position: absolute !important;
  margin-left: 30px !important;
  margin-bottom: 50px !important;
  margin-top: 25px !important;
  border-radius: 5px !important;
  border: 2px solid #005a6a
    !important;
  background-color: white
    !important;
  color: #005a6a !important;
  justify-content: center
    !important;
  padding: 8px 12px 8px 12px
    !important;
  align-items: center !important;
  display: inline-flex !important;
  text-align: center !important;
  font-size: 13px !important;
  font-weight: bold !important;
}
```

4.2.4. Paleta de colores

Ha habido que decidir una paleta de colores para mostrar los resultados. La decisión tomada ha sido usar colores suaves y que no sean agresivos para el usuario:

TIPO DE RESULTADO	NOMBRE EN INGLÉS	CÓDIGO HTML	COLOR
Se cumple	Passed	#C8FA8C	
Fallo	Failure	#FA8C8C	
Advertencia	Can't tell	#F5FA8C	
No presente	Not Present	#FFFFFF	
No comprobado	Not Checked	#8CFAFA	

Tabla 4.1: Paleta de colores de los resultados

Estos son los colores que se usarán tanto para la tabla de resultados como para los resultados del estándar. En la siguiente página podemos ver dos ejemplos, uno de cada uno.

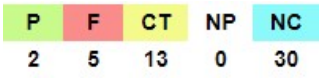

Figura 4.13: Ejemplo de tabla de resultados

Figura 4.14: Ejemplo de estándar cumplido

4.2.5. Resultado final

En las figuras a continuación, podemos observar el resultado final de la extensión. Una para cuándo la web no tiene información y otra para una web con información.

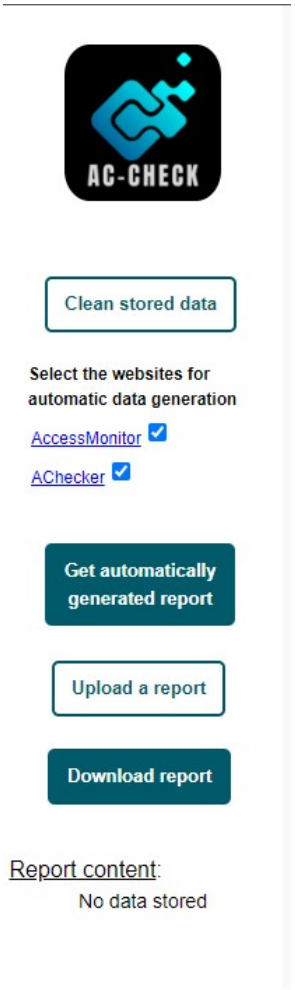

Figura 4.15: Extensión sin datos almacenados

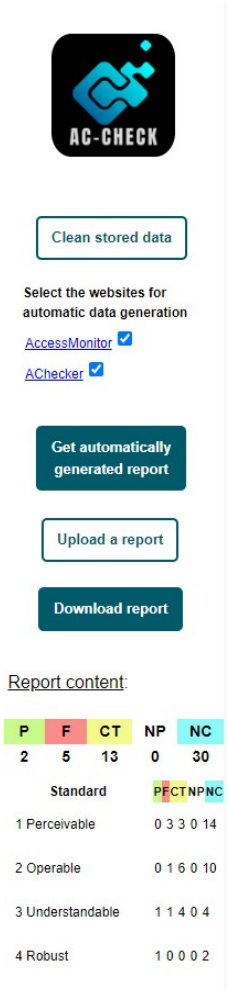

Figura 4.16: Extensión con datos almacenados

Si hay información se mostrarán dos tipos de resultados. El primero será una tabla con un recuento de los criterios que se cumplen, que fallan, advertencias, criterios no presentes y criterios no comprobados. Podemos ver un ejemplo en la figura 4.13.

Además, se mostrarán los resultados concretos del informe mediante una **tabla desplega-**

ble. La tabla desplegable agrupará los criterios (o sub-sub-estándares) y los resultados de los mismos por estándares y sub-estándares. Por cada agrupación, se mostrará los resultados en términos de criterios cumplidos, fallados, advertencias... que tiene en su interior. Como se observa en la figura 4.18, la suma de los resultados de los sub-estándares da el resultado para el estándar 2.

Para acceder a los resultados, se irá haciendo click en el "padre" a través del árbol. Véase, si queremos acceder al criterio 2.1.1 "Keyboard", deberemos hacer click primero en el estándar 2 "Operable"(4.17), después en el sub-estándar 2.1 "Keyboard accessible" (4.18) y por último en el criterio 2.1.1 "Keyboard" (4.19).

Figura 4.17: Vista de estándares

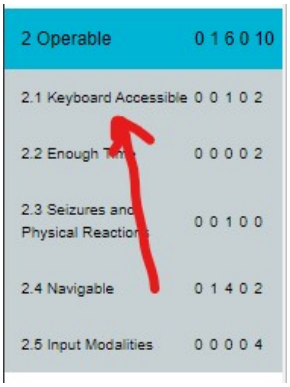

Figura 4.18: Vista de sub-estándares

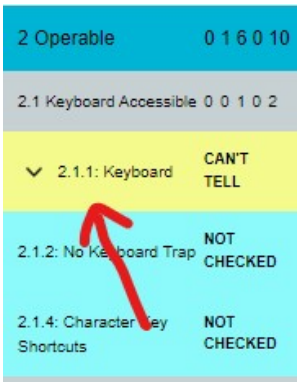

Figura 4.19: Vista de criterios del sub-estándar selecionado

Una vez hagamos el click en un criterio, cuyo resultado sabremos dependiendo del color de fondo además de que está especificado a la derecha de la fila, si tiene el icono de una flecha en el lado izquierdo (por ejemplo, el 2.1.1 de la figura 4.19) se podrá hacer click sobre el criterio y se mostrá información para ese criterio. Eso incluye un resultado (aceptado, fallado...), el analizador que ha obtenido ese resultado, un mensaje y un código.

Vamos a coger como ejemplo el criterio 1.1.1. Lo primero que podemos observar en la figura 4.20 es que el icono de la flecha ha cambiado de sentido y ahora mira para arriba en vez de para abajo. Aunque esté cortada la captura para que no fuera demasiado larga, aparecen todos los resultados para ese criterio, pero nos quedaremos con el primero. Vemos que el analizador es *AccessMonitor*, el resultado es de fallo y el mensaje en inglés indica que puede haber un fallo con el texto alternativo de la imagen. El analizador está en lo cierto, porque la imagen no tiene texto alternativo, simplemente tiene un espacio en blanco en el atributo "alt".

Si clicamos en el código, que tiene un estilo especial de color de texto rojo fuerte y fondo rojo claro, la extensión marcará el elemento que comete el error y le añadirá un borde rojo. Además, moverá la vista del usuario hacia el elemento recién marcado. Podemos observarlo en la figura 4.21.

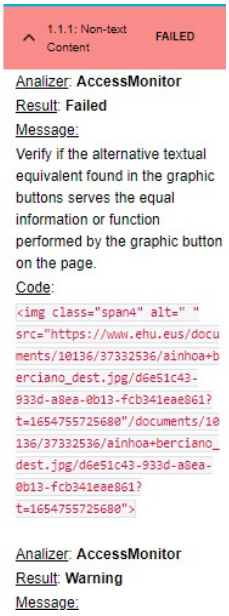

Figura 4.20: Vista de resultados del criterio 1.1.1

Figura 4.21: La imagen está marcada

Si después de clicar este elemento clicamos otro, se pintará el otro elemento y el elemento que se ve en la figura 4.21 volverá a cómo estaba anteriormente.

5. CAPÍTULO

Implementación

Este capítulo se centrará en la implementación del proyecto. Esto incluye la preparación del entorno de desarrollo, la estructura de los ficheros, las funciones usadas a destacar, la documentación del proyecto y los posibles aspectos de mejora en un futuro.

5.1. Preparación del entorno

5.1.1. Herramienta del desarrollo

Antes de comenzar a desarrollar el código para el proyecto, hubo que decidir la herramienta que se usará para llevar a cabo tal hecho. La herramienta que yo he seleccionado ha sido *Sublime Text 3*. *Sublime* [sub, 2008] es una herramienta de edición muy popular entre los desarrolladores. Pese a ser una herramienta de pago, tiene una versión gratuita que permite trabajar con absoluta normalidad y con todas las funcionalidades. La única diferencia es que en la versión gratuita, cada mucho tiempo, al hacer un guardado del documento salta un mensaje recomendándote la compra de la herramienta.

Otra ventaja de *Sublime* es que puedes instalar complementos o "add-ons" que facilitan mucho el desarrollo de código. Uno de los complementos que yo instalé fue *JSFormat* [Clark, 2022]. Este complemento, completamente gratuito y de open-source, ayuda al formateo de archivos JSON, algo que me fue realimente útil porque yo usé archivos JSON de manera muy habitual, al ser el tipo de fichero de los informes. Con el complemento, bastaba con hacer "CTRL"+"ALT"+"F" para que el fichero JSON se indentara[1] correctamente, facilitando mi labor de leer el documento.

Además, sublime, permite ver los ficheros dentro de la carpeta, como se puede observar en la figura 5.1.

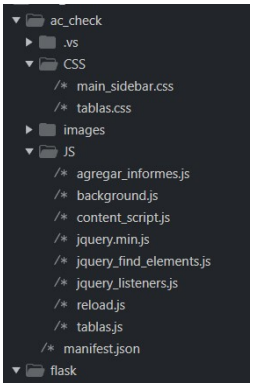

Figura 5.1: Vista de ficheros

Figura 5.2: Logo de Sublime Text 3

[1] Indentación: Es un anglicismo, de la palabra inglesa *indentation* de uso común en informática. También conocido como *sangrado*, este término significa mover un bloque de texto hacia la derecha insertando espacios para así separarlo del margen izquierdo y distinguirlo mejor del texto.

5.1.2. Creación del entorno virtual y puesta a punto de *Flask*

Para poder usar la aplicación de servidor necesitamos instalar *Flask*. Para ello, no basta simplemente con instalarlo, es necesaria la creación de un entorno virtual [fla, 2010b]. Una vez creado el entorno virtual (ver proceso en la sección A.4, Puesta a punto de la aplicación de servidor), se procede a instalar las librerias de *Python* que utilizaremos. Los comandos se detallan a continuación.

```
$ pip install Flask
$ pip install flask_cors
$ pip install bs4
$ pip install datetime
$ pip install selenium
$ pip install chromedriver-autoinstaller
```

Una vez tenemos el entorno virtual operativo y las librerías instaladas podemos empezar a programar sobre el fichero *app.py*, que será el fichero en el que se encontrará la totalidad de las funciones de la aplicación de servidor. Lo podemos ver en el apartado siguiente.

5.2. Estructura de los ficheros del proyecto

Los ficheros del proyecto se dividirán en dos carpetas: una carpeta para los ficheros de la extensión y otra para los ficheros de la aplicación de servidor. Las carpetas se llamarán *ac-check* y *flask* respectivamente.

5.2.1. Carpeta Flask

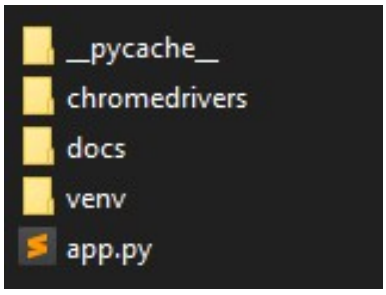

Figura 5.3: Directorio de flask

- __pycache__: Carpeta en la que se guarda la caché de *Flask*.
- chromedrivers: Carpeta en la que se instalará/actualizará automáticamente el *Chromedriver* adecuado para el equipo (ver 5.3.10).
- docs: Carpeta en la que se almacena la documentación creada con *Sphinx* para las funciones del fichero *app.py*.

- venv: Carpeta del entorno virtual creado. Será necesaria borrarla y volver a crearla tal y como se detalla en la sección A.4, Puesta a punto de la aplicación de servidor.
- app.py: Fichero en el que se encuentran todas las funciones de la aplicación de servidor.

5.2.2. Carpeta ac-check

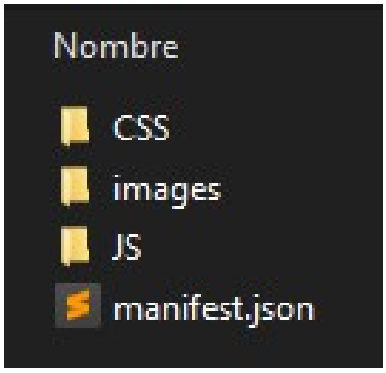

Figura 5.4: Directorio de ac-check

- CSS: Carpeta en la que se guardarán los ficheros relativos al estilo de la extensión de Chrome. Ficheros con extensión .css.
- images: Carpeta en la que se guardan las imágenes que usará la extensión. Eso incluye tanto el logo como los iconos. También será la carpeta desde la que la aplicación de servidor cogerá los iconos de las flechas.
- JS: Carpeta en la que se guardará el grueso del código de la extensión de Chrome. Fichero con extensión .js.
- manifest.json: El fichero *manifest.json* es el fichero necesario para que Chrome reconozca esa carpeta como una extensión. En ese fichero se definen aspectos importantes como el icono, la ejecucón de scripts o los permisos que necesitará la aplicación.

5.2.3. Carpeta ac-check/CSS

La carpeta contiene dos ficheros.

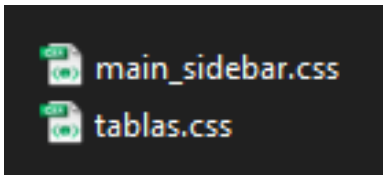

Figura 5.5: Directorio de CSS

- main_sidebar.css: Será el fichero donde se guardarán los estilos generales a la barra lateral de la extensión, como el estilo de la propia barra, el del logo o el de los botones.
- tablas.css: Fichero en el que están definidos los estilos relativos a la tablas de resultados del informe.

5.2.4. Carpeta ac-check/JS

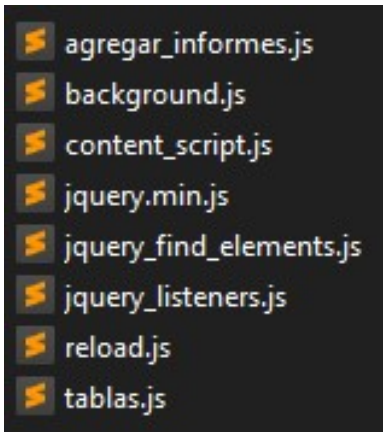

Figura 5.6: Directorio de JS

- agregar_informes.js: Fichero en el que se encuentran los ficheros de agregación de informes como *merge(json,json1)*. Además, se encuentra la función *main()* que actualiza las soluciones de informe guardadas en memoria.
- background.js: Es el fichero más importante de la extensión porque es el fichero en el que se realizan las acciones de fondo de la extensión de Chrome. Tiene solo una función, *main_bk()*, que es la encargada de añadir el listener para el clicado en el icono de la extensión desde la barra de extensiones de Chrome, permitiendo encender y apagar la extensión.
- content_script.js: Es el script encargado en crear la estructura de la extensión y añadirlo al árbol DOM de web en la que se encuentra el usuario mediante la función *init()*.
- jquery.min.js: El fichero fuente de *JQuery*. Versión 3.5.1.
- jquery_find_elements.js: Es el fichero encargado de, dado un código de un elemento, buscarlo en el árbol DOM. Las funciones se ejecutarán cuando el usuario haga clic sobre un código de los resultados del informe. Escrito usando *JQuery*.
- jquery_listeners.js: El fichero que añade los listeners para los botones, enlaces, tablas y elementos de la extensión. Escrito usando *JQuery*.
- reload.js: Contiene la instrucción de recarga de la página. Se usa cuando la extensión se apaga o se enciende para hacer desaparecer o mostrar la barra lateral respectivamente.
- tablas.js: Es es el fichero que se encarga del análisis de los informes JSON que recibe la extensión. Tras analizarlos crea la tabla de resultados y la guarda en memoria. A destacar, la función *update()*, que realiza el análisis y crea las tablas de resultados y la función *print_report_result(keyST)*, que imprime, si lo tuviera, el código de los resultados del informe.

5.2.5. Carpeta ac-check/images

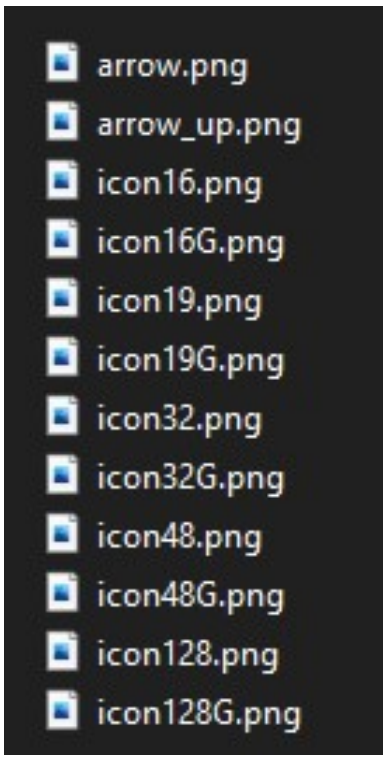

Figura 5.7: Directorio de images

- arrow.png: Icono de flecha mirando para abajo. Usado en criterios cuando están sin desplegar.
- arrow_up.png: Icono de flecha mirando para arriba. Usado en criterios cuando están desplegados.
- icon16.png: Icono de la extensión a tamaño 16x16. Usado en la barra de extensiones cuando la extensión está encendida.
- icon16G.png: Icono de la extensión en gris a tamaño 16x16. Usado en la barra de extensiones cuando la extensión está apagada.
- icon19.png: Icono de la extensión a tamaño 19x19. Usado por Chrome dependiendo del tamaño de pantalla.
- icon19G.png: Icono de la extensión en gris a tamaño 19x19.
- icon32.png: Icono de la extensión a tamaño 32x32. Usado por Chrome dependiendo del tamaño de pantalla.
- icon32G.png: Icono de la extensión en gris a tamaño 32x32.
- icon48.png: Icono de la extensión a tamaño 48x48. Usado por Chrome dependiendo del tamaño de pantalla.
- icon48G.png: Icono de la extensión en gris a tamaño 48x48.
- icon128.png: Icono de la extensión a tamaño 128x128. Usado cuando se muestra el logo de AC-Check en la extensión.
- icon128G.png: Icono de la extensión en gris a tamaño 128x128. No se usa pero está preparado por si fuera necesario usarlo.

5.3. Funciones a destacar

Para la realización de este proyecto se han usado muchas funciones de las herramientas detalladas en el apartado 2.7. Sería imposible ponerlas todas, pero sí que se van a destacar algunas funciones interesantes.

5.3.1. Funciones destacadas de *Flask*

Cómo he mencionado anteriormente, *Flask* es un framework muy sencillo. Lo único que hay que hacer para que se ejecute una función cuando entra a una determinada URL es escribir la siguiente instrucción antes de la función:

```
@app.route('/<<url>>)'
```

Siendo `<<url>>` la URL que desees que haga ejecutar la función. El resultado que muestre en pantalla la aplicación web será lo que devuelva la función mediante `return`.

5.3.2. Funciones destacadas de *Flask_cors*

Tal y cómo vimos en la figura 2.3, por defecto la aplicación de servidor no se puede conectar con la extensión por políticas CORS. Para arreglar ese error, hay que especificar que funciones quieres que usen CORS añadiendo la siguiente instrucción en la cabecera de la función.

```
@cross_origin()
```

5.3.3. Funciones destacadas de *Concurrent.futures*

Recordemos que *Concurrent.futures* es la librería que permite ejecutar funciones en paralelo. Para expresar al programa que funciones de la aplicación de servidor queremos que se realicen en paralelo usamos las siguientes funciones:

- Para crear la "pool"de trabajo.
  `with concurrent.futures.ThreadPoolExecutor(max_workers=2) as ex:`

- Para ejecutar la función Y que deseamos con el parámetro P.

```
f1 = ex.submit(Y, P))
```

- Para obtener el resultado de la función que se había mandado llamar.

```
informe = f1.result()
```

Una vez se obtengan ambos resultados, el programa seguirá su curso.

5.3.4. Funciones destacadas de *BeautifulSoup*

Las funciones usadas para moverse fácilmente a través de los elementos han sido las siguientes:

- Para crear el objeto soup, siendo "html" el código HTML de la página en la que se encuentra el scraper.

```
s = soup(html, features="html.parser")
```

- Para buscar elementos en el árbol HTML usamos `find`, que busca la etiqueta indicada que cumpla los parámetros que se le indican (en este caso, se hace una búsqueda de un elemento tipo tabla con una determinada clase).

```
tabla = s.find('table', {'class': 'evaluation-table'})
```

- Una vez se ha obtenido un elemento, como en el ejemplo anterior, se puede seguir buscando sobre dicho elemento, obteniendo resultados de los elementos con las etiquetas especificadas que se encuentren dentro de ese elemento.

```
cabeza = tabla.find('tbody')
```

- Si queremos buscar todos los elementos, lo realizamos usando `find_all`.

```
rows = cabeza.find_all('tr')
```

- Para obtener los atributos de un determinado elemento, por ejemplo de un hipervínculo (<a>), se puede usar `get`. A su vez, podemos observar que se puede acceder a los elementos internos poniendo un punto.

```
link = cols[3].a.get('href')
```

5.3.5. Funciones destacadas de *Selenium Webdriver*

Las funciones que se destacan para navegar a través de páginas web a la hora de hacer el scraping son las siguientes.

- Una vez tenemos la URL que queremos visitar y el navegador definido [2], podemos hacer *get* para obtener la página web deseada.

```
browser.get(url)
```

- Para localizar un elemento de la web, usamos *find_element* y pasamos por parámetro el atributo de identificación por el que queremos encontrarlo.

```
campo_de_texto = browser.find_element(By.ID, "checkuri")
```

- Si queremos hacer click en un botón, se usará la función *click*.

```
boton_submit.click()
```

- Si queremos escribir algo dentro de un campo de texto en el navegador podemos usar *send_keys* y como parámetro la cadena de caracteres a escribir. [3]

```
campo_de_texto.send_keys(url)
```

[2] El navegador se define usando ChromeDriver, en la subsección 5.3.6
[3] Para usar send_keys será necesario importar Keys desde selenium.webdriver.common.keys

- Para obtener el código HTML de la página actual se usa *page_source*

```
html = browser.page_source
```

- Por último, para cerrar el navegador usamos *quit*

```
browser.quit()
```

5.3.6. Funciones destacadas de *Chromedriver*

Tendremos que configurar el driver para que *Selenium* pueda usarlo.

- Primero crearemos el objeto *options* al que le añadiremos las opciones que queremos que tenga nuestro navegador. Para ello usaremos el constructor *ChromeOptions*

```
options = webdriver.ChromeOptions()
```

- Si queremos añadir algún parámetro de configuración, usaremos *add_argument*. Esto es un aspecto importante ya que por ejemplo para usar *AccessMonitor* es necesario que el navegador esté en inglés para que el contenido de la página se muestre en inglés. De no ser así, al ser una página del gobierno de Portugal, el contenido de la página estaría en portugués.

```
options.add_argument("--lang=en");
```

Por último, crearemos el objeto del navegador usando *Chrome* y pasando como parámetro las opciones que hemos defenido anteriormente y la dirección relativa en la que se encuentra el driver.

```
browser = webdriver.Chrome(options=options, executable_path='./ch
   romedriver.exe')
```

5.3.7. Funciones destacadas de *WebDriverWait*

El funcionamiento de la clase es sencillo. Una vez el browser ha pedido una URL, le especificamos que espere un determinado numero de segundos a que aparezca el elemento con la clase mencionada. Si no lo hace en ese numero determinado de segundos, salta la excepción.

```
try:
    browser.get(url)
    timeout_in_seconds = 50
    WebDriverWait(browser, timeout_in_seconds).until(ec.presence
       _of_element_located((By.CLASS_NAME, 'evaluation-table')))
    ...
except TimeoutException:
    print("I give up...")
```

5.3.8. Funciones destacadas de *JavaScript*

Al estar toda la extensión programada en *JavaScript*, se han usado decenas de funciones distintas, pero hay algunas para destacar, en el aspecto del marcado de elementos. Las funciones se detallan a continuación.

- Si queremos buscar un elemento usando *XPATH*, se ha usado la función _x, que es la función que usa la cosola de Chrome para realizarlo (con nombre de función $x).

```
function _x(STR_XPATH) {
    var xresult = document.evaluate(STR_XPATH, document,
        null, XPathResult.ANY_TYPE, null);
    var xnodes = [];
    var xres;
    while (xres = xresult.iterateNext()) {
        xnodes.push(xres);
    }
    return xnodes;
}
```

- Para buscar un elemento por su posición usamos *document.querySelector(X)*, siendo X un string con la posición del elemento en el árbol DOM.

```
let elemento = document.querySelector(ultimo);
```

- Para mover la vista del navegador al elemento que desee, podemos usar *scrollIntoView()*.

```
ele.scrollIntoView();
```

5.3.9. Funciones destacadas de las extensiones de Chrome

Ha sido necesario usar funciones específicas de las extensiones de Chrome para garantizar el correcto funcionamiento de la misma.

- Para que se ejecute una función cada vez que se instale una extensión podemos usar `chrome.runtime.onInstalled`.

```
chrome.runtime.onInstalled.addListener(function() {
    ...
});
```

- Para que se ejecute una función cada vez que se haga click en el logo de la extensión en barra superior del navegador simplemente usaremos `chrome.action.onClicked`. En mi caso, lo uso para encender/apagar la extensión.

```
chrome.action.onClicked.addListener((tab) => {
    ...
});
```

- Para cambiar el icono de la extensión podemos usar `chrome.action.setIcon`. En mi caso, lo uso para cambiar el logo en función de si la extensión está apagada o encendida para que el usuario pueda saber el estado de la extensión.

```
chrome.action.setIcon({path: "/images/icon16.png"});
```

- Para que la extensión ejecute un script cada vez que sea clicada podemos usar `chrome.scripting.executeScript`. En mi caso se utiliza para recargar la página al encender/apagar la extensión.

```
chrome.scripting.executeScript({
    target: {tabId: tab.id},
    files: ['/JS/content.js']
});
```

- Podemos usar algo que se podría definir como una variable global de navegador que todas las pestañas podrían obtener. Esto es algo muy interesante porque el comportamiento de la extensión debe ser el mismo en todas las pestañas/ventanas. No se concede que la extensión esté encendida en una ventana y apagada en otra ventana del mismo navegador. Es para lo que uso `chrome.storage.sync.set`.

```
chrome.storage.sync.set({'toggle':false});
```

- Al igual que en el anterior punto guardamos una variable "global", también la podemos obtener usando `chrome.storage.sync.get`. El valor guardado se devuelve en el parámetro result del callback.

```
chrome.storage.sync.get(['toggle'], function(result) {
    ...
});
```

5.3.10. Funciones destacadas: Autoinstalación de *Chromedriver*

Como ya se ha mencionado a lo largo de la memoria, la aplicación de servidor necesitará tener un driver de Chrome instalado para poder hacer el scraping. Ese driver tiene que ser de la misma versión que el navegador Chrome instalado en el equipo para poder funcionar de manera correcta.

Debido a las frecuentes actualizaciones de Chrome sumado a que la aplicación de servidor está diseñada para usarla en distintos sistemas operativos, se decidió que lo más cómodo sería realizar una auto-instalación del driver y que éste se actualizase de forma automática, de modo que el usuario no necesitara actualizar el driver de forma manual.

Ese proceso se podido realizar gracias a la librería *chromedriver_autoinstaller* [Jo, 2021] creada por *Yeongbig Jo*, miembro del departamento de *Computer Science & Engineering* de la *Seoul National University*.

El proceso de auto-instalación es simple. Se usará una estructura `try/catch`. En el `try`, primero se comprobará si hay algún driver instalado en la carpeta *chromedrivers*, si no lo hay, se lanza una excepción. Si lo hay, intenta cogerlo y crear un *browser*. En caso de conseguirlo, significará que a versión del driver es la correcta. Si no lo consigue, saltará una excepción.

Si el programa llega a la parte del `catch`, significará que hace falta actualizar el driver. Es entonces cuando se ejecutará la función para la autoinstalación, en la que especificamos que se realice en el directorio *chromedrivers*.

```
chromedriver_autoinstaller.install(path='./chromedrivers')
```

Una vez se haya instalado la nueva versión del driver, la aplicación de servidor elimina la versión anterior del driver guardada en la carpeta *chromedrivers*, ahorrando así espacio en memoria del equipo del usuario.

5.4. Documentación

El código está completamente comentado. Todas las funciones tienen una cabecera que explica qué hace la función. Adicionalmente, usando la herramienta *Sphinx*, se ha realizado una documentación en HTML del código de la aplicación de servidor escrita en Python. La herramienta coge las cabeceras de las funciones y crea un documento HTML con esas cabeceras. Además incluye un índice y hipervinculos entre el índice y las funciones y entre la definición de la función y el código de la misma.

Para acceder a la documentación automática, hay que abrir el fichero *index.html* en el directorio *flask/docs/_build/html* de la raíz del proyecto.

Figura 5.8: Índice de la documentación HTML

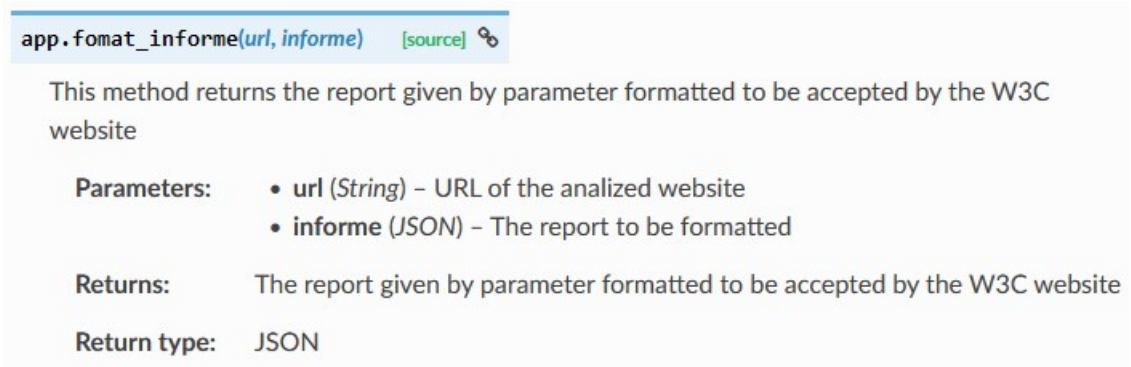

Figura 5.9: Ejemplo de documentación de una función

5.5. Extensibilidad y mejoras posibles

El proyecto puede ser mejorado en un futuro, si así se deseara. Las mejoras posibles podrían ser dos: alojar la aplicación de servidor en un servidor web como podría ser AWS[4] y la publicación de la extensión la tienda de Chrome[5] y mejorar la extensión añadiendo más analizadores automáticos de accesibilidad.

5.5.1. Publicación del proyecto

La publicación de proyecto no se ha realizado ya que, a parte de suponer un incremento en la planificación horaria que ya de por sí sobrepasa las 300 horas requeridas, supone un coste económico que no estoy dispuesto a asumir. Si alguien quisiera asumirlo, trataré de hacer hacer una aproximación del coste.

La publicación de la extensión en la tienda de Google sería la parte más económica de la operación, ya que únicamente hay que realizar **un pago simbólico de 5USD que solo se realiza una vez**. Una vez te hayas registrado como desarrollador y pagado la tasa, podrás mandar la extensión y será verificada por el equipo de Google. En un plazo de al rededor de dos días sería, presumiblemente, aprobada, ya que no incumple ninguna regla de Google.

El problema estaría en el alojamiento de la aplicación de servidor en la nube ya que el proceso de scraping de la aplicación consume recursos. Además, habría que realizar un escalado para saber cuántos usuarios usarían la aplicación y contratar un plan en base a eso, algo difícil de calcular sin que ni siquiera se haya publicado la extensión. Lo recomendable sería, al menos al principio del ciclo de vida, que la app fuera publicada en un servidor privado, como un ordenador o una *Raspberry Pi*[6], y monitorear el uso de recursos para en un futuro alojarla en alguna web de almacenamiento.

5.5.2. Extensión del proyecto

El proyecto podría extenderse y mejorarse añadiendo nuevos analizadores automáticos. Modificar el proyecto para añadir nuevos analizadores no sería difícil y se explicará el

[4] AWS: Amazon Web Services. Servicio de alojamiento en la nube

[5] Tienda online en la que se pueden obtener extensiones de Chrome https://chrome.google.com/webstore

[6] https://es.wikipedia.org/wiki/Raspberry_Pi

proceso a continuación, pero si que la ampliación de la extensión es complicada.

Es complicada por la falta de analizadores automáticos de calidad y, sobre todo, **gratuitos**. La mayoría de los analizadores de la web de herramientas de accesibilidad de *W3C* [her, 2022] que se consultaron eran servicios de pago. Si se quisiera ser ambicioso en un futuro, se podría contactar con los desarrolladores de algunos de esos analizadores y llegar a un acuerdo pero sin duda no sería barato.

Otras webs gratuitas tenían deshabilitado el scripting sobre sus páginas y no permitían que se obtuvieran los datos de forma automática. Uno de los ejemplos es la web de *Wave*[7].

Aún así. si se encontrará alguna web apropiada para añadirla a la extensión se podría hacer sin problema siguiendo los pasos explicados a continuación.

Cambios en la extensión

Los cambios en la extensión serían mínimos, únicamente habría que añadir el "checkbox" para que el usuario pueda seleccionar ese analizador. Eso se realizaría en el fichero *content_script.js*, cerca de la línea 35. También habría que modificar el evento del click sobre la obtención de informes automáticos para añadirle que tenga en cuenta el nuevo checkbox. Para ello, en la linea 119 del fichero *jquery_listeners.js* habría que añadir lo siguiente:

```
'YYY': $('#ZZZ').is(":checked")
```

Siendo YYY el nombre del analizador y ZZZ el id del checkbox.

Cambios en la app de servidor

La parte de la app del servidor sería la que más cambios sufriría. Primero, el desarrollador tendría que realizar una función que hiciera el scraping para el analizador añadido, al que llamaremos `analizar_web3`. Esa web deberá devolver un informe *JSON* con la siguiente estructura:

[7] https://wave.webaim.org/

```
{
    'Tester_Name': 'Codigo_nombre',
    'RESULTADO': 'Res',
    'Cases': {
        'Codigo_Estandar': {
        'Resultado': 'Codigo_res1',
        'Texto': 'Texto_mensajes_total',
        'Codigos': [{
            'web': 'Nombre_web',
            'tipo': 'Codigo_res2',
            'texto': 'texto_mensaje',
            'codigo': [{
                'texto_codigo': 'texto_de_codigo',
                'location': 'posicion_del_codigo'
            }(1)]
        },
        (2)]
    }(3)}
```

Siendo:

- Codigo_nombre: Código del nombre del analizador.
- Res: Si lo tuviera, resultado numérico de la web (e.g. 7.5/10).
- Codigo_Estandar: Código del estándar WCAG (e.g. WCAG21:non-text-content).
- Codigo_res1: Código del tipo de resultado. Opciones posibles: 'Failed','Passed' y 'Cannot Tell'
- Texto_mensajes_total: Texto agregado de todos los mensajes dentro del elemento "Codigos" del propio estándar. La concatenación de todos los *texto_mensaje*.
- Nombre_web: Nombre de la web. Con espacios, será como se mostrará en la extensión.
- Codigo_res2: Código del tipo de resultado para ese elemento. Opciones posibles: 'Failed','Passed','Warning' y 'Potential Problem'.

- texto_mensaje: Texto del mensaje que produce el analizador para ese elemento.
- texto_de_codigo: Texto del código del elemento en conflicto.
- posicion_del_codigo: Si lo hubiera, ubicación del elemento en el árbol DOM

Además, los números que aparecen en el JSON significan lo siguiente:

1. Es un array de los códigos. Podrá haber tantos como se deseen. Puede haber varios elementos que generen el mismo resultado de criterio.
2. También es un array. Dentro del código del estándar podrá haber tantos elementos como se deseen. Para un mismo criterio puede haber varios resultados en un análisis (un error, una advertencia...). Finalmente, el resultado de *Codigo_res1* será el *Codigo_res2* que más peso tenga[8].
3. Para el tercer número habrá un elemento por cada estándar con el que se haya obtenido resultado en la página web.

Una vez tenemos la función que devuelve el resultado, habrá que añadir esa función al pool de los analizadores pero solo si el usuario la ha seleccionado. Para ello, primero sacaremos la decisión del usuario y si es *true*, la añadiremos a la pool y esperaremos el resultado.

En el fichero *app.py*, en la línea 72 añadimos:

```
valor_checkboxY = received_json['YYY']
```

Siendo *YYY* el mismo valor que tiene YYY en la parte de "Cambios en la extensión".

[8]Orden de importancia: 'Failed', 'warning' o 'potential problem', 'passed'

Cambiamos la pool para que acepte tres trabajadores. cambiando la línea 77:

```
with concurrent.futures.ThreadPoolExecutor(max_workers=3) as ex:
```

Añadimos en las líneas 82 y 87 las siguientes instrucciones respectivamente:

```
if valor_checkboxY:
    f3 = ex.submit(analizar_web3, url)

if valor_checkboxY:
    informe_3 = f3.result()
```

Ya que tendremos el resultado del informe generado por el evaluador 3, ya solo nos quedará agregarlo al resultado de la agregación del primer y del segundo informe. Para ello, sustituimos la línea 88 por lo siguiente:

```
informe_final = merge_reports(informe_1, informe_2)
informe_final = merge_reports(informe_final, informe_3)
```

Ya tendríamos el tercer analizador implementando. Esto se podría hacer tantas veces como se quisiera, no tiene un número finito de analizadores que la aplicación del servidor pueda aceptar.

6. CAPÍTULO

Pruebas realizadas

En este capítulo se mostrarán algunas de las pruebas realizadas para comprobar el comportamiento correcto de la extensión sobre diversas páginas de distinto ámbito. Además, se hará una comparación entre el resultado de un solo analizador automático y el resultado agregado de los dos analizadores, un ejemplo de agregación manual de informes, una visualización del informe en la web de W3C así como un resumen de los errores detectados.

6.1. Preliminares

Antes de comenzar con las pruebas tendremos que preparar el entorno para realizarlas. Para ello, primero crearemos un nuevo perfil de navegador de Google Chrome, para que esté limpio de datos.

Tras crear el perfil, introducimos en la URL `chrome://extensions/`. Una vez ahí, activamos el modo de desarrollador y hacemos clic en "cargar descomprimida",

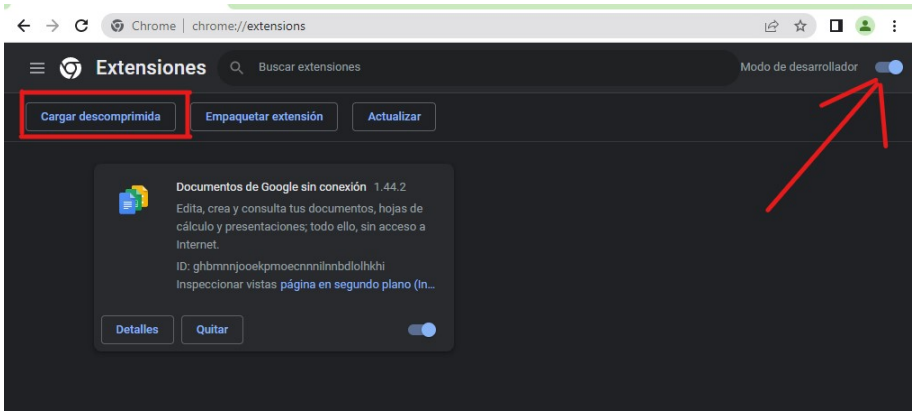

Figura 6.1: Proceso de carga de la extensión

Seleccionamos la carpeta de "ac-check" de la raíz del proyecto y a continuación veremos como la extensión ya aparece. Podremos observar el logo así como el nombre y la versión de la misma.

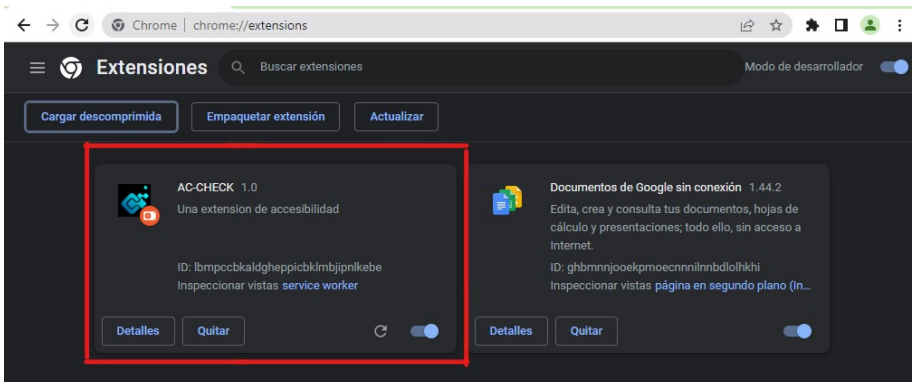

Figura 6.2: La extensión ya se muestra como instalada

6.2. Análisis automático y marcado de webs educativas

6.2.1. Página de la UPV/EHU

La primera página que probaremos será la página de la universidad a la que se presenta este TFG, la UPV/EHU, así que introducimos la URL de la página[1]. Podremos ver que ya aparece la extensión en el lado izquierdo de la página, además del icono de la extensión en el lado derecho de la barra de búsqueda. El icono tiene el color azul porque la extensión esta encendida.

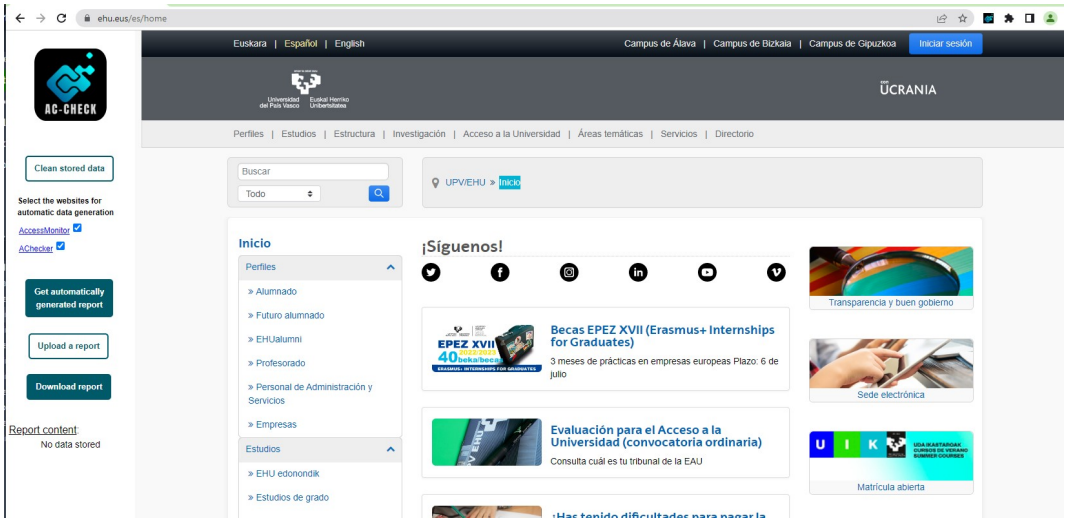

Figura 6.3: La extensión se muestra en la wed de EHU

Podemos observar que no hay datos almacenados ya que aparece la etiqueta de "No data stored". **Activamos la aplicación de servidor** (ver sección A.4, Puesta a punto de la aplicación de servidor) y hacemos click en "Get automatically generated report". Vemos que ahora en la extensión aparece un loader dinámico dando vueltas:

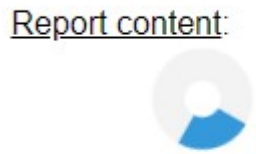

Figura 6.4: Aparece un loader en la extensión

[1] https://www.ehu.eus/es/home

Mientras, en la aplicación de servidor podemos observar que se muestra en la consola la URL a la que se le solicita hacer el análisis así como los valores de los checkbox de los dos analizadores. Como a la hora de hacer la solicitud del informe las dos estaban seleccionadas (es así por defecto), el valor de las dos es `true`.

```
127.0.0.1 - - [12/Jun/2022 18:53:37] "OPTIONS /getJSON/ HTTP/1.1" 200 -
Url: https://www.ehu.eus/es/home
AM: True
AC: True
```

Figura 6.5: La aplicación de servidor recibe las instrucciones para el análisis automático

Una vez la aplicación de servidor termina de hacer el análisis automático, envía el JSON resultante a la extensión. Cuando la extensión lo recibe, realiza un análisis de los datos y crea las tablas de resultados. Cuando la extensión finaliza, emite una alerta al usuario de que la información ya se ha recibido y recarga la página.

Como observamos en la figura 6.6, la extensión ya contiene información. En este caso, el resultado del informe es el siguiente: 2 criterios cumplidos, 5 criterios fallados, 13 criterios que podrían ser un error pero necesitan revisión manual, 0 criterios que se han detectado como no presentes y 30 criterios que no han sido testeados. Podemos ver que también nos salen resultados por cada estándar. Por ejemplo, viendo la tabla sabemos que los 5 criterios fallados se dividen de la siguiente manera: 3 en el estándar 1, 1 en el estándar 2 y otro más en el estándar 3.

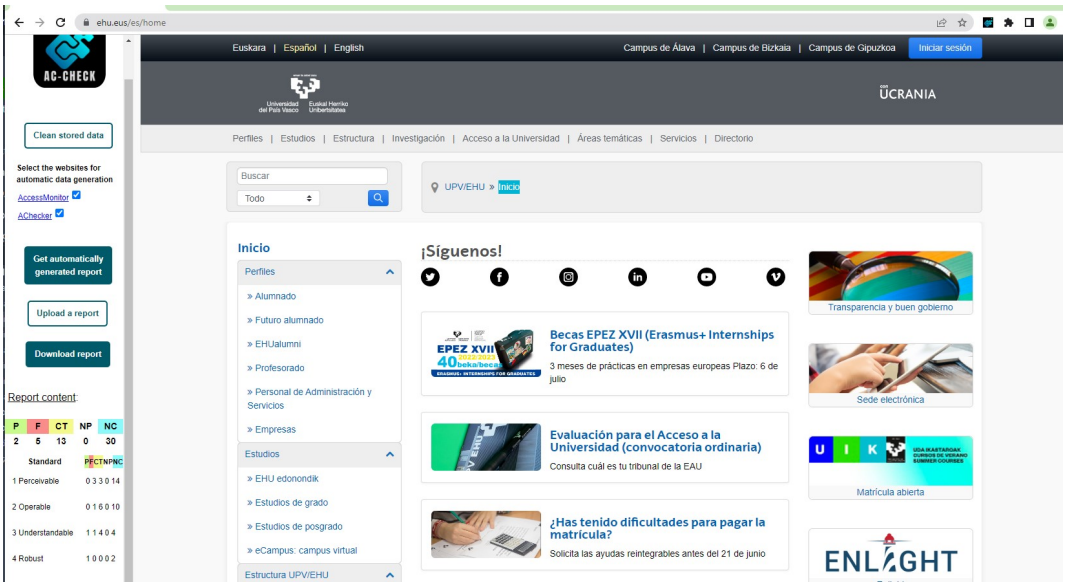

Figura 6.6: La extensión ya contiene información

Vamos ahora a hacer unas pruebas de marcado. Se verán como pareja de imágenes. A la izquierda, aparecerá el error y a la derecha el elemento marcado[2].

Error en una imagen en el criterio 1.1.1:

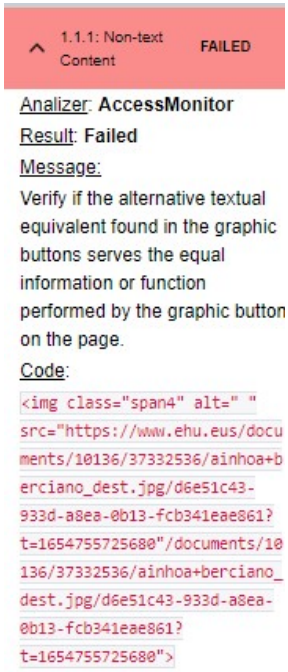

Figura 6.8: Elemento marcado

Figura 6.7: Mensaje y código del error

El de antes era de *AccessMonitor*, vamos a probar ahora con uno de *AChecker*. Un ejemplo podría ser un error de un input en el criterio 1.3.1. Vemos que este además tiene posible solución:

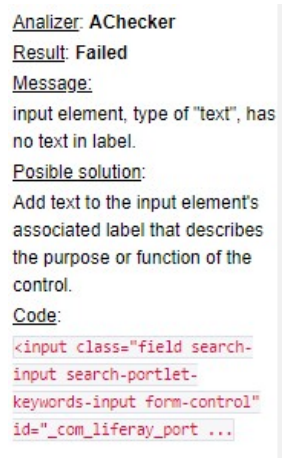

Figura 6.10: Elemento marcado

Figura 6.9: Mensaje y código del error

[2]Las capturas no irán editadas. El recuadro rojo lo produce la extensión

Los elementos anteriores eran de tipo de `img` y de tipo `input`. Vamos a probar otro elemento, en este caso de tipo `p`, del criterio 1.3.1 también. En este caso, es un "warning" a diferencia de los anteriores, que eran errores.

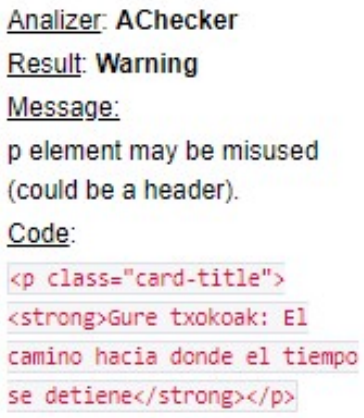

Figura 6.11: Mensaje y código del error

Figura 6.12: Elemento marcado

Ahora marcamos un elemento del tipo hipervínculo, con etiqueta `a`. Es un error del criterio 2.4.4:

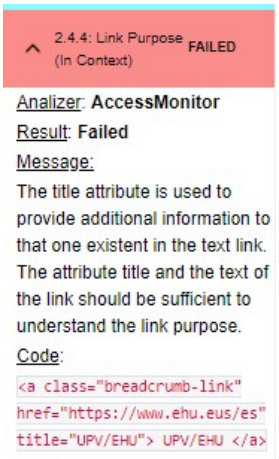

Figura 6.13: Mensaje y código del error

Figura 6.14: Elemento marcado

Por último, vamos a visualizar como se verían criterios satisfechos o criterios no comprobados. Se puede observar en la figura 6.15 que el criterio 4.1.2 tiene el logo de la flecha. Esto se debe a que tiene contenido (mensaje del analizador que da el criterio por satisfecho. nombre del analizador...) mientras que los no comprobados no tienen la flecha porque no tienen contenido.

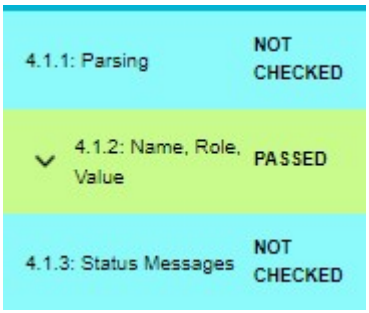

Figura 6.15: Muestra de criterios satisfechos y no comprobados

6.2.2. Página de la EOI de San Sebastián

La siguiente página a analizar ha sido la web de la EOI[3] de San Sebastián. Se ha elegido esta web porque además de no ser una web de educación muy común como la que podría ser la de una universidad o un IES[4], tiene un diseño anticuado lo que es propicio para la obtención de más errores de accesibilidad.

Hacemos click en generar el informe de manera automática y observamos los resultados. Esta web tiene 3 fallos más que la web de la UPV/EHU. Veremos algunos de ellos.

Figura 6.16: Resultados para la web de la EOI de San Sebastián

[3]EOI: Escuela Oficial de Idiomas
[4]IES: Instituto de Enseñanza Secundaria

Como en la web anterior, vuelve a haber fotos que no tienen un texto alternativo correcto como puede ser el siguiente ejemplo del criterio 1.1.1.

Figura 6.17: Mensaje y código del error

Figura 6.18: Elemento marcado

Otro elemento visto anteriormente es el siguiente, de tipo `input`. En este caso es un "Potential Problem" de Achecker.

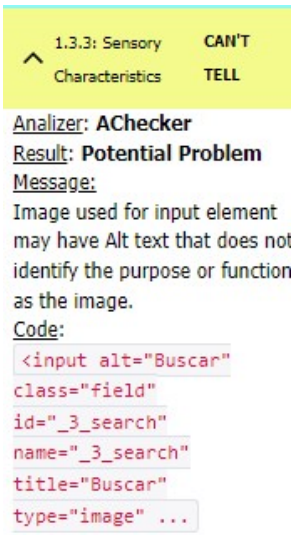

Figura 6.19: Mensaje y código del error

Figura 6.20: Elemento marcado

A continuación, vamos a marcar elementos que no hemos visto hasta ahora.

Elemento de negrita, con etiqueta `b` que incumple el criterio 3.3.2.

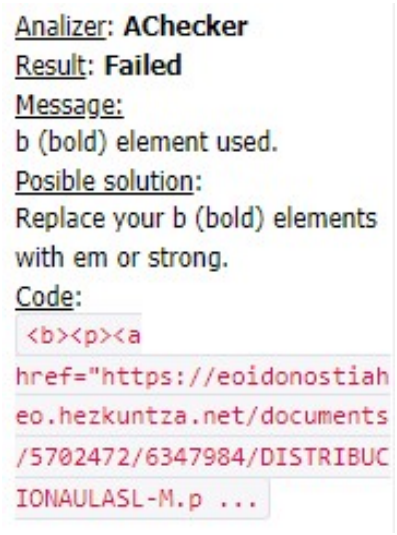

Figura 6.21: Mensaje y código del error

Figura 6.22: Elemento marcado

Vamos a probar ahora con otro elemento que tampoco habíamos visto, un elemento de tipo `body`. Una advertencia del criterio 2.1.1 generado por *AChecker*.

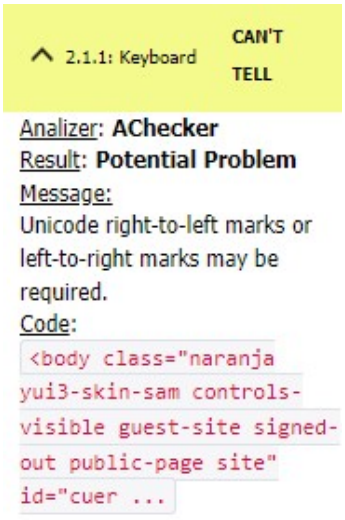

Figura 6.23: Mensaje y código del error

Figura 6.24: Elemento marcado

Otro elemento nuevo sería los que tienen la etiqueta `div`. El siguiente es una advertencia de *AccessMonitor* para el criterio 2.4.1. Además en la figura 6.25 podremos ver un ejemplo de que un resultado de criterio puede tener más de un código que lo genera. Nosotros solo clicaremos el segundo código.

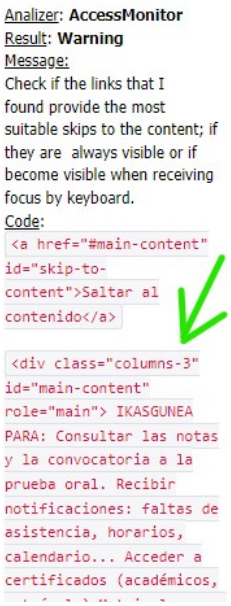

Figura 6.25: Mensaje y código del error

Figura 6.26: Elemento marcado

Por último, veremos un elemento de tipo `form`. Se trata de un error del estándar 3.2.2 detectado por *AccessMonitor*.

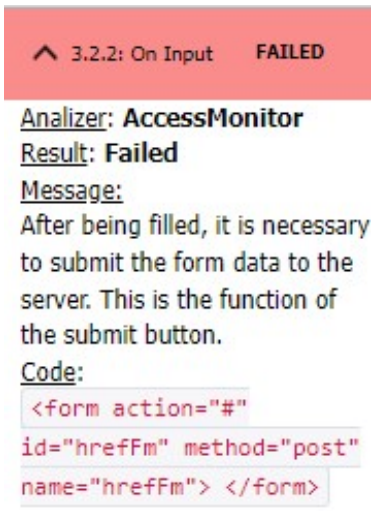

Figura 6.27: Mensaje y código del error

Figura 6.28: Elemento marcado

Analizadas ya webs de ámbito educativo, vamos a pasar al ámbito deportivo.

6.3. Análisis automático y marcado en webs deportivas

6.3.1. Página de la organización nacional de fútbol siete

La siguiente web en ser analizada es la web de la organización de fútbol siete[5]. Es una web muy frecuentada debido a la popularidad del fútbol siete. De hecho, han abierto ligas en 10 provincias, entre ellas en los 3 territorios históricos vascos, y solo en Madrid cuentan con 6 ligas diferentes, con sus respectivas divisiones.

Visitamos la web y solicitamos de nuevo el informe automático. De nuevo, con los dos analizadores seleccionados (posteriormente probaremos seleccionando solo uno de los dos analizadores). El resultado se observa a continuación.

Figura 6.29: La extensión muestra los resultados para la web de fútbol siete

Al igual que en las pruebas anteriores, intentaremos marcar algunos elementos que produzcan conflictos. Empezamos por una foto que provoca un error en el criterio 1.1.1. En este caso, clicamos en la encontrada por *AChecker*, pero también lo detecta *AccessMonitor*.

[5] https://www.futbolsiete.com/inicio/index.asp

Analizer: **AChecker**
Result: **Failed**
Message:
img element missing alt attribute.
Posible solution:
Add an alt attribute to your img element.
Code:

```
<img src =
"../_img/futbolsiete/ba
nner_fairplay3.gif"
width = "120px" height
= "66px" border = "0">
```

Figura 6.30: Mensaje y código del error

Figura 6.31: Elemento marcado

Probamos ahora con un `input` de tipo `password`, error en el criterio 1.3.1. El error indica que al campo le falta una etiqueta de tipo `label`.

Analizer: **AChecker**
Result: **Failed**
Message:
input element, type of "password", missing an associated label.
Posible solution:
Add a label element that surrounds the control's label. Set the for attribute on the label element to the same value as the id attribute of the control. And/or add a title attribute to the input element. And/or create a label element that contains the input element.
Code:

```
<input type ='password'
name = 'txtClave'
maxlength = '20' class
= 'inputLogin
identificacionClv' ...
```

Figura 6.32: Mensaje y código del error

Figura 6.33: Elemento marcado

Otro ejemplo podría ser un hipervínculo. A continuación, un error de etiqueta `a`, con foto incluida, encontrado por *AccessMonitor* que incumple el criterio 2.4.4.

Analizer: **AccessMonitor**
Result: **Failed**
Message:
This fail occurs always that a link is composed by an image and that image has an empty nature as alternative textual equivalent - I suspect that the Assistive Technologies' users don't know their destiny or purpose, or may even don't realize the existence of the link.
Code:

```
<a
href="http://www.Koolto
n.com" target="_blank">
</a>
```

Figura 6.34: Mensaje y código del error

Figura 6.35: Elemento marcado

También hay elementos que no habíamos visto hasta ahora. Por ejemplo, la siguiente tabla que provoca un error en el criterio 1.3.1, detectado por *AccessMonitor*:

Analizer: **AccessMonitor**
Result: **Failed**
Message:
I suspect that I might find data tables which is missing title identification `<caption>`.
Code:

```
<table
id="tablaNavegacion"
width="155px"
border="0"
cellpadding="0"
cellspacing="0"
style="border:0px
solid;">1 / 2442 / 2443
/ 2444 / 2445 / 2446 /
2447 / 2448 / 2449 /
24410 / 24411 / 24412 /
24413 / 24414 / 24415 /
24416 / 24417 / 24418 /
24419 / 24420 / 24421 /
24422 / 24423 / 24424 /
24425 / 24426 / 24427 /
```

Figura 6.36: Mensaje y código del error

Figura 6.37: Elemento marcado

Otro ejemplo no visto sería la etiqueta `select`, que ha sido detectada de nuevo en el criterio 1.3.1.

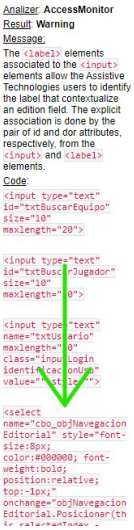

Figura 6.39: Elemento marcado

Figura 6.38: Mensaje y código del error

Hay más errores, pero son de la mayoría de los mismos tipos que ya hemos visto, con lo que procedemos con otra página de ámbito deportivo.

6.3.2. Página del CTA de Guipúzcoa

La página del CTA[6] de Guipúzcoa[7] no es una página muy popular pero es interesante comprobarla porque tiene una estructura distinta a las anteriores. Volvemos a obtener el informe automático.

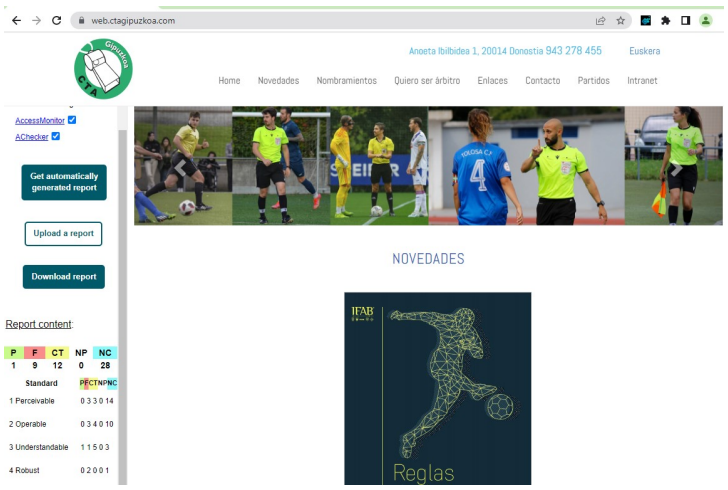

Figura 6.40: La extensión muestra los resultados para la web del CTA

[6] CTA: Comité técnico de árbitros
[7] https://web.ctagipuzkoa.com/

Lo primero que podemos ver es que el header aparece por encima de la extensión. Esto es un pseudoerror detectado que se detalla en la sección 6.8 Errores detectados.

Comenzamos con el marcado de un elemento de tipo `div` pero con un comportamiento curioso, ya que tiene un comportamiento de `listbox`. Error del criterio 1.3.1.

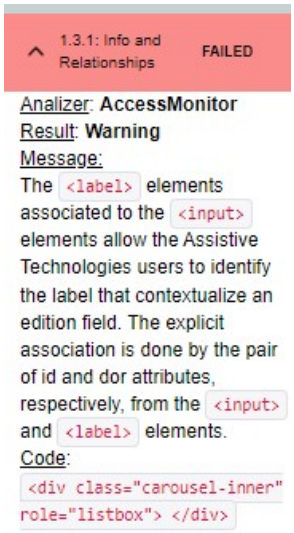

Figura 6.41: Mensaje y código del error

Figura 6.42: Elemento marcado

Otra etiqueta que no habíamos visto era la etiqueta `section`, error del criterio 2.4.1.

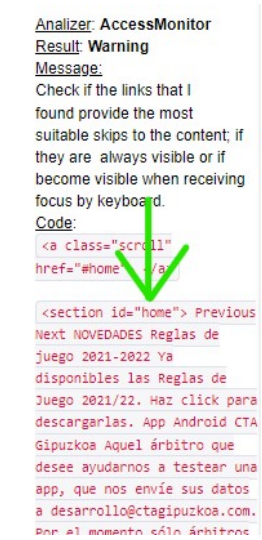

Figura 6.43: Mensaje y código del error

Figura 6.44: Elemento marcado

Por último, la etiqueta `H1` tampoco había sido marcada con anterioridad. Es un error del

criterio 2.4.6.

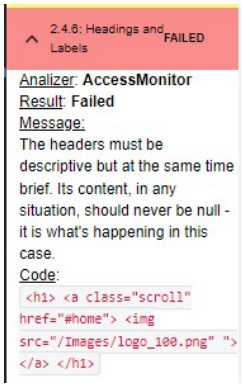

Figura 6.45: Mensaje y código del error

Figura 6.46: Elemento marcado

Visto ya una multitud de etiquetas distintas, simplemente vamos a realizar el análisis a otro tipo de paginas web y poner algunos ejemplos puntuales.

6.4. Análisis automático y marcado en webs de otros ámbitos

6.4.1. Webs de ocio: foro online de Reddit

El foro online de *Reddit*[8] es el foro online por excelencia, el más famoso de Internet. Vamos a realizar un análisis sobre él.

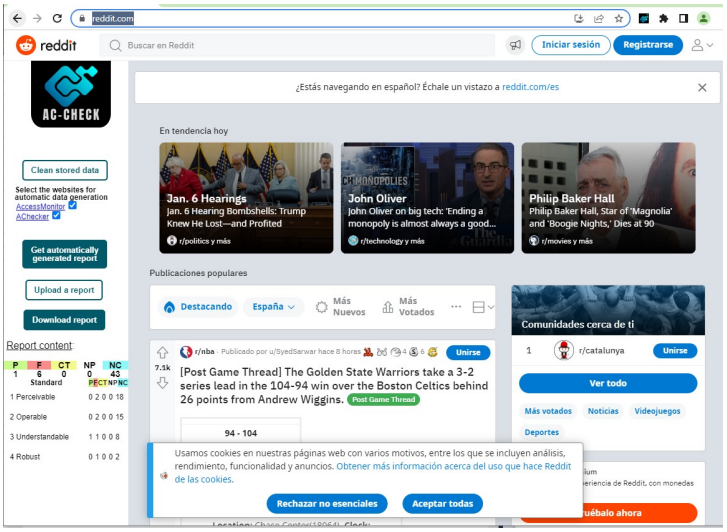

Figura 6.47: La extensión muestra los resultados para la web Reddit

[8] https://www.reddit.com/

Vemos que, de nuevo, el header se pone por encima de la extensión. Tal y cómo mencionamos antes y que se detalla en la sección 6.8 Errores detectados.

Para no saturar con ejemplos, vamos a poner solo dos, uno de etiqueta `a` y otro de la etiqueta `form`.

Etiqueta `a`: Error del criterio 2.4.4.

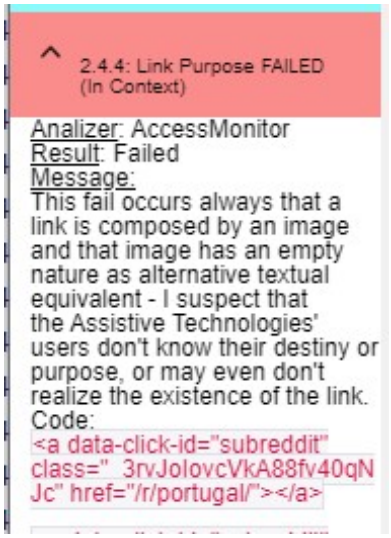

Figura 6.48: Mensaje y código del error

Figura 6.49: Elemento marcado

Etiqueta `form`: Error del criterio 3.2.2

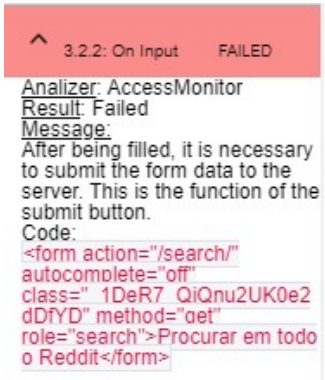

Figura 6.50: Mensaje y código del error

Figura 6.51: Elemento marcado

6.4.2. Webs de noticias: página de EITB

El último análisis que voy a realizar va a ser el de una página web de noticias. En concreto, la de EITB[9]. Es la web de noticias de la televisión pública vasca. Realizamos el análisis.

[9] https://www.eitb.eus/es/noticias/

Figura 6.52: La extensión muestra los resultados para la web de EITB

Vamos a ver un ejemplo que no habíamos visto hasta ahora. Se trata del **marcado múltiple de elementos**. Si varios elementos tienen exactamente el mismo código incumplirán el mismo criterio. En la foto podemos observar que varios iconos tienen el bordeado.

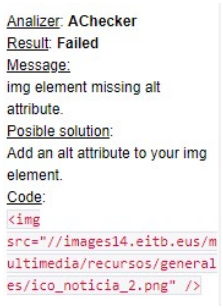

Figura 6.53: Mensaje y código del error

Figura 6.54: Varios elementos marcados

Para variar de tanto error, a continuación podemos observar un problema potencial del criterio 3.3.1. Un `input` de tipo text.

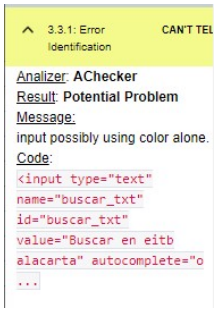

Figura 6.55: Mensaje y código del error

Figura 6.56: Varios elementos marcados

6.5. Analizador único vs agregación automática

Una de las características de esta extensión es que, mediante el uso de "checkbox", permite al usuario seleccionar solo los analizadores que desee. Esto se realiza porque puede ser que no le guste alguno de los analizadores automáticos y prefiera no usarlo.

Además de eso, permite elegir que informes quiere que sean agregados. En este caso, solo hay dos pero podría ser que en un futuro hubiera muchos más y el usuario podría elegir cuáles agregar y cuáles no. Esto se puede hacer gracias a que está implementada la agregación de informes no solo en el lado del cliente, sino que también en el lado del servidor.

Para verlo más claro, realizaremos pruebas para la web de la UPV/EHU, primero realizando el informe con *AccessMonitor*[10], después con *AChecker*[11] y por último seleccionando las dos opciones.

TIPO DE RESULTADO	SOLO AM	SOLO AC	AGREGADOS
Passed	5	0	2
Failure	4	3	6
Can't tell	2	14	14
Not Present	0	0	0
Not Checked	39	33	28

Tabla 6.1: Comparación de los resultados dependiendo del analizador

Los resultados de la agregación se pueden ver a simple vista. En un principio, con *AccessMonitor*, había 5 criterios que se cumplían. Después de la agregación, solo son 2. Eso se debe a que *AChecker* encuentra conflictos para 3 de los 5 criterios que aprueba *AccessMonitor*. Veamos un ejemplo.

El criterio 3.3.2, "Labels or instructions" aprueba en *AccessMonitor*, como podemos ver en la figura 6.57. El criterio aparece de color verde y su texto dice "Passed".

Sin embargo, *AChecker* detecta dos resultados para ese mismo criterio: un error y un problema pontencial. Como el error es más grave que un problema pontencial, *AChecker* reporta ese criterio como fallado. Por eso aparece en rojo y con el predicado "Failed" en la figura 6.58.

[10] Representado como "AM" en la tabla.
[11] Representado como "AC" en la tabla.

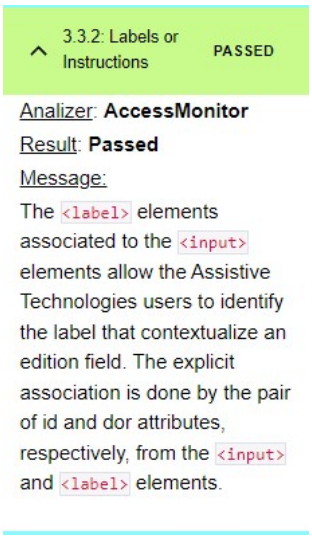

Figura 6.57: El criterio 3.3.2 es aprobado por AccessMonitor

Al hacer la agregación, se coge el resultado con más importancia[12]. Para *AccessMonitor* el criterio pasa, pero como para *AChecker* es suspenso, y el suspenso tiene más peso que el aprobado, el resultado de la agregación será el suspenso como se ve en la figura 6.59.

Algo que también se puede apreciar en la agregación es que los distintos resultados obtenidos se agregan. En la figura 6.59 se puede observar que aparecen los 3 resultados: El aprobado de *AccessMonitor* y el error y el problema potencial de *AChecker*.

Es por eso por lo que es importante la agregación. Porque criterios que se iban a dar por satisfechos en realidad no lo estaban. Algo que sería una tarea muy tediosa el realizarlo a mano y que realizarlo de manera automática facilita mucho el trabajo. Cuanto más analizadores automáticos haya, más fiable será el resultado.

Revisar muchos analizadores manualmente es prácticamente imposible porque el coste en materia de tiempo es inasumible. Pero mediante la informática, solo toma unos pocos segundos

[12]Orden de importancia: 'Failed', 'Can't tell', 'Passed'

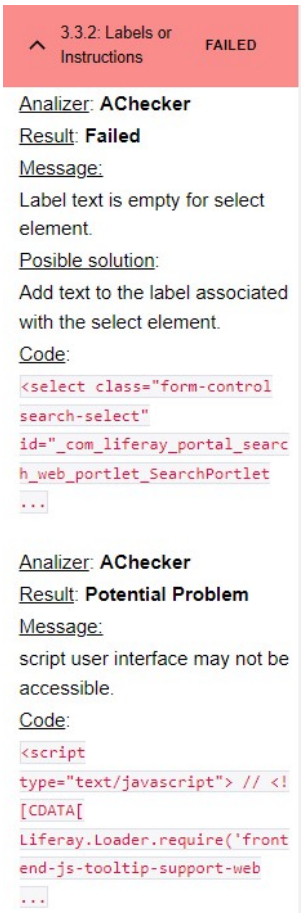

Figura 6.58: Resultados de AChecker

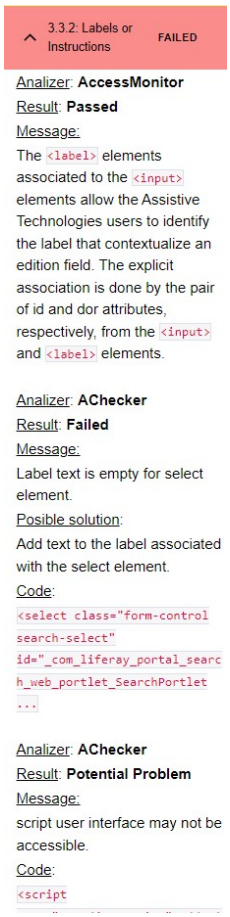

Figura 6.59: Resultados agregados

6.6. Agregación manual de informes

Ya hemos observado cómo se realiza la agregación de informes de servidor, ahora, vamos a ver cómo se realiza en la parte del cliente. Para ello, vamos a ir a la *Report Tool* de W3C[13] y vamos a crear el informe al que le rellenaremos manualmente los criterios 4.1.1 y 4.1.2 y definiremos los resultados como "Passed" y "Can't tell", respectivamente. Finalmente descargamos el informe desde la web en formato JSON y lo agregamos a la extensión. Desplegamos los criterios para que se vean los mensajes manuales. El informe lo realizará el *informador1*. El nombre del evaluador es importante porque después se mostrará su nombre cuando se agreguen varios informes, para saber quién ha escrito qué. El resultado del primer informe se puede ver en la figura 6.60.

Ahora, añadimos un segundo informe que habrá creado el *informador2* en el que diremos

[13] https://www.w3.org/WAI/eval/report-tool/

que el criterio 4.1.2 falla sin duda y el 4.1.3 necesita revisión. Se realizará una **agregación de informes**. El resultado del 4.1.1 será el del *informador1*, porque el *informador2* lo ha dejado como "not checked". El resultado 4.1.3, será justo lo contrario y se quedará el del *informador2*.

Sin embargo, para el criterio 4.1.2, ambos han realizado un análisis. En ese caso, tal y cómo se realiza en el automático, el tipo de resultado será el que más gravedad tenga pero **los mensajes se agregarán**, identificando que evaluador[14] ha escrito qué y qué resultado ha dado cada uno. Eso se puede apreciar en la figura 6.61

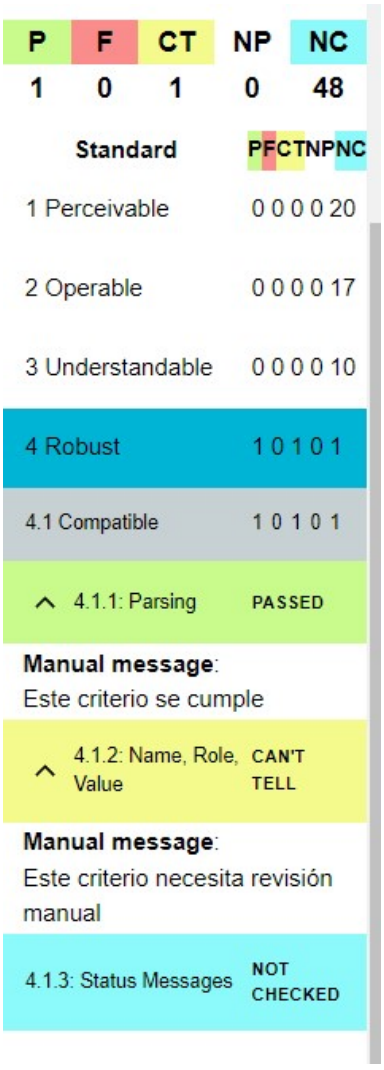

Figura 6.60: Resultados del primer informe

Figura 6.61: Resultados tras agregar el segundo informe

Como podemos observar, en la figura 6.61, al haber informes agregados, pasa a decir el nombre de cada informador, el resultado que han obtenido y finalmente el mensaje que

[14]El nombre del *Evaluator* será el que aparezca, no el del *Commisioner*.

han dado.

Lo bueno de esta agregación es que no tiene límite, se pueden agregar tantos informes como se quiera. Vamos a agregar otro tercer informe, cuyo evaluador será “mikeliturria” y que dirá que el criterio 4.1.1 falla y el 4.1.2 aprueba.

Eso hará que el tipo de resultado del criterio 4.1.1 pase a ser *Failed* y el los mensajes del criterio 4.1.2 y 4.1.3 se agreguen.

Figura 6.62: Resultado tras la tercera agregación

Se observa como aparecen los nombres de los tres informadores, sus conclusiones y sus resultados. Algo muy útil para que varias personas puedan trabajar haciendo un informe sobre la misma página web.

6.7. Descarga y visualización del informe en la web del W3C

A continuación vamos a probar la característica de descargar el informe desde la extensión y posteriormente visualizar los resultados en la *Report Tool* de W3C[15]

Realizaremos un análisis automático a la web de la UPV/EHU, con lo que la información mostrada en el informe serán los resultados automáticos para dicha página.

Cuando descarguemos el documento, la extensión nos redireccionará a la web del W3C. Una vez ahí, haremos click en el botón de "Open Report" y seleccionaremos el informe que ha descargado la aplicación.

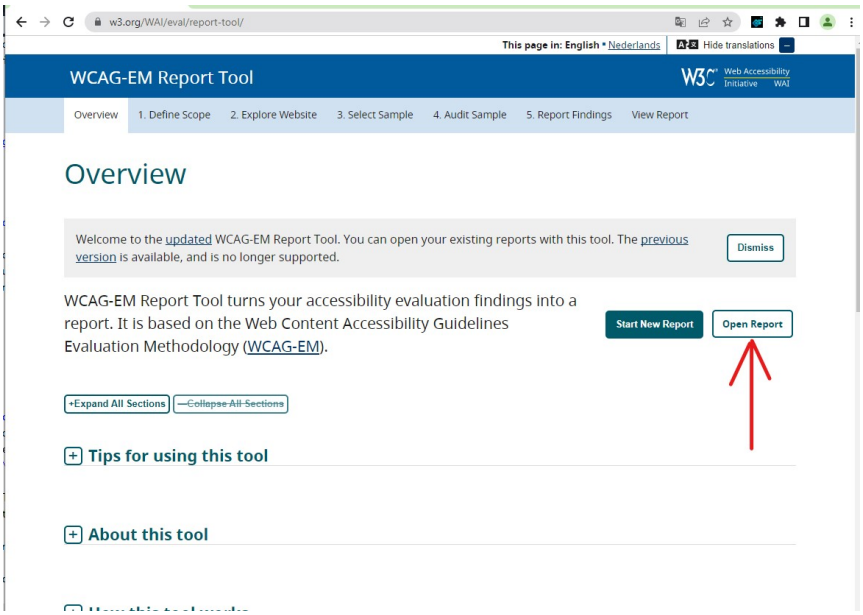

Figura 6.63: Web del W3C

La web podrá abrir el informe sin ningún problema, porque el formato JSON en el que descarga el documento es el mismo que usa la web para la generación de sus informes. Una vez subido el documento, podremos ver cómo se han rellenado los campos de manera automática. Como observamos en la figura 6.64, se pueden ver los criterios que se han rellenado de automáticamente.

Si hacemos click en el botón "View Report" que aparece en la propia figura 6.64, la web nos mostrará el informe en formato HTML. Veamos algunos ejemplos.

[15] https://www.w3.org/WAI/eval/report-tool

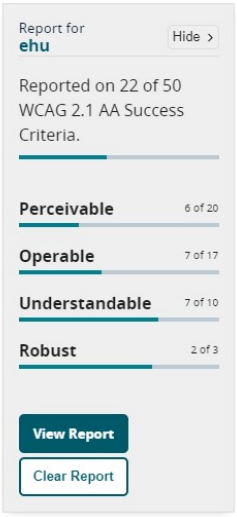

Figura 6.64: Estándares que se han rellenado automáticamente

Primero, mostrará quién ha generado el informe. En este caso, los analizadores son *AccessMonitor* y *AChecker*.

Figura 6.65: Resultado de los evaluadores del informe

También se nos muestra el alcance del informe, en el que podemos ver el nombre de la web, la URL, la versión WCAG y el objetivo de conformidad.

Figura 6.66: Alcance del informe

Respecto a los resultados, podremos ver primero unos resultados generales en los que veremos los criterios cumplidos, fallados, que necesitarán revisión manual, no presentes y no comprobados. Además, se podrá ver el número de criterios que hayan sido comprobados (en este caso, 22 de 50).

Figura 6.67: Resultados del informe

Por último, por cada resultado se mostrarán tres columnas: una con el criterio, otra con el resultado y la tercera con el mensaje. El mensaje es fruto de la agregación de mensajes del informe automático. Primero dirá quién lo ha realizado, usando el carácter '@' seguido del nombre del analizador en negrita y cursiva.

Después, identificará el tipo de error y comenzará con la frase: "The next X was found:", siendo X un valor entre los siguientes: 'ERROR', 'CORRECTION CHECK', 'WARNING' y 'POTENTIAL PROBLEM'. A continuación, se mostrará el mensaje generado por el analizador automático.

Si el resultado del criterio tiene código, se mostrará con la etiqueta `code` y en azul.

Un ejemplo podría ser el criterio 2.4.4, con resultado ''Failed", que tiene un mensaje de *AccessMonitor* y otro de *AChecker* y ambos mensajes tienen código.

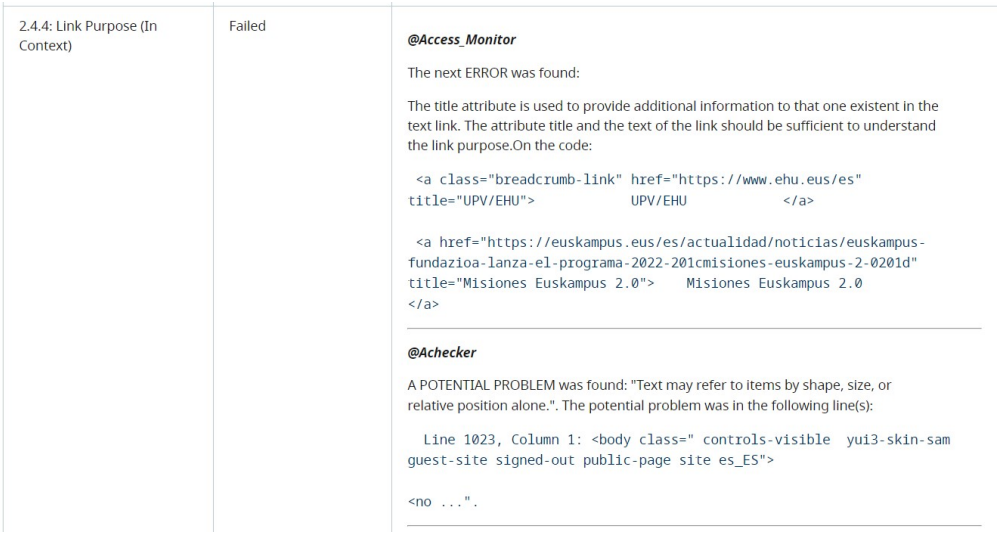

Figura 6.68: Resultado para el criterio 5.4.4

6.8. Errores detectados

La extensión tiene algunos errores menores detectados que no se han solucionado porque no suponen ningún impedimento para el correcto funcionamiento de la extensión y o bien por falta de tiempo o bien porque no se pueden solucionar.

Los tres errores detectados son los siguientes:

1. **La cabecera se sobrepone a la extensión**: Tal y como podemos ver en las figuras 6.40 y 6.47, algunas páginas web que tienen una cabecera fija tienen un pequeño error visual que hace que la cabecera aparezca encima de la extensión. Esto se debe a que el proceso de añadir la extensión a la web es el de crear la barra lateral y un `div` y mover todo el contenido del `body` original a dentro del `div`. Al no encontrarse la cabecera dentro del `body`, no la puede mover y se sobrepone. Se ha dejado así porque no afecta en nada al funcionamiento de la aplicación.

2. **Algunos elementos de *AChecker* no se pueden marcar**: Esto es un error que se sabe pero que no se puede hacer nada para solucionar. Todos los elementos de *AccessMonitor* se marcan porque el analizador emite como resultado la posición del elemento en el árbol DOM, con el que se puede identificar inequívocamente al elemento. Pero *AChecker* no solo no hace eso, sino que además no muestra el código entero.

   Si el código en conflicto es muy largo, *AChecker* lo recorta y le añade tres puntos suspensivos ("...") lo que hace que si el analizador corta demasiado texto el elemento en conflicto pueda ser imposible de encontrar. La mayoría de veces se encuentra el elemento porque el trozo no cortado tiene las suficientes características para identificarlo inequívocamente, pero a veces no lo puede encontrar. En esos casos, se manda una alerta de que el elemento no se ha encontrado. No obstante, pueden ser buscados manualmente usando el inspector de elementos, ya que se conoce parte del código.

3. **Error de la extensión en la vista de desarrolladores en chrome://**: Cuando se visita `chrome://extensions/` se registra como que ha habido un error en la extensión al no haber podido cargar el fichero *background.js*. Este error se desprecia porque se genera debido a que Chrome no permite cargar scripts en direcciónes que empiecen por *chrome://*, pero nosotros no tenemos intención de ejecutarlo en páginas internas de Chrome.

6.9. Agregación automática en las webs probadas

Para que se vea la importancia de la agregación en los informes automáticos, vamos a hacer una comparación en resultados con y sin agregación en todas las webs probadas.

Como se puede observar en las tablas ubicadas a continuación, los resultados se diferencian mucho entre ellos. Esta variación de resultados justifica el uso de la agregación ya que al combinar fuentes de datos diferentes, tal y como vimos en la sección 3.3 Problemática de la evaluación, se obtienen mejores análisis debido a que lo que un analizador no haya comprobado lo ha podido comprobar el otro.

El resultado de web de la UPV/EHU se encuentra en la tabla 6.1.

Web de la EOI de San Sebastián:

TIPO DE RESULTADO	SOLO AM	SOLO AC	AGREGADOS
Passed	2	0	0
Failure	7	5	9
Can't tell	2	13	13
Not Present	0	0	0
Not Checked	39	32	28

Tabla 6.2: Comparación de los resultados para la web de la EOI

Web de la organización de fútbol siete:

TIPO DE RESULTADO	SOLO AM	SOLO AC	AGREGADOS
Passed	1	0	0
Failure	12	6	13
Can't tell	2	12	10
Not Present	0	0	0
Not Checked	35	32	27

Tabla 6.3: Comparación de los resultados para la web de fútbol siete

Web del CTA de Guipúzcoa:

TIPO DE RESULTADO	SOLO AM	SOLO AC	AGREGADOS
Passed	1	0	1
Failure	6	5	9
Can't tell	4	12	12
Not Present	0	0	0
Not Checked	39	33	28

Tabla 6.4: Comparación de los resultados para la web del CTA de Guipúzcoa

Web de Reddit:

TIPO DE RESULTADO	SOLO AM	SOLO AC	AGREGADOS
Passed	2	0	2
Failure	5	0	5
Can't tell	0	0	0
Not Present	0	0	0
Not Checked	43	50	43

Tabla 6.5: Comparación de los resultados para la web de Reddit

AChecker no obtiene resultados para la web de *Reddit* porque, tal y cómo indica la página, "The available memory is not enough to process this size of html."

Web de noticias de EITB:

TIPO DE RESULTADO	SOLO AM	SOLO AC	AGREGADOS
Passed	2	0	0
Failure	5	6	9
Can't tell	3	14	14
Not Present	0	0	0
Not Checked	40	30	27

Tabla 6.6: Comparación de los resultados para la web de noticias de EITB

7. CAPÍTULO

Seguimiento y Control

En este capítulo podremos observar los desvíos que han surgido sobre la planificación del proyecto debido a la complejidad de realizar una correcta estimación de horas.

7.1. Control de la planificación

TAREA	HORAS ESTIMADAS	HORAS REALES	DESVIACIÓN EN HORAS	DESVÍO
P - Planificación	11	12	-1	-9 %
P1- Definición del proyecto	4	6	-2	-50 %
P2- Planificación	7	6	+1	+14.3 %
H- Estudio de las herramientas	21	19	+2	+9.5 %
H1- Estudio de Flask	5	3	+2	+40 %
H2- Estudio de Beatiful Soup	5	6	-1	-20 %
H3- Estudio de Selenium	5	3	+2	+40 %
H2- Estudio avanzado JS	3	4	-1	-33.3 %
H2- Estudio avanzado JQuery	3	3	0	0 %
S- Seguimiento	4	8	-4	-100 %
S1- Control del funcionamiento	2	2	0	0 %
S2- Reuniones con el tutor	2	6	-4	-200 %
DS - Desarrollo de la app de servidor	92	95	-3	-3.3 %
DS.AM - AccessMonitor	45	44	-1	-2 %
DS.AM1- Análisis de la página	5	7	-2	-40 %
DS.AM2- Programación del scraping	30	26	+4	+13.3 %
DS.AM3- Procesado de resultados	10	11	-1	-10 %
DS.AM - AChecker	45	49	-4	-8.9 %
DS.AC1- Análisis de la página	5	6	-1	-20 %
DS.AC2- Programación del scraping	30	31	-1	-3.3 %
DS.AC3- Procesado de resultados	10	12	-2	-20 %
DS1- Agregación de resultados de los dos analizadores	2	2	0	0 %
DJS - Desarrollo de la app JavaScript	85	93	-8	-9.4 %
DJS1- Creación de la estructura	5	3	+2	+40 %
DJS2- Implementación de tareas menores	15	16	-1	-6.6 %
DJS3- Impl. de la agregación	20	25	-5	-25 %
DJS4- Impl. del mostrado de resultados	25	32	-7	-28 %
DJS5- Impl. de resaltar los resultados en la página web	20	17	+3	+15 %

EX- Creación de la extensión	41	44	-3	-7.3 %
EX1- Análisis de extensiones Chrome	5	3	+2	+40 %
EX2- Creación de la extensión	10	11	-1	-10 %
EX3- Integración del código JS	15	18	-2	-13.3 %
EX4- Conexión con la app de servidor	3	2	+1	+33.3 %
EX5- Diseño de la extensión	8	10	-2	-25 %
P- Pruebas en webs reales	11	10	+1	+9.1 %
P1- Pruebas de generación de informe	5	3	+2	+40 %
P2- Pruebas de marcado de elementos	6	7	-1	-16.6 %
M- Memoria	50	65	-15	-23 %
Total	**315**	**346**	**-31**	**-9.8 %**

Tabla 7.1: Tabla de tiempos de la planificación

7.2. Estimación y desviación de las tareas

P- Planificación: No hubo grandes desviaciones salvo en la definición del proyecto, que me llevó menos tiempo de lo esperado.

H- Estudio de las herramientas: Bastante diferencia en lo que a las subtareas respecta ya que la planificación se hizo sin saber cómo funcionaba cada herramienta. Eso provocó que en alguna herramienta llevará más tiempo de lo planificado, como en *Beatiful Soup*, y en otras menos tiempo de lo planificado, como *Flask*, que resulto más sencillo de lo esperado.

S- Seguimiento: Seguimiento fue sin duda el apartado que más variación hubo, llegando incluso a una desvío del 100 %. El desvío fue tan alto porque las duraciones de las reuniones con el tutor fueron el triple de lo esperado.

DS- Desarrollo de la aplicación de servidor: El desvió en el desarrollo de la aplicación de servidor fue mínimo, de poco más del 3 %. Dentro del dicho desarrollo, hubo un desvío mayor en el desarrollo del analizador de *AChecker*, debido a que los datos se mostraban de manera distinta a *AccessMonitor* y, en consecuencia, el código también lo era.

DJS- Desarrollo de la aplicación JavaScript: El desarrollo de la aplicación de JavaScript tuvo unos desvíos considerables en las subtareas. Los desvíos se podrujeron sobre todo en las tareas DJS3 y DJS4, con unos desvíos negativos de 5 y 7 horas respectivamente. El desvío de la tarea DJS4, la tarea con mayor desviación de horas de toda la planificación,

fue causado porque el planteamiento inicial del mostrado de datos no funcionó y se tuvo que replantear y volver a empezar de inicio.

EX- Creación de la extensión: El desvío en la creación de la extensión fue minímo, de menos del 2.5

P- Pruebas de webs reales: La planificación de las pruebas de webs reales se realizaron al alza pero aún así llevaron una hora más de lo planeado.

M- Memoria: La desviación en la memoria fue mayor de lo esperada, con una desviación negativa del 23 %.

7.3. Diagrama de Gantt

A continuación se podrá observar el inicio y fin de las tareas mediante el uso de un diagrama de Gantt.

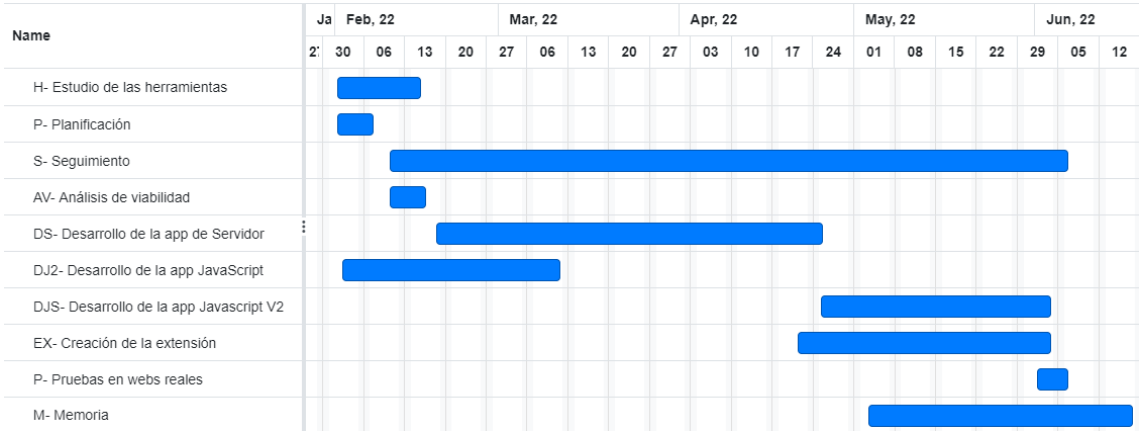

Figura 7.1: Diagrama de Gantt final del proyecto

Si recordamos el diagrama de Gantt realizado para la planificación del proyecto (figura 2.2) podemos observar que ha habido una notable desviación en cuanto a las fechas se refiere, aunque no se ha visto reflejado en las horas de proyecto.

Este retraso se debió a que invertí más tiempo de lo esperado en la asignatura de Programación Lógica, cumpliéndose el riesgo posible **R4**, haciendo que el TFG tuviera un retraso en la finalización de tareas de unos 15 días. Tal y como mencioné en el riesgo R4, un retraso no comprometería en ningún momento la fecha de entrega, y así fue.

8. CAPÍTULO

Conclusiones

En este capítulo se encuentran las conclusiones finales del proyecto. Las conclusiones se dividirán en tres secciones. La primera será la conclusión del proyecto, en la que se hablará, como su nombre indica, de las conclusiones del proyecto. En la segunda se detallarán los conocimientos adquiridos a lo largo de la realización del proyecto y por último se tratarán posibles mejoras que podría tener el proyecto en el futuro.

8.1. Conclusión del proyecto

Se ha conseguido completar de manera satisfactoria la totalidad del proyecto cumpliendo con los plazos de entrega. Es cierto que ha habido una ligera desviación en las horas invertidas y en el inicio y finalización de tareas pero no ha supuesto ningún contratiempo para cumplir los objetivos.

Respecto a la tecnología, tanto la aplicación de servidor como la extensión funcionan de manera estable y fluida. Pese a ser una tecnología diseñada por un estudiante, bien podría ser usado por los desarrolladores en general y probablemente cumplirían sus expectativas sin problema. Esto se debe a que es una aplicación útil, sencilla y con muchas funcionalidades que puede facilitar trabajos tediosos.

A mi parecer, ha sido un proyecto muy extenso que incluso se podría haber dividido en dos TFGs distintos. Se podría haber hecho un TFG independiente del servidor web o API con añadirle un analizador más y quizás hacer que la API además de obtener el JSON en formato W3C tuviera otras funciones. Por ejemplo, devolver únicamente los resultados o almacenar en una base de datos los análisis. La extensión también podría haber sido un TFG independiente si se añadiera alguna funcionalidad más (e.g. editar resultados desde la propia extensión) y si se publicase.

Es cierto que tanto la API como la extensión podrían ser mejores, con lo añadido en el párrafo anterior, pero había que ajustarse a la duración estimada, que ya supera con creces las horas que hay que dedicar al TFG (300) aún habiendo puesto todas las duraciones reales de las tareas a la baja.

No hay mucho más que destacar en las conclusiones del proyecto debido a que no han surgido problemas durante el desarrollo y se ha finalizado el mismo tal y como estaba esperado.

8.2. Conocimientos adquiridos y experiencia personal

Este proyecto me ha servido para aprender muchísimo a nivel tecnológico. Recordemos que tal y cómo vimos en la sección 2.7, Herramientas utilizadas, se han usado un total de 17 herramientas distintas de las cuales 12 son relacionadas directamente con la programación.

Inicié el proyecto con conocimientos básicos de *JavaScript*, *JQuery* y *Python* y, tras fi-

nalizar el proyecto, puedo afirmar que mis conocimientos en estas tres herramientas son avanzados. No es solo el haber aprendido más de las herramientas que antes ya conocía, sino que he aprendido a usar tecnologías muy interesantes como *Flask*, de cuya existencia no tenía conocimiento hasta dar comienzo al proyecto.

Otras herramientas que he aprendido como *Selenium* o *BeatifulSoup* me pueden ser de gran utilidad en un futuro profesional debido a la gran popularidad que tiene el *web scraping*.

Además de las herramientas, he aprendido a desarrollar tanto una aplicación de servidor como una aplicación de cliente y hacer que se puedan conectar entre ellas. Algo que sin duda me vendrá muy bien porque la gran mayoría de aplicaciones web tienen ese tipo de estructura y está bien conocerla y saber las ventajas de tenerla así como los inconvenientes que pueden surgir.

Me ha resultado una experiencia agradable ver en la herramienta tan potente que se ha convertido algo que yo he creado desde cero. No usé ningún repositorio como base, no reutilicé código, todo ha sido escrito por mi expresamente para el proyecto. Mientras desarrollaba el proyecto, cuando algo no me funcionaba, por ejemplo en las etapas finales a la hora de marcar elementos, perdía la perspectiva de todo lo que había hecho. Me fijaba en el marcado pero no era consciente de que los informes automáticos por *web scraping* y la extensión sobre la que trabajaba eran mías. Cuando recuperaba la perspectiva, a riesgo de parecer soberbio, me sentía orgulloso.

Dentro de las experiencias personales no me podría olvidar de destacar el placer que es tener un tutor involucrado en el proyecto. Juan Miguel mostró disponibilidad total además de realizar un seguimiento continuo del proyecto y aportar ideas que a mí no se me podrían haber ocurrido. Además, me ofrecía ayuda en todo momento y no tenía inconvenientes en ayudarme cuando tenía algún problema, sin importar la disponibilidad. Tener un tutor involucrado hizo que resultara muchísimo más ameno realizar el proyecto y tuviera muchas más ganas de realizarlo.

Ha sido un proyecto en el que he trabajado muy a gusto. Era un tema, la accesibilidad, que a priori no era algo por el que tuviera devoción ya que la asignatura de IPC[1] me resultó bastante pesada y muy poco atractiva. Pero como el desarrollo web del proyecto me resultaba muy interesante, al final no me importó trabajar sobre accesibilidad. De hecho, podría decir que he desarrollado un ligero interés por la accesibilidad, pero tampoco algo excesivo.

[1] IPC: Interacción Persona Computador, asignatura obligatoria de 3º de carrera.

8.3. Mejoras futuras

Como he mencionado en la sección 5.5, Extensibilidad y mejoras posibles, el proyecto es extensible y tiene muchas mejoras que se pueden realizar. A mi parecer, se podría realizar un TFG partiendo de mi TFG como base. El nuevo TFG podría tener como alcance los siguientes puntos:

- Análisis de la extensión y aplicación desde el estado base para conocer su funcionamiento.
- Publicación de la extensión en la tienda de Google, explicado en la subsección 5.5.1, Publicación del proyecto. Por supuesto citándome como autor original y sin ánimo de lucro.
- Hacer pública la API, asignandole una dirección IP pública y haciendo que cualquiera en la red pudiera llamarla.
- Búsqueda y agregación de scrapers adicionales. Uno de ellos podría ser la herramienta *Mauve* ++, [mau, 2020] mencionado en el paper de Broccia y otros [Broccia et al., 2020]. El proceso de cómo agregarlo está en la subsección 5.5.2, Extensión del proyecto, pero el alumno tendría que implementar la función que realice el web scraping del nuevo analizador usando *Selenium* y *BeatifulSoup*. Con esto se cubriría la parte del desarrollo en la aplicación de servidor.
- Permitir editar los resultados en la extensión de forma dinámica, mediante el uso *JavaScript* y *JQuery*. Actualmente si quieres editar algo hay que descargar el informe y volver a subirlo. Se podría hacer que se pudiera editar en la propia página. Hay dos ejemplos en la página siguiente. En la figura 8.1 podemos observar como cada mensaje tendría un enlace/botón con el que editaría el texto. Al clicarlo, el mensaje pasaría a ser un `textarea` y aparecería un nuevo botón para salvar los cambios, como se puede observar en la figura 8.2. Con este punto el alumno habría aprendido sobre la parte del cliente.
- Que la edición o mostrado de los resultados se hiciera usando una opción que pueden realizar las extensiones de Chrome, con la que pueden abrir una página interna html que empieza por `chrome-extension://`. Muchas extensiones la realizan, vease como ejemplo la extensión de gestión de proxys *SwitchyOmega* en la figura 8.3 situada dos páginas más adelante.

- Otras opciones que considere el tutor.

Me parece que podría ser interesante para un alumno que no supiera sobre qué hacer el TFG el realizar una extensión del mio, ya que me parece que las mejores son suficientes para que el proyecto sea considerado como un TFG viable y sin llegar a ser difícil ni llegar a tener demasiada carga de trabajo.

Por supuesto, yo estaría dispuesto a echar una mano al alumno en todo momento si no entendiera alguna parte del funcionamiento del código o necesitara ayuda puntual y no tendría ningún problema en contactar con él/ella mediante correo electrónico.

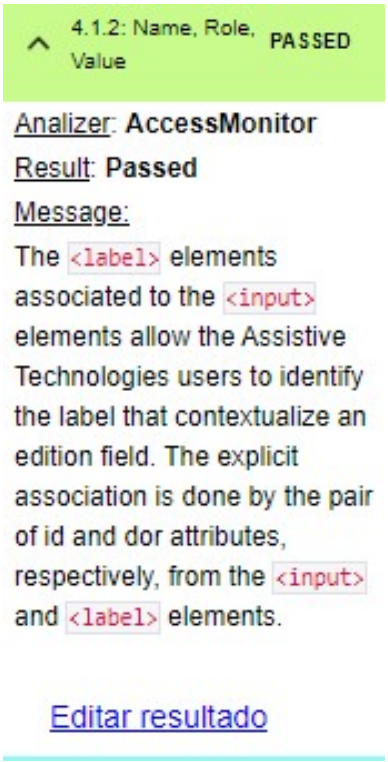

Figura 8.1: Cada mensaje tiene un hiperenlace para editar texto

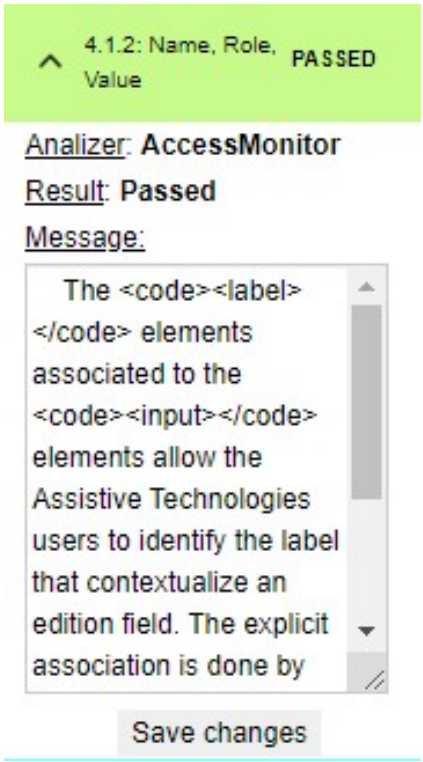

Figura 8.2: El texto se convierte en un textarea y aparece un botón para guardar los datos.

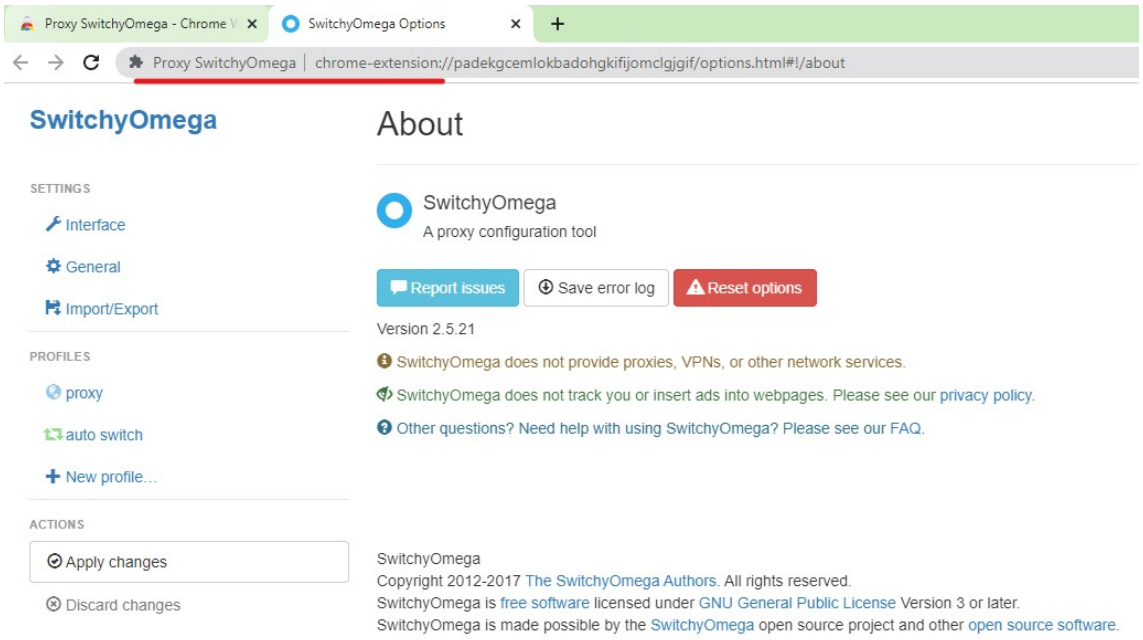

Figura 8.3: Página HTML generada desde una extensión

Anexos

A. ANEXO

Instalación del proyecto en Windows/Linux

En este capítulo se explicará como instalar el proyecto desde cero en un nuevo equipo.

A.1. Instalación de Python

El proyecto necesita tener *Python* instalado para poder ejecutarse. Si no está instalado, hay que ir a la página web de *Python* [pyt, 1991], visitar la sección de "Downloads" y descargar el ejecutable de la última versión disponible. Una vez finalizada la descarga, se ejecutará el ejecutable y se instalará *Python*.

Para comprobar que se haya instalado correctamente, abrimos la consola de comandos de Windows y ejecutamos el siguiente comando: `python --version`. Nos debería aparecer la versión instalada.

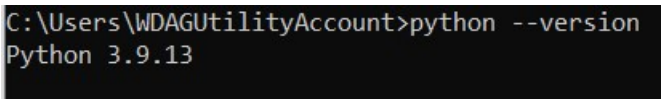

Figura A.1: Se puede observar la versión de Python

A.2. Instalación de Git

Se recomienda que *Git* esté instalado en el equipo para poder clonar el proyecto de manera satisfactoria. Si no está instalado, habrá que visitar la web de descarga de *Git* [git, 2007] y descargar el ejecutable que más convenga a nuestro equipo.

Una vez descargado e instalado, abriremos el *Bash* de Git y comprobaremos la versión ejecutando `git --version`. Si nos aparece la versión, es que está bien instalado.

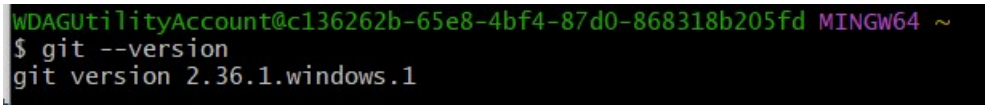

Figura A.2: Se puede observar la versión de Git

A.3. Clonación del proyecto

Ahora procederemos a clonar el proyecto. Es un proceso muy sencillo ya que el código se encuentra en un repositorio público de *GitHub* en el siguiente enlace: `https://github.com/Itusil/TFG`.

Creamos la carpeta en la que querremos que se instale el proyecto y accederemos a ella. Dentro de la carperta haremos click derecho y clicaremos sobre “Git Bash here”.

Cuando se abra la consola ejecutaremos los siguientes comandos

```
git init;
git clone https://github.com/Itusil/TFG.git;
```

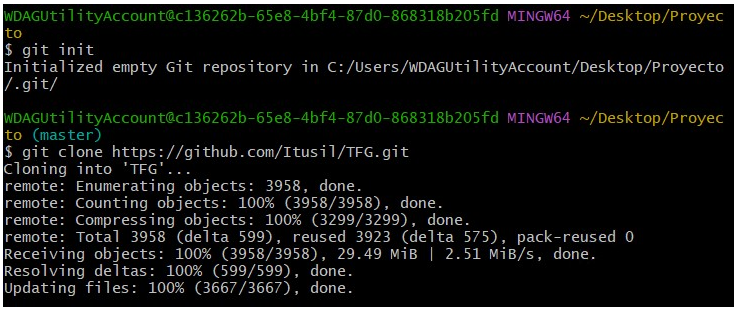

Figura A.3: Comandos ejecutados

Si no se desea usar *Git*, aunque es lo recomendable, *GitHub* permite descargar el fichero ZIP del proyecto desde el propio repositorio.

A.4. Puesta a punto de la aplicación de servidor

Una vez está instalado el proyecto, vamos a instalar los complementos necesarios para que funcione. Para ello, ejecutaremos el **Windows PowerShell** como administrador si nos encontramos en **Windows** o la **consola de comandos** en el caso de trabajar en **Linux**. Mediante el comando `cd` nos situaremos en la carpeta *flask* de la raíz del proyecto.

(Solo en **Windows**) Una vez ahí, primero ejecutaremos el comando que permita ejecutar scripts. Para ello ejecutamos:

```
powershell Set-ExecutionPolicy RemoteSigned
```

Una vez ejecutado, tendremos que crear de nuevo el entorno virtual.

Para crear el entorno virtual primero **eliminaremos** la carpeta `venv` y la carpeta `__pycache` de la raíz del proyecto. Una vez eliminado, desde el *PowerShell* ejecutamos:

En **Windows**:

```
py -3 -m venv venv
```

En **Linux**:

```
conda create -n tfg_iturria python=3.9
```

Por último, activamos el entorno virtual:

En **Windows**:

```
.\venv\Scripts\activate
```

En **Linux**:

```
source activate tfg_iturria
```

Ahora veremos que sale `(venv)` delante de la línea de comandos en Windows (figura A.4 y `tfg_iturria` si se ha realizado en Linux (figura A.5). Eso significa que hemos accedido correctamente al entorno virtual.

Figura A.4: Activación del entorno virtual en Windows

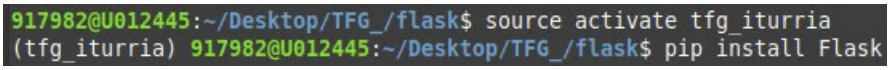

Figura A.5: Activación del entorno virtual en Linux

Ahora, tal y como se menciona en la sección 5.1.2, Creación del entorno virtual y puesta a punto de *Flask*, instalaremos los complementos necesarios para ejecutar la aplicación de servidor.

```
$ pip install Flask
$ pip install flask_cors
$ pip install bs4
$ pip install datetime
$ pip install selenium
$ pip install chromedriver-autoinstaller
```

Una vez instalados todos los complementos, ya solo quedan dos pasos. Primero, se especificará que *Flask* trabajará en entorno de desarrollo ejecutando el siguiente comando:

En **Windows**:

```
$env:FLASK_ENV = "development"
```

En **Linux**:

```
export FLASK_ENV =development
```

Como hemos mencionado, la aplicación de servidor necesitará tener la versión del driver compatible con la versión del Chrome del usuario, pero esto no es algo de lo que debamos preocuparnos. Tal y como vimos en la subsección 5.3.10, Funciones destacadas: Autoinstalación de *Chromedriver*, éste es un proceso que se realiza de manera automática. No obstante, **en Linux**, será necesario dar permiso de ejecución al driver, para ello, desde la carpeta que se encuentre dentro de *chromedrivers*, una vez instalada, ejecutaremos el siguiente comando:

```
sudo chmod +x chromedriver
```

El último paso será iniciar *Flask*. Para eso ejecutaremos el comando a continuación y veremos cómo *Flask* ya está funcionando. Se puede observar en la figura A.6.

```
flask run
```

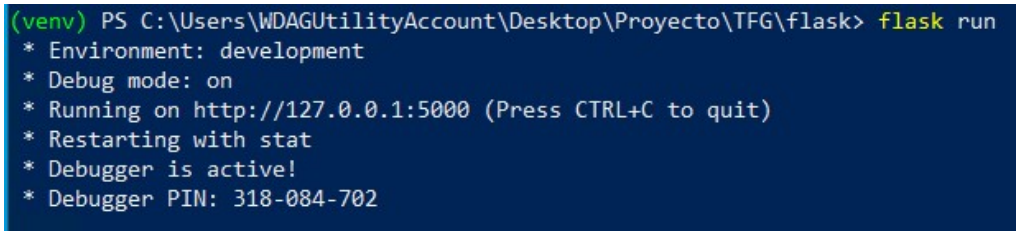

Figura A.6: Flask se está ejecutando

Solo quedaría instalar la extensión, cuyos pasos están el sección 6.1, Preliminares.

Bibliografía

[pyt, 1991] (1991). *Python.* Python Software Foundation. https://www.python.org/.

[js, 1995] (1995). *JavaScript*. Mozilla Fundation. https://developer.mozilla.org/es/docs/Web/JavaScript.

[jqu, 2006] (2006). *JQuery*. John Resig. https://jquery.com/.

[bea, 2007] (2007). *Beautiful Soup Documentation.* crummy software. https://www.crummy.com/software/BeautifulSoup/bs4/doc/.

[git, 2007] (2007). *Instalación de Git en Windows.* Git-SCM. http://git-scm.com/download/win.

[sph, 2008] (2008). *Sphinx: Genera documentación para Pyhton de manera automática.* Sphinx. https://www.sphinx-doc.org/en/master/.

[sub, 2008] (2008). *Sublime Text 3: Una herramienta de edición de texto.* Sublime Text. https://www.sublimetext.com/.

[fla, 2010a] (2010a). *Flask: Crea aplicaciones web rápidamente.* Pallets Projects. https://flask.palletsprojects.com/en/2.1.x/.

[fla, 2010b] (2010b). *Proceso de la instalación de Flask.* Pallets Projects. https://flask.palletsprojects.com/en/2.1.x/installation/.

[ove, 2011] (2011). *Overleaf: Editor en la nube de Latex.* Overleaf. https://www.overleaf.com/.

[glo, 2018] (2018). *GlooMaps: Crea diagramas de EDT de forma gratuita.* HotGloo. https://www.gloomaps.com/.

[chr, 2019] (2019). *Module selenium-webdriver/chrome.* Selenium. https://www.selenium.dev/selenium/docs/api/javascript/module/selenium-webdriver/chrome.html.

[fon, 2020] (2020). *Google Fonts: Librería de fuentes de texto y logos gratuitos.* Google LLC. https://fonts.google.com/.

[mau, 2020] (2020). *Mauve ++, una herramienta de validación de accesibilidad.* Human Interfaces In Information Systems Laboratory. https://mauve.isti.cnr.it/.

[dia, 2021] (2021). *Diagrams: Crea digramas de manera sencilla y gratuita.* JGraph Ltd. https://app.diagrams.net/.

[ext, 2022a] (2022a). *Add-on de AInspector*. Firefox Addons Store. https://addons.mozilla.org/es/firefox/addon/ainspector-wcag/.

[wai, 2022] (2022). *Class WebDriverWait.* Selenium. https://www.selenium.dev/selenium/docs/api/java/org/openqa/selenium/support/ui/WebDriverWait.html.

[par, 2022] (2022). *Concurrent.futurews — Launching parallel tasks.* Python Software Foundation. https://docs.python.org/3/library/concurrent.futures.html.

[W3C, 2022a] (2022a). *Evaluating Web Accessibility Overview*. W3C Web Accessibility Initiative (WAI). https://www.w3.org/WAI/test-evaluate/.

[ext, 2022b] (2022b). *Extensión Accessible web helper*. Google Chrome Store. https://chrome.google.com/webstore/detail/accessible-web-helper/gdnpkbipbholkoaggmlblpbmgemddbgb.

[ext, 2022c] (2022c). *Extensión ACheck Helper*. Google Chrome Store. https://chrome.google.com/webstore/detail/achecker-helper/eoleepgjlfafppekfgofmgbejadnnphn.

[ext, 2022d] (2022d). *Extensión Axe DevTools*. Google Chrome Store. https://chrome.google.com/webstore/detail/axe-devtools-web-accessib/lhdoppojpmngadmnindnejefpokejbdd.

[ext, 2022e] (2022e). *Extensión de Wave*. Google Chrome Store. https://chrome.google.com/webstore/detail/wave-evaluation-tool/jbbplnpkjmmeebjpijfedlgcdilocofh.

[ext, 2022f] (2022f). *Extensión LERA*. Google Chrome Store. https://chrome.google.com/webstore/detail/lera/neninfnjnhknniefcpognoooalfdaofc.

[ext, 2022g] (2022g). *Extensión SiteImprove*. Google Chrome Store. https://chrome.google.com/webstore/detail/siteimprove-accessibility/djcglbmbegflehmbfleechkjhmedcopn.

[cor, 2022] (2022). *Flask_cors: Una extensión que permite utilizar CORS en Flask.* https://flask-cors.readthedocs.io/en/latest/.

[W3C, 2022b] (2022b). *Making the web accesible*. W3C Web Accessibility Initiative (WAI). https://www.w3.org/WAI/.

[onl, 2022] (2022). *Online Gantt: Crea diagramas de Gantt de forma gratuita.* Online Gantt. https://www.onlinegantt.com/.

[chr, 2022] (2022). *Repositorio con versiones del Chromedriver.* Google LLC. http://chromedriver.storage.googleapis.com/index.html.

[sel, 2022] (2022). *Selenium with Python.* Apache Software Foundation. https://selenium-python.readthedocs.io/.

[W3C, 2022c] (2022c). *Using Combined Expertise to Evaluate Web Accessibility*. W3C Web Accessibility Initiative (WAI). https://www.w3.org/WAI/test-evaluate/combined-expertise/.

[W3C, 2022d] (2022d). *WCAG 2 Overview*. W3C Web Accessibility Initiative (WAI). https://www.w3.org/WAI/standards-guidelines/wcag/.

[W3C, 2022e] (2022e). *WCAG-EM Overview: Website Accessibility Conformance Evaluation Methodology*. W3C Web Accessibility Initiative (WAI). https://www.w3.org/WAI/test-evaluate/conformance/wcag-em/.

[her, 2022] (2022). *Web Accessibility Evaluation Tools List by W3C*. W3C Web Accessibility Initiative (WAI). https://www.w3.org/WAI/ER/tools/.

[W3C, 2022f] (2022f). *Web Content Accessibility Guidelines (WCAG) 2.1*. W3C Web Accessibility Initiative (WAI). https://www.w3.org/TR/WCAG21/.

[Broccia et al., 2020] Broccia, G., Manca, M., Paternò, F., and Pulina, F. (2020). Flexible automatic support for web accessibility validation. *Proceedings of the ACM on Human-Computer Interaction*, 4(EICS):1–24.

[Clark, 2022] Clark, D. (2022). *JSFormat: Complemento para el formateo de documentos JSON*. https://github.com/jdavisclark/JsFormat.

[Jo, 2021] Jo, Y. (2021). *Chromedriver autoinstaller*. Seoul National University, Computer Science & Engineering department. https://pypi.org/project/chromedriver-autoinstaller/.

[Vigo et al., 2013] Vigo, M., Brown, J., and Conway, V. (2013). Benchmarking web accessibility evaluation tools: measuring the harm of sole reliance on automated tests. In *Proceedings of the 10th International Cross-Disciplinary Conference on Web Accessibility*, pages 1–10.