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

Aplicación para análisis de mercado de las extensiones de Google Chrome

Autor

Unai Ahedo de Luis

2021

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

Aplicación para análisis de mercado de las extensiones de Google Chrome

Autor

Unai Ahedo de Luis

Director

Oscar Diaz Garcia

Resumen

Esta memoria corresponde al Trabajo de Fin de Grado (TFG a partir de ahora) "*Aplicación* para análisis de mercado de extensiones de Google Chrome". Se ha implementado una aplicación, a modo prueba de concepto, la cual sirva para ayudar a los desarrolladores interesados en desarrollar extensiones de Google Chrome, mostrándoles una representación de las necesidades de los usuarios. El objetivo del proyecto es automatizar una función, la cual hecha manualmente, sería demasiado costosa tanto en tiempo como en recursos, a parte, al automatizar dicha función, se evitan los errores que podría conllevar el hacerlo manualmente.

Índice general

Índice de figuras

Índice de tablas

1. CAPÍTULO

Introducción

El mercado de las extensiones de navegador es amplio y dichas extensiones abarcan muchas funcionalidades, las cuales no siempre son necesarias o deseadas por el usuario que va a hacer uso de ellas, o puede darse el caso de que las funcionalidades deseadas por el usuario, no se vean presentes en las extensiones. Para evitar este problema, a la hora de desarrollar una extensión para un navegador, es necesario conocer las necesidades de los usuarios que la van a utilizar, y también es importante saber qué puede desagradar a los usuarios al utilizar las extensiones. Este punto puede hacer que un desarrollador de extensiones tenga incertidumbre a la hora de realizar la implementación y al desarrollador puede no interesarle emplear tiempo en implementar una funcionalidad qué puede llegar a no tener éxito, e incluso desagradar a los usuarios.

Para evitar este problema, es necesario conocer el mercado y la demanda de los usuarios sobre las funcionalidades deseadas en las extensiones. Es decir, dentro de un tipo de extensiones, se necesita conocer qué funcionalidades complacen a los usuarios, y cuáles pueden ser negativas según el criterio del usuario, para no implementarlas.

En este documento se detalla el proyecto realizado describiendo el diseño y la implementación de una aplicación que permite realizar un análisis de mercado de las extensiones de Google Chrome. Mediante esta aplicación, un desarrollador podrá realizar una búsqueda de las extensiones que tienen un cierto parecido a la que tiene pensada desarrollar, y la aplicación de manera automática, recopilará la información (comentarios y opiniones) de los usuarios sobre dichas extensiones, para ofrecer al usuario de la aplicación una lista con las posibles funcionalidades a implementar en la extensión. De esta manera, se automatiza un proceso que manualmente puede ser muy lento.

- **Capítulo 2:** descripción del contexto en el cual se encuentra el TFG.
- **Capítulo 3:** planificación inicial del proyecto, en donde se describen apartados importantes como: alcance, tiempo, riesgos...
- **Capítulo 4:** descripción de las herramientas utilizadas para la implementación del proyecto. Breve explicación del funcionamiento de las mismas.
- **Capítulo 5:** diseño utilizado a la hora de implementar el proyecto en la fase de desarrollo, tanto de la aplicación como de los servicios.
- **Capítulo 6:** descripción del desarrollo de la implementación del proyecto. Explicación de las funcionalidades principales del programa, y los problemas y soluciones surgidos durante el desarrollo del mismo.
- **Capítulo 7:** pruebas de funcionamiento y con usuarios.
- **Capítulo 8:** seguimiento y control del proyecto, donde se detallan los cambios realizados en la primera planificación, así como las desviaciones finales en el tiempo.
- **Capítulo 9:** conclusiones finales del proyecto, así como un apartado de posibles mejoras y líneas a futuro del proyecto.

2. CAPÍTULO

Antecedentes

Dentro del mundo de las extensiones se ha estudiado la distribución de las extensiones en Google Chrome, donde se da una distribución en base a la Teoría de la Larga Cola (Véase Figura 2.1). Un ejemplo de esta Teoría de Larga Cola es la tienda de Amazon, tiene muchos productos muy variados y no se especializa en nada en concreto, por ello acaba vendiendo menos unidades pero productos mucho más variados, con los cuales puede satisfacer las necesidades de muchos mas compradores. En relación a la teoría de la larga cola, en el trabajo de investigación desarrollado dentro del grupo se extrajeron las siguientes conclusiones:

- Los usuarios prefieren las extensiones de nicho, es decir, a mayor número de instalaciones de una extensión, menor es la satisfacción de los usuarios. Por ello, las extensiones que menos instalaciones tienen, suelen suplir mejor las necesidades de los usuarios, pero hay menos oferta de las mismas. Esto sucede porque las extensiones de nicho, por lo general, se adaptan mejor a las diferentes necesidades de los usuarios, de la misma manera qué el ejemplo anterior de la tienda de Amazon.
- Dichas extensiones de nicho, son las que se encuentra al final de la Teoría de Larga Cola, mediante la cual se explica como dentro de un sector, hay unos pocos productos muy populares (llamados blockbusters o hits), pero muchos poco populares. Según dicha teoría, este nicho de mercado va a acabar superando al mercado prioritario.
- Lo que caracteriza a las extensiones de nicho, es qué todas cumplen una misma práctica (por ejemplo, tomar anotaciones), pero se utilizan para diferentes propósi-

tos (por ejemplo, tomar apuntes o compartir opiniones), por consiguiente las funcionalidades pueden variar un poco entre ellas.

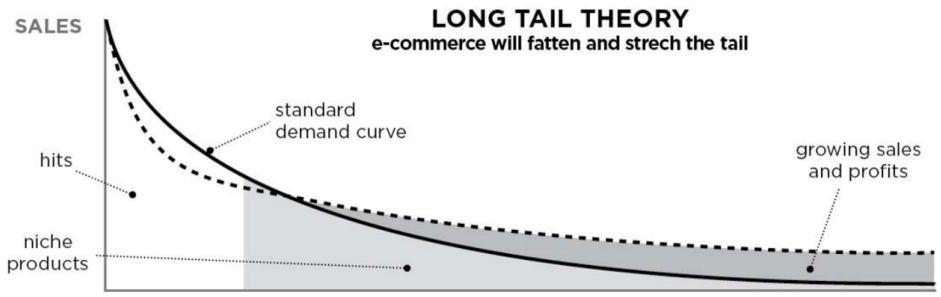

Figura 2.1: Teoría de larga cola [Dik, 2018].

Google Chrome no dispone de una API con la cual trabajar para extraer los datos de las extensiones, de manera que es necesario realizar web-scrapping de la Web Store de Chrome. Por lo cual, la idea de este TFG surge de la necesidad de que los desarrolladores de extensiones puedan hacer un análisis de mercado automatizado.

De esta manera el desarrollador puede encontrar un hueco a su extensión de nicho, en una dominio dentro de las extensiones de navegador qué otras extensiones no abordan, la cual está dirigida a un grupo muy específico de usuarios (donde dichos *blockbusters* no suplen las necesidades), en el mercado de las extensiones y poder ofrecer alternativas al resto de extensiones, con más posibilidad de éxito que el resto, gracias al uso de la aplicación resultante de este TFG, sabrá qué funcionalidades satisfacen más a los usuarios. Para realizar esta clasificación se utilizará el modelo Kano, descrito en el siguiente apartado.

2.1. Modelo Kano

El modelo Kano es una teoría de desarrollo de productos y satisfacción de clientes. Esto modelo tiene como objetivo principal qué el fabricante pueda identificar cuales son los atributos valorados por los consumidores, y de esta manera, poder ofrecer un producto acorde a esas valoraciones.

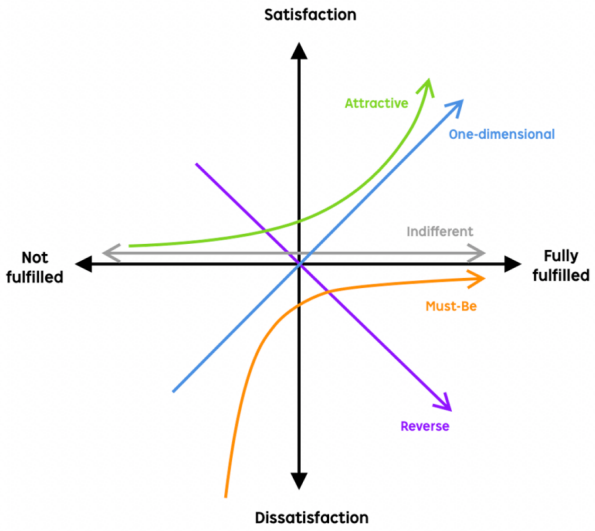

Figura 2.2: Modelo Kano a lo largo del desarrollo [IZO, 2020].

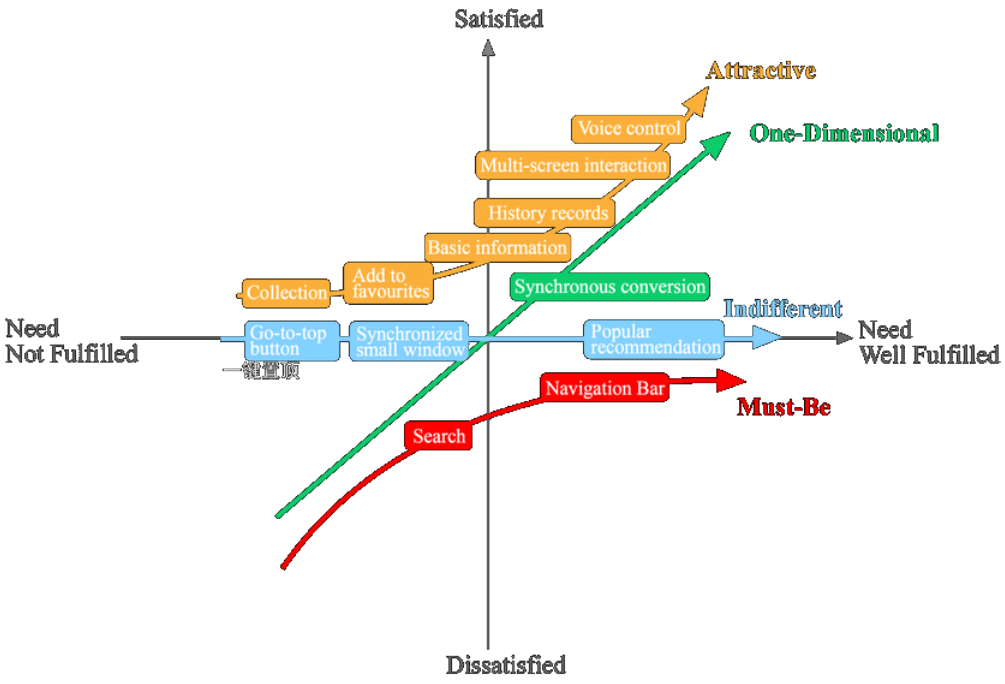

Figura 2.3: Ejemplo de modelo Kano finalizado [Chen and Li, 2019].

Se describen las siguientes calidades, ordenadas de mayor importancia a menor:

- Calidad requerida (*Must-be Quality*): Estos atributos son los que el usuario da por sentado, es decir, atributos que si están no pasa nada, pero si no están, generan insatisfacción. Por este motivo, estos atributos deben estar presentes obligatoriamente.

Por ejemplo: un teléfono móvil no permite hacer llamadas telefónicas, por lo cual genera insatisfacción, pero en caso de poder realizar llamadas, no incrementa la satisfacción del cliente.

- Calidad unidimensional (*One-dimensional Quality*): Estos atributos generan satisfacción cuando están presentes, e insatisfacción cuando no lo están. Por ejemplo: Un teléfono móvil promete tomar fotografías a calidad 4K, valiendo lo mismo que uno que solo saca fotos a FullHD, esto generaría satisfacción. Pero si en vez de fotos en 4K las tomase en 2K, generaría insatisfacción.

- Calidad atractiva (*Attractive Quality*): Estos atributos generan satisfacción cuando están presentes, pero no generan insatisfacción cuando no están presentes. Por ejemplo: Un teléfono móvil con cámara en 360º, en caso de estar presente generaría satisfacción, pero en caso de no estarlo, no genera insatisfacción porque es un atributo que el usuario no espera que esté presente.

- Calidad indiferente (*Indifferent Quality*): Estos atributos no tienen impacto negativo ni positivo en la satisfacción del cliente. Por ejemplo: El grosor del embalaje de un móvil. Puede qué este grosor este diseñado para el transporte o fabricación del mismo, pero no afecta a los usuarios.

- Calidad inversa (*Reverse Quality*): La existencia de estos atributos son percibidos de manera negativa por los usuarios. Por ejemplo: Que un teléfono móvil tenga menos resolución de pantalla que el resto de su precio parecido.

En este proyecto el modelo Kano lo utiliza el desarrollador, en función de su criterio, a la hora de clasificar las funcionalidades agrupadas por él mismo, y estas vienen extraídas de los comentarios de los usuarios en las páginas de las extensiones de la Chrome Web Store.

3. CAPÍTULO

Planificación

En este capítulo se detalla la planificación del proyecto, describiendo el alcance, el tiempo, la dedicación, comunicación, información y los interesados. El objetivo de la planificación es que el proyecto satisfaga los objetivos definidos previamente en la captura de requisitos, teniendo en cuenta que las incidencias tengan el menor impacto posible. Por ello es necesario planificar adecuadamente las tareas del proyecto, junto a un plan de gestión de riesgos, mediante el cual, estos sean lo menor perjudiciales posibles, en caso de que ocurran. Durante el desarrollo del proyecto, algunos de los puntos definidos en este capítulo han sido modificados, y estas desviaciones están recogidas en el Capítulo 8.

3.1. Descripción del producto

El programa tiene como objetivo, ayudar a un usuario a realizar un análisis de mercado de las extensiones de Google Chrome. Es decir, dicho cliente antes de comenzar a desarrollar una extensión para Chrome, será capaz de saber que funciones son necesarias para complacer a los clientes que vayan a utilizar dicha extensión. Esto se realiza analizando las extensiones que ya existen, y comprobando las funciones o funcionalidades que contienen, y si dichas funciones satisfacen a los usuarios. Para realizar este análisis de mercado son necesarias una forma de búsqueda y una forma de representación.

Para la búsqueda se utilizará una consultada basada en las 4 Ws. Es decir, a parte de introducir el *objetivo* de la extensión, opcionalmente se podrán introducir las respuestas a las 4 Ws, que son las preguntas: *Why?*, *Where?*, *How?* y *What?*.

Una vez respondidas dichas preguntas, se filtrarán las extensiones que las cumplan, y el

usuario podrá seleccionar las deseadas de dicha lista. Tras esto, mediante *web scrapping* se extraerán los *tags*, de los comentarios de los usuarios, asociados a las extensiones. Estas *tags* son palabras claves como podrían ser: copiar texto, exportar como PDF... Las cuales el usuario agrupará en *features*, que serán las características a implementar en su extensión, como podrían ser: edición de texto, exportación del archivo... Tras esto, estas *features* se representarán mediante el modelo Kano.

3.2. Gestión del Alcance

El proyecto tiene como meta principal desarrollar una aplicación de escritorio mediante la cual un usuario, pueda realizar un análisis de mercado sobre las extensiones de la Web Store de Google Chrome. Es decir, mediante el uso de este programa, se generará un modelo Kano (ver Figura 2.3) sobre las características que debería tener una extensión para que tenga éxito, en base a las *reviews* y descripciones de otras extensiones similares. De esta manera, una persona que quiera desarrollar una nueva extensión para Google Chrome, podrá disponer de este programa, para facilitar el diseño previo de las funcionalidades de dicha extensión, y conseguir el mayor éxito posible. Para poder llevar adelante el proyecto, se necesita descomponer el trabajo en paquetes (Véase 3.2.4). Además se deben controlar los riesgos y sus planes de acción (Véase 3.4).

3.2.1. Objetivos

Como ya se ha comentado previamente, el objetivo principal del proyecto es desarrollar una aplicación que realice análisis de mercado de las extensiones de Google Chrome. A parte de este, hay otros objetivos principales, que se van a detallar en este apartado. En caso de que el tiempo lo permita, se completarán los objetivos secundarios, los cuales servirían para mejorar la calidad del proyecto.

- **Obtener un filtrado de las extensiones:** Realizar un filtrado para que el usuario pueda filtrar y escoger las extensiones que le interesen a la hora de realizar el análisis de mercado, de esta manera, por ejemplo, el usuario podrá decidir si solo quiere un análisis de las extensiones más exitosas.
- **Recopilación de datos mediante web scrapping:** Como el análisis de mercado se realiza sobre las extensiones ya existentes en la WorkShop de Google Chrome, la aplicación se encargará de recopilar todos los datos necesarios de las extensiones que se van a utilizar para generar el modelo de Kano. Estos datos pueden ser: las

reviews, las puntuaciones, cantidad de descargas, la descripción de la extensión... Con esta información recopilada se generarán los *tags*.

- **Agrupación de *tags* en *features*:** Una vez que se hayan recopilado los *tags* asociados a las extensiones, se le dará la opción al usuario de que las agrupe en *features*, las cuales servirán para generar el modelo de Kano.
- **Generación de modelo Kano:** Una vez que se dispongan de las *features*, se proporcionarán los medios para que el usuario pueda clasificar las mismas en los diferentes apartados del modelo Kano.

Objetivos secundarios

Una vez cumplidos los objetivos principales, y en caso de disponer del tiempo necesario, se procederá a desarrollar los objetivos secundarios.

- **Recuperar historial de análisis previos:** Se añadirá la opción de que el usuario pueda recuperar los análisis previos, es decir, que pueda cargar una búsqueda que haya hecho previamente, y podrá realizar modificaciones.
- **Automatizar la generación del modelo Kano:** Una vez que el usuario haya generado todas las agrupaciones, la aplicación se encargará de realizar una primera categorización (a modo boceto) de las mismas, dando la opción al usuario de modificar en caso de no ser correcta.
- **Mostrar la agrupación de los *tags* como una *tag cloud*[1]:** La idea principal es que las *tags* aparezcan en una lista, y que el usuario las seleccione de dicha lista a la hora de agruparlas. Este objetivo trata de sustituir dicha lista por una *tag cloud*.
- **Permitir guardado en la nube:** Tras realizar el análisis, dar la opción al usuario de subir dicho resultado a la nube (Drive, Dropbox...).

3.2.2. Requisitos

Requisitos funcionales

- **Filtrado**

[1] Tag cloud: https://media.nngroup.com/media/editor/2012/11/18/wordle-word-cloud-applications.png.

  - **R-01:** La aplicación permitirá filtrar las extensiones en base a las 4Ws.
  - **R-02:** La aplicación permitirá al usuario escoger las extensiones que desee de las filtradas previamente.
  - **R-03:** Al realizar varias búsquedas de extensiones, la aplicación mostrará las extensiones nuevas que se hayan incluido o descartado en cada búsqueda.

- **Visualización**
  - **R-04:** La aplicación mostrará un resumen de manera gráfica (descargas, puntuación...) de las extensiones al usuario tras el filtrado.
  - **R-05:** La aplicación permitirá editar el modelo de Kano al usuario, para que pueda adaptarlo a su criterio.

- **Análisis de datos**
  - **R-06:** La aplicación recogerá la información mediante *web scrapping* de la Workshop de Google Chrome.
  - **R-07:** La aplicación recogerá tanto la descripción de la extensión como las reseñas de los usuarios.

- **Exportación**
  - **R-08:** La aplicación permitirá exportar el modelo Kano en diferentes formatos (imagen, CSV...).

Requisitos no funcionales

- **Rendimiento:**
  - **R-09:** Realizar *web scraping* puede conllevar mucho tiempo de ejecución (sobre todo trabajando con extensiones grandes), por ello la aplicación tendrá que ser lo más rápida posible en el resto de aspectos.

- **Usabilidad:**
  - **R-10:** La aplicación debe ser intuitiva y fácil de utilizar, aunque sea la primera vez que se usa.
  - **R-11:** La aplicación debe proporcionar mensajes de error, ayuda e información para dar un *feedback* al usuario que la esté utilizando.

- **Fiabilidad:**

  - **R-12:** La aplicación debe tener casos de prueba para comprobar el correcto funcionamiento de la misma.

- **Modificabilidad:**

  - **R-13:** La aplicación debe ser fácilmente extensible, en caso de que en un futuro se quieran añadir funcionalidades.

Exclusiones del proyecto

- **E-01:** La aplicación no debe ser totalmente autónoma. Las agrupaciones de *tags* y el modelo Kano debe quedar en manos del usuario.

3.2.3. Fases del proyecto

Debido a que se va a trabajar con nuevas tecnologías con las cuales no se dispone de un gran dominio, y dado a que se quiere disponer de un prototipo previo a la finalización del proyecto, se va a realizar un ciclo de vida del proyecto **Incremental**. De esta manera, entre los diferentes incrementos, se podrán ir adquiriendo las habilidades para utilizar correctamente las herramientas. A parte, gracias a este ciclo de vida, se pueden ir alcanzando las funcionalidades gradualmente, y se puede proporcionar un *feedback* al cliente, en este caso al director del proyecto.

En un ciclo de vida incremental, es complicado calcular las fechas de inicio y final de las fases, pero se ha realizado una estimación (Véase 3.2.7), estas estimaciones tienen en cuenta la fecha máxima de entrega de la memoria del proyecto y la defensa del mismo. El objetivo en cada incremento es poder mostrar un prototipo, o la aplicación final, al cliente.

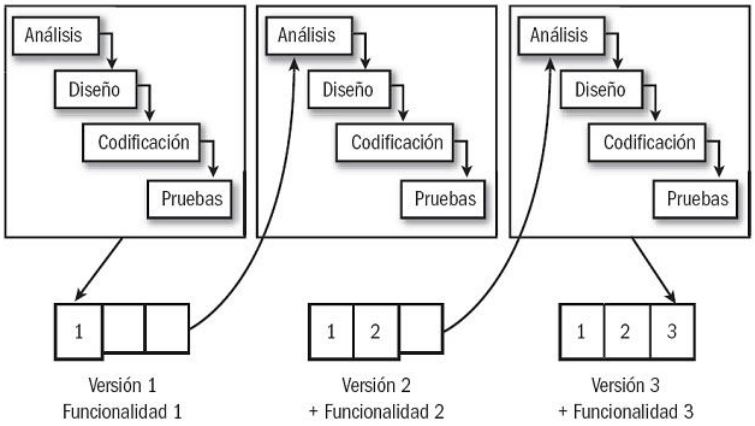

Figura 3.1: Ciclo de vida incremental del proyecto [Salas, 2017].

3.2.4. Descomposición de tareas

La Estructura de Descomposición de Trabajo (EDT) del proyecto se ha creado teniendo en cuenta el ciclo de vida incremental del proyecto. Mediante el EDT, se pretende dividir las tareas en paquetes más pequeños, y de esta manera, se pretende que sean más fáciles de entender y gestionar.

En la Figura 3.2 se encuentra el EDT. En dicha figura, se pueden identificar 7 paquetes principales, que dividen las tareas principales del proyecto. Estos paquetes disponen de unas dependencias entre ellos (véase el apartado 3.2.6), y de una fecha de finalización estimadas, las cuales se presentarán en el diagrama de Gantt (véase el apartado 3.2.7).

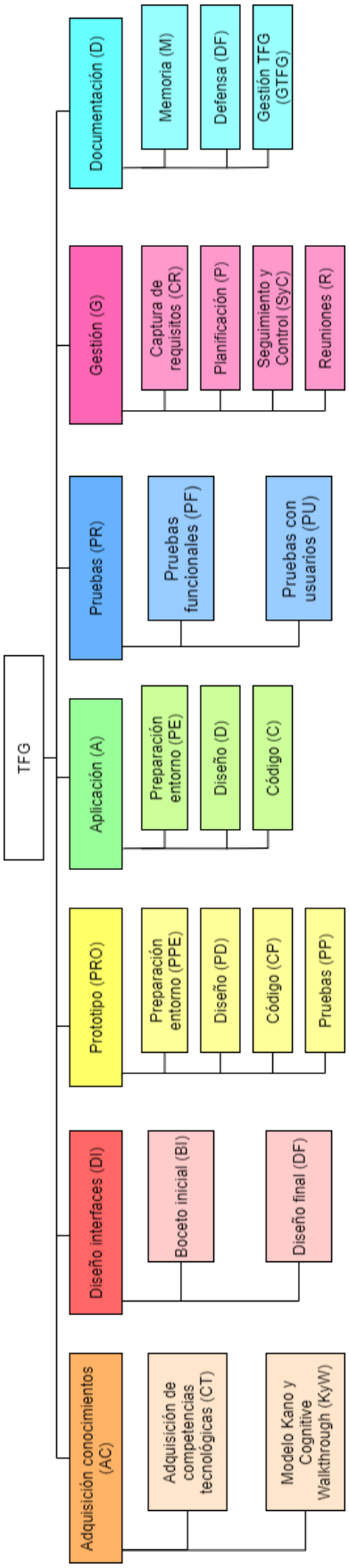

Figura 3.2: Diagrama EDT

3.2.5. Paquetes del proyecto

En este apartado se muestra la descripción de los paquetes de cada una de las fases, ya que cada paquete se divide en más paquetes, para facilitar el desarrollo del proyecto.

- **Adquisición de conocimientos (AC):**
  - *Adquisión de competencias tecnológicas (CT):* estudio y aprendizaje sobre las herramientas que se van a utilizar a la hora de desarrollar la aplicación. Este paquete se realizará a lo largo de todo el TFG, ya que no se dispone de experiencia previa con React, Electron y Node, que son las principales herramientas a utilizar, aunque puede que se añadan nuevas a lo largo del proyecto.
  - *Modelo Kano y Cognitive Walkthrough (KyW):* estudiar y recolectar información sobre el modelo que va a servir para mostrar las características relevantes al usuario que utilice la aplicación y sobre la metodología seguida a la hora de realizar el diseño de interfaces.
- **Diseño interfaces (DI):**
  - *Boceto inicial (BI):* diseño del mockup de las interfaces gráficas sobre las que se va a trabajar.
  - *Diseño final (DF):* diseño de las interfaces gráficas, es decir, el mockup exportado a React, y modificaciones de las mismas para adecuarse a la aplicación.
- **Prototipo (PRO):**
  - *Preparación entorno (PPE):* la preparación del entorno, es decir, instalación de los programas y herramientas a utilizar, habilitación de repositorio en GitHub para guardar el código del prototipo...
  - *Diseño (PD):* realizar el diseño del prototipo, tanto gráfico como a la hora de la implementación.
  - *Código (CP):* desarrollo de las funcionalidades del proyecto: web scraping, buscador de webs, generador de modelo Kano...
  - *Pruebas (PP):* al ser un prototipo solo se realizarán pruebas funcionales que comprueben el correcto funcionamiento de las funcionalidades implementadas.

- **Aplicación (A):**
  - *Preparación entorno (PE):* una vez terminado el prototipo, en caso de añadir nuevas herramientas, se volverá a preparar el entorno de trabajo.
  - *Diseño (DI):* realizar el diseño de las partes restantes de la aplicación, tanto gráfico como a la hora de la implementación.
  - *Código (C):* desarrollo de las funcionalidades restantes de la aplicación.
- **Pruebas (PR):**
  - *Pruebas funcionales (PF)):* en la fase final del proyecto, se harán pruebas funcionales sobre todas las funcionalidades que se hayan implementado en la aplicación.
  - *Pruebas con usuarios (PU):* tras realizar las pruebas funcionales, se realizarán pruebas con usuarios, de los cuales se recolectará un *feedback* de cambios posibles a realizar para facilitar el uso de la aplicación. Si dichos cambios son factibles, y el tiempo lo permite, se realizarán.
- **Gestión (G):**
  - *Captura de requisitos (CR):* al comienzo del proyecto será necesario definir los requisitos del proyecto. Estos deberán estar bien definidos para evitar futuros problemas y ambigüedades.
  - *Planificación (P):* es necesario realizar una planificación para mantener el control del proyecto.
  - *Seguimiento y Control (SyC):* seguimiento y control de todos los puntos definidos en la planificación. Esta tarea perdurará en todo el proyecto, ya que es necesario mantener el control durante su ciclo de vida.
  - *Reuniones (R):* control de las reuniones realizadas a lo largo del proyecto. Es necesario que se redacten actas, de manera adecuada, de todas las reuniones, para llevar un control de las tareas a realizar tras dichas reuniones. A menos que sea necesario (ya sea para resolver dudas, intercambio de información...), las reuniones se realizarán cuando las tareas definidas en la reunión previa se hayan terminado.
- **Documentación (D):**

- *Memoria (M):* redacción de la memoria del proyecto. Es necesario redactar la memoria durante el desarrollo de proyecto, para que todos los conceptos queden anotados con el máximo detalle. Llevando la memoria al día de esta manera, se evita el riesgo de olvidar documentar alguna parte del proyecto.
- *Defensa (DF):* preparación de la defensa del proyecto. Para ello se preparará una presentación con diapositivas, con el objetivo de presentar el trabajo realizado a lo largo del proyecto. También se realizará el póster asociado al proyecto.
- *Gestión TFG (GTFG):* gestionar los estándares de calidad del trabajo, para ello se comprobarán las rúbricas de corrección del proyecto a través de la plataforma de la UPV/EHU. También se buscará información y se comprobarán proyectos de años anteriores. Durante la fase final del proyecto, se realizarán las gestiones necesarias a través de ADDI: subir la memoria, presentación...

3.2.6. Dependencias entre tareas

Los paquetes del proyecto tienen unas dependencias entre ello, y resulta importante identificarlas, ya que puede darse el caso de que alguna de estas dependencias podría generar un problema en el desarrollo del proyecto. A continuación se procede a explicar estas dependencias, las cuales se puede apreciar en la Figura 3.3.

El proyecto comienza con el paquete **G.CR**, que indica el comienzo del proyecto. El proyecto comienza en la captura de requisitos, ya que esto fue lo que se trató en la primera reunión del mismo, por lo cual, antes que comenzar cualquier otro paquete de tareas, se necesitaba saber sobre que se iba a desarrollar el proyecto, el alcance... Tras esto, se expanden 4 caminos en paralelo.

El primer paquete **AC.KyW** es la adquisición de los conocimientos sobre los modelos y métodos a utilizar en el ámbito gráfico. Una vez terminado, da pie al **D.BI**, el cual es un boceto inicial sobre el que se va a trabajar la GUI, y **AC.CT**, que es la adquisición de las competencias necesarias para hacer uso de las herramientas. Una vez que estos dos paquetes han terminado, comienza la preparación del entorno del prototipo **PRO.PPE**, en la cual se instalan herramientas y sus dependencias. El prototipo se desarrollará de manera secuencial siguiendo los paquetes **PRO.PD**, **PRO.PC** y **PRO.PP**. Una vez que el prototipo reciba el visto bueno, se comenzará con la aplicación, la cual seguirá el mismo camino que el prototipo solo que ampliando funcionalidades, de manera secuencial: **A.PE**, **A.D** y **D.DF** van seguidos, ya que el diseño final se trabajará tras hacer el diseño

de la aplicación y por último **A.C**. Una vez terminada la aplicación, dará comienzo a las pruebas funcionales de las mismas **PR.PF** y una vez que estas terminen, se finalizará con las pruebas a usuarios **PR.PU**.

El paquete **G.R**, se va a desarrollar a lo largo de todo el proyecto, por lo cual solo depende del paquete anterior y de si mismo. Tras la captura de requisitos, da comienzo a **G.PL**, el cual una vez terminado, comienza **G.SyC**. Así mismo, tras la captura de requisitos, se comenzará **D.GTFG** el cual consistirá en analizar TFGs de otros años para realizar una correcta memoria, y en la entrega de la misma cuando corresponda. Una vez completado el paquete anterior, comenzará **D.M**, el cual consiste en el desarrollo del documento de la memoria. Por último, una vez completados los paquetes de tareas **D.GTFG**, **D.M** y **D.DF**, se finalizará el proyecto con el paquete **D.DF**, el cual consiste en la defensa del proyecto.

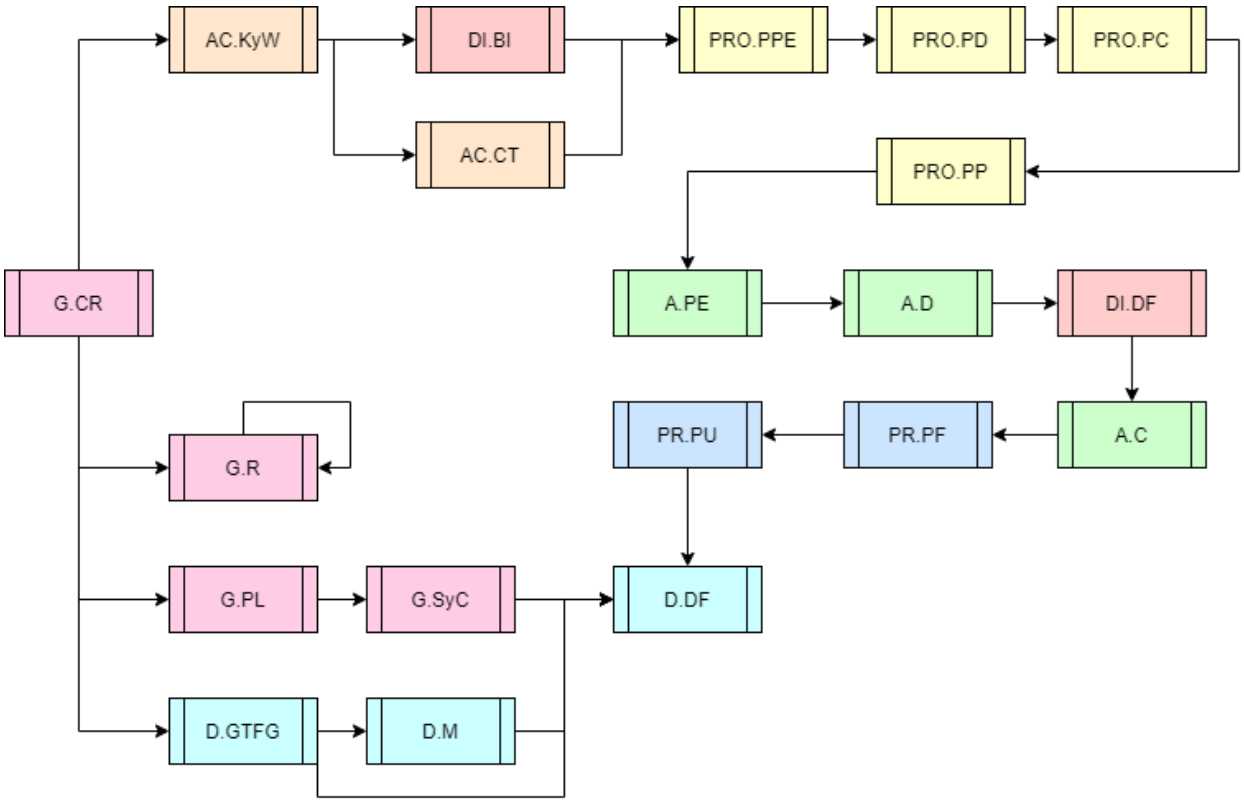

Figura 3.3: Dependencias entre tareas

3.2.7. Diagrama de Gantt

En este apartado se presenta el diagrama de Gantt, que servirá para tener una idea general de la duración de cada tarea del proyecto. El proyecto dio comienzo el día **8 de septiembre de 2021**, con una reunión entre el alumno Unai y el director Oscar. En esta reunión,

se presentó el proyecto, se realizó la captura de requisitos y se concluyó el alcance del proyecto.

En el diagrama se presentan todas las fases y paquetes presentados en el anterior apartado. También se han añadido los hitos más importantes del proyecto.

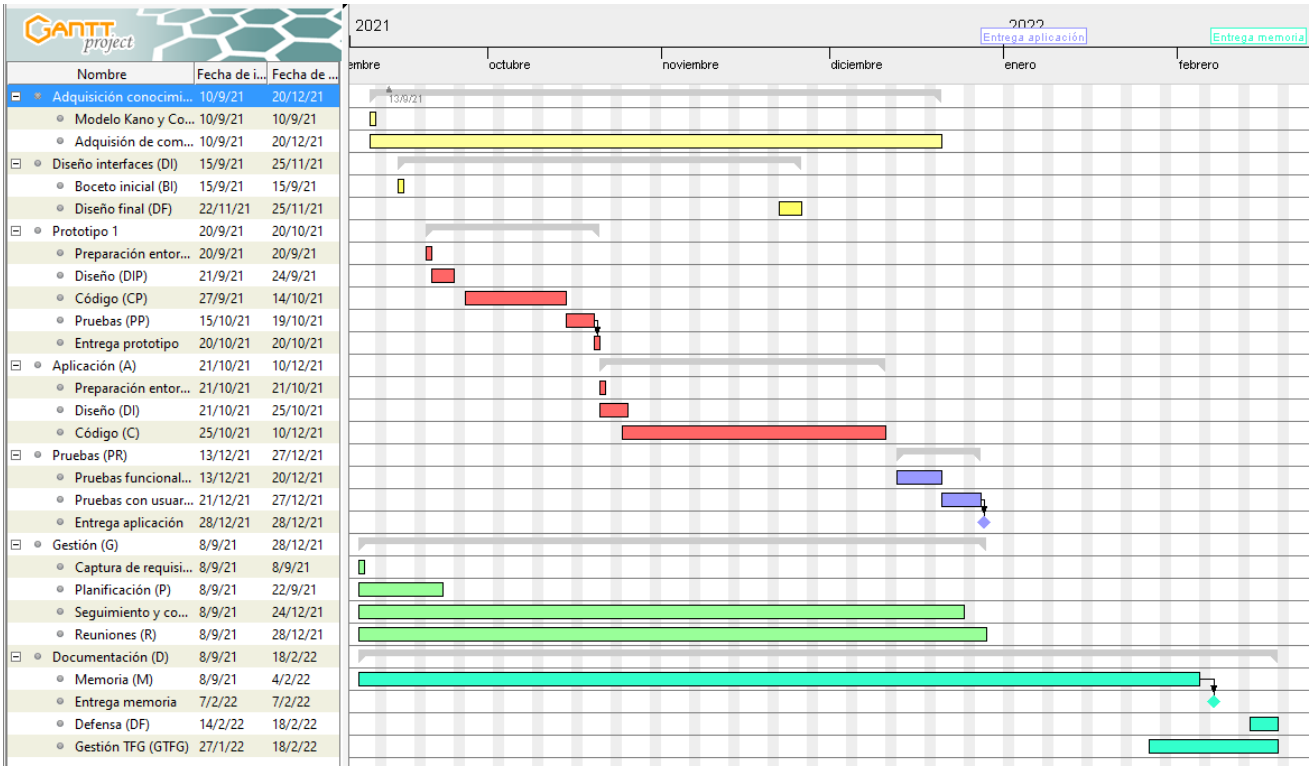

Figura 3.4: Diagrama de Gantt del proyecto

3.3. Gestión del Tiempo

Teniendo en cuenta el alcance definido, se ha realizado una estimación previa para poder controlar el tiempo, con el objetivo de cumplir con las fechas e hitos establecidos en el diagrama de Gantt 3.4.

3.3.1. Tiempo estimado a cada tarea

BLOQUE DE TRABAJO	TAREA	ESTIMACIÓN
DISEÑO INTERFACES (DI)	Boceto inicial (BI)	4h
	Diseño final (DF)	16h
	SUBTOTAL	**20h**
PROTOTIPO (PRO)	Preparación entorno (PPE)	4h
	Diseño (PD)	10h
	Código (PC)	40h
	Pruebas (PP)	6h
	SUBTOTAL	**60h**
APLICACIÓN (A)	Preparación entorno (PE)	4h
	Diseño (D)	10h
	Código (C)	70h
	SUBTOTAL	**84h**
PRUEBAS (PR)	Pruebas funcionales (PF)	30h
	Pruebas con usuarios (PU)	10h
	SUBTOTAL	**40h**
GESTIÓN (G)	Captura de requisitos (CR)	4h
	Planificación (P)	20h
	Seguimiento y Control (SyC)	6h
	Reuniones (R)	14h
	SUBTOTAL	**44h**
DOCUMENTACIÓN (D)	Memoria (M)	50h
	Defensa (DF)	6h
	Gestión TFG (GTFG)	4h
	SUBTOTAL	**60h**
ADQUISICIÓN CONOCIMIENTOS (AC)	Adquisión de competencias tecnologías (CT)	20h
	Modelo Kano y Cognitive Walkthrough (KyW)	4h
	SUBTOTAL	**24h**
HORAS TOTALES ESTIMADAS		**332h**

Tabla 3.1: Horas estimadas para cada tarea del proyecto

3.4. Gestión de Riesgos

R1-Captura de requisitos:

- *Descripción:* la especificación de los requisitos es de suma importancia, ya que con estos se define el alcance del proyecto, y toda la planificación se realiza sobre dicho alcance. En caso de que algún requisito esté mal definido, el alcance del proyecto cambiará, por lo cual también lo hará la planificación.
- *Probabilidad*: baja.
- *Impacto*: muy alto.
- *Prevención*: es importante que se definan de manera correcta y que estén verificados por el cliente.

- *Plan de acción*: los requisitos se acotaron en la primera reunión. En caso de que estos requisitos se modifiquen, se añadan o se eliminen requisitos, será necesario documentarlo.

R2-Compatibilidad con el curso académico:

- *Descripción:* el proyecto da comienzo con el inicio del primer cuatrimestre del quinto curso del grado. El alumno ya ha completado el curso académico, a falta del TFG y de la convalidación de las prácticas realizadas en verano.

- *Probabilidad*: muy baja.

- *Impacto*: alto.

- *Prevención*: realizar la planificación teniendo en cuenta los problemas que puedan surgir a la hora de convalidar las prácticas.

- *Plan de acción*: el alumno hará la entrega de los documentos necesarios para convalidar las prácticas a principios de Octubre de este mismo año, y de esta manera solventar cualquier problema que pudiera suceder lo antes posible.

R3-Uso de tecnologías desconocidas:

- *Descripción*: al hacer uso de nuevas tecnologías que no se han usado con anterioridad, es necesario tener en cuenta que la adaptación al uso de estas herramientas puede modificar las estimaciones del proyecto.

- *Probabilidad*: media.

- *Impacto*: medio.

- *Prevención*: buscar información y realizar un estudio previo de las herramientas a utilizar. En caso de necesitar ayuda y si es posible, solicitarla.

- *Plan de acción*: comunicar de la situación al grupo para obtener ayuda. Si supone un gran problema, realizar una nueva planificación.

R4-Problemas personales:

- *Descripción*: siempre en necesario tener en cuenta los riesgos que puede tener una persona, ya sean enfermedades, accidentes... Además hay que añadir el problema actual con el COVID-19, aunque la situación ha mejorado considerablemente y el alumno esta vacunado de pauta completa, no se asegura al completo que el alumno no vaya a contagiarse con el virus, o de una nueva cepa.
- *Probabilidad*: baja.
- *Impacto*: alto.
- *Prevención*: cuidado de salud.
- *Plan de acción*: se asumen los riesgos de problemas personales. En el caso de padecer alguno, será necesario comunicarlo directamente al director del proyecto. En caso de que estos problemas sean graves, se tendrá en cuenta posponer la entrega del proyecto.

R5-Pérdida del sistema de información:

- *Descripción*: el proyecto se desarrollara en los dispositivos del alumno, por ello será necesario que la información esté en todo momento sincronizada. Además, se puede dar el caso de que esta información se pierda, por problemas tecnológicos o errores humanos.
- *Probabilidad*: muy baja.
- *Impacto*: muy alto.
- *Prevención*: realizar copias de seguridad del sistema de información en diferentes plataformas. En este caso: copia en local del código, documentos y archivos (fotos, diagramas...) asociados al proyecto; GitHub para el código de la aplicación; Overleaf para los documentos escritos en Latex; y Drive, donde se hace otra copia de seguridad todo lo previamente mencionado.
- *Plan de acción*: comunicar lo ocurrido al director del proyecto para realizar un plan de recuperación. En caso de que la pérdida de información sea con relación al código, se podrá recuperar una versión previa ya sea de GitHub o Drive. En caso de perder documentación, se podrán recuperar las decisiones tomadas... mediante las actas del proyecto, y el control de versiones de Drive y Overleaf.

R6-Planificación incorrecta:

- *Descripción*: puede que el proyecto se dificulte a lo largo del desarrollo, llegando al caso de que algunas tareas se retrasen, y esto conlleve al retraso de los diferentes paquetes de trabajo definidos en el EDT.
- *Probabilidad*: media.
- *Impacto*: alto.
- *Prevención*: realizar una planificación perfecta es complicado, por ello, para conseguir la mejor planificación posible, se deberá invertir el tiempo suficiente antes de comenzar el proyecto realizando dicha planificación.
- *Plan de acción*: replanificar las tareas que se vean afectadas por una mala previa planificación, intentando que el impacto de dichas modificaciones sea el menor posible en el proyecto.

3.5. Gestión de la Calidad

En este apartado se describe la calidad del proyecto y sus entregables (actas y documentos). Para esto se han determinado los siguientes criterios de calidad junto a sus métodos de calificación:

- **Manejo de los datos:** el programa deberá de gestionar y almacenar un gran número de datos, ya que el programa deberá extraer muchas *reviews* de la Workshop de Google Chrome.
  - *Mala:* se pierden datos en la ejecución del programa.
  - *Buena:* se conservan los datos en la ejecución del programa.
  - *Excelente:* se conservan los datos en la ejecución del programa, y además se dispone de mecanismos de control (como puede ser realizar una copia de seguridad temporal durante el proceso de análisis).
- **Usabilidad:** la interfaz deberá ser simple y fácil de utilizar. Deberá disponer de menús intuitivos y textos explicativos mediante los cuales los usuarios puedan realizar las acciones que deseen sin complejidad. Para asegurar la calidad de este apartado, se utilizará el método *Cognitive walkthrough*.

  - *Mala:* un usuario es incapaz de usar el programa incluso siguiendo los textos que sirven de guía durante la ejecución del programa.
  - *Buena:* un usuario puede entender la herramienta haciendo uso de los textos que sirven de guía durante la ejecución del programa.
  - *Excelente:* un usuario puede entender la herramienta sin necesidad de leer los textos de guía.

- **Entregables:** los entregables (las actas y documentos) deberán cumplir con los estándares básicos: fecha, objetivos, resultados...
  - *Mala:* las actas carecen de la información completa de la reunión. Los documentos carecen de estructura y formato.
  - *Buena:* las actas plasmas lo acordado y hablado en las reuniones. Los documentos sigues los estándares básicos de calidad.
  - *Excelente:* las actas recogen lo acordado en las reuniones y a parte sirven para tener una trazabilidad del estado del proyecto. Los documentos contienen componentes añadidos (imágenes por ejemplo) que aportan más calidad a los mismos.

3.6. Gestión de Comunicaciones e Información

3.6.1. Sistema de información

Es muy importante mantener la información segura y disponible en todo momento. Para ello se ha creado un sistema de información, mediante el cual se pueda evitar o minimizar el R9. El sistema de información es distribuido en dos dispositivos locales: ordenador de sobremesa y portátil, y en la nube:

- **GitHub:**[2] se utilizará para mantener una copia de seguridad del código mediante un repositorio privado. De esta manera se tendrá acceso personal a las diferentes versiones de la aplicación durante el desarrollo. Para facilitar el uso de Git[3], se utilizará el programa GitKraken[4], el cual sirve para tener un GUI a la hora de trabajar con Git.

[2] GitHub: https://github.com/.
[3] Git: https://git-scm.com/.
[4] GitKraken: https://www.gitkraken.com/.

- **Google Drive:**[5] se utilizará para almacenar una copia de seguridad del código y de los diferentes documentos relacionados a la memoria (diagramas, capturas...).
- **Overleaf:**[6] se utilizará para almacenar los ficheros de LaTeX[7], con sus diferentes versiones y modificaciones.

3.6.2. Sistema de comunicación

La comunicación se irá realizando a lo largo del desarrollo del proyecto, según vaya siendo necesario, para esto se dispondrán de diferentes tecnologías:

Herramientas de comunicación

- **Correo electrónico:** principal herramienta de comunicación. Se utiliza para resolver dudas, concretar fechas de reuniones y para el seguimiento y control del proyecto.
- **Teléfono:** herramienta secundaria en caso de que el correo electrónico no funcione.
- **Reuniones telemáticas:** debido a que el alumno vive en otra ciudad, y por conveniencia y ahorrar tiempo, se procurarán realizar de manera telemática la gran parte de las reuniones. Para esto se utilizará el programa Webex[8]. Para poder tener un control de las reuniones, estas serán resumidas en sus correspondientes Actas, que serán redactadas por el alumno (Anexo C).
- **Reuniones presenciales:** en el caso de ser necesario, se realizarán reuniones presenciales. Estas se realizarán para presentar funcionalidades finalizadas, las pruebas de usuarios o el cierre del proyecto.

Reuniones y actas

Todas las reuniones que se realicen se verán reflejadas en las actas. Todas estas actas tendrán una misma plantilla (Véase C). En estas actas, se definen muchos puntos de gran ayuda: Orden del día, tareas asignadas...

[5]Google Drive: https://drive.google.com/drive/u/0/my-drive.
[6]Overleaf: https://www.overleaf.com/.
[7]LaTeX: https://www.latex-project.org/.
[8]Webex: https://www.webex.com/es/video-conferencing.html.

4. CAPÍTULO

Herramientas utilizadas en el desarrollo

Para el desarrollo del TFG se han utilizado distintas herramientas, procurando que las mismas sean utilizadas actualmente, útiles en el futuro laboral y sirvan para aportar nuevo conocimiento al alumno, aunque por esto último, el alumno debía familiarizarse con dichas herramientas (Riesgo a tener en cuenta en el Capítulo 3). Para la elección de las herramientas, el grupo de investigación **Onekin** realizó un análisis previo de las mismas, por lo cual antes de empezar este proyecto ya se tenía una idea de las herramientas a utilizar, las cuales se van a detallar más adelante en este mismo apartado.

4.1. Diseño aplicación

4.1.1. Mockplus Classic

Mockplus Classic[1] es una herramienta utilizada para realizar prototipos. Las características de esta herramienta son las siguientes, entre otras: rapidez de diseño (disponibilidad de un gran número de librerías y componentes), interacción (opción de interacción entre diferentes componentes) y testeo (vistas en diferentes dispositivos con opción a exportarlo como HTML); y colaboración en tiempo real.

Mockplus se ha utilizado en este proyecto para realizar los prototipos previos a la implementación de la aplicación, de esta manera, antes de comenzar dicha implementación, se disponía de una base visual y de funcionalidades de las que partir.

[1]Mockplus Classic: https://www.mockplus.com/download/mockplus-classic.

4.2. Desarrollo aplicación

4.2.1. Node.js

Node.js[2] es un entorno de ejecución multiplataforma de código abierto, que como su nombre indica, esta basado en el lenguaje de programación **JavaScript**, el cual es uno de los lenguajes de programación más utilizados hoy en día en el desarrollo de aplicaciones y servidores (el objetivo de este proyecto). Node sirve para desarrollar tanto front-end como back-end, y en este proyecto se ha utilizado para ambos: en el front-end para el desarrollo de de la aplicación junto a Electron y React; y en el back-end para el desarrollo de los servicios. Además de esto, Node es fácil de combinar con diferentes frameworks, entre ellos Express y Puppeteer, a parte, el grupo de investigación **Onekin** ya tenía experiencia con esta herramienta.

4.2.2. Electron

Electron[3] es un framework de código abierto, muy utilizado hoy en día (Discord, WhatsApp y Visual Studio Code entre otras aplicaciones) mediante el cual permite desarrollar aplicaciones de escritorio utilizando componentes del lado del cliente y del servidor. A parte, Electron permite realizar aplicaciones portátiles y muy fácilmente adaptables a distintos sistemas operativos, e incluso se puede exportar como una aplicación de Android/IOS.

[2]Node.js: https://nodejs.org/es/.
[3]Electron: https://www.electronjs.org/.

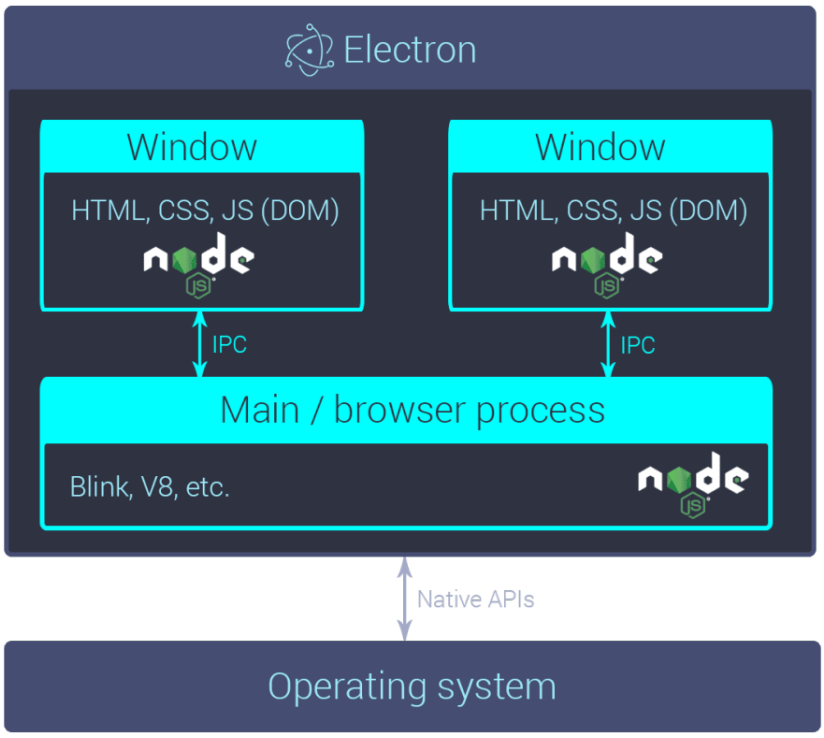

Figura 4.1: Funcionamiento Electron [Sekhon, 2020].

4.2.3. React

React[4] es una biblioteca utilizada en Electron, mediante la cual se pueden realizar GUIs interactivas, y con la que se facilita el desarrollo de aplicaciones en una sola página, siguiendo los principios de *Single-Page Application* (SPA). De esta manera, se definen distintos componentes, y estos se irán actualizando y cambiando dependiendo del estado en el que se encuentre la aplicación, de esta manera se evita la carga continua de las páginas desde el servidor.

4.3. Servicios

4.3.1. Docker

Docker[5] es un sistema operativo para proporcionar contenedores ligeros que ejecutan procesos de manera aislada, funciona de manera similar a una máquina virtual ya que virtualiza el hardware del servidor. La ventaja principal de utilizar Docker, es que a cada

[4]React: `https://es.reactjs.org/`.
[5]Docker: `https://www.docker.com/`.

contenedor solo se le suministra los recursos necesarios para que funcione, sin necesidad de una máquina virtual completa. Para esto, hace uso de un fichero llamado *Dockerfile*, en el cual se definen los comandos a ejecutar (como podría ser instalar paquetes, ejecutar alguna aplicación o programa...) para el despliegue de dicho contenedor. En este proyecto, se ha utilizado para el despliegue los servicios encargados de realizar el web-scrapping de la información de las extensiones de Google Chrome y de la extracción de las palabras clave de los comentarios.

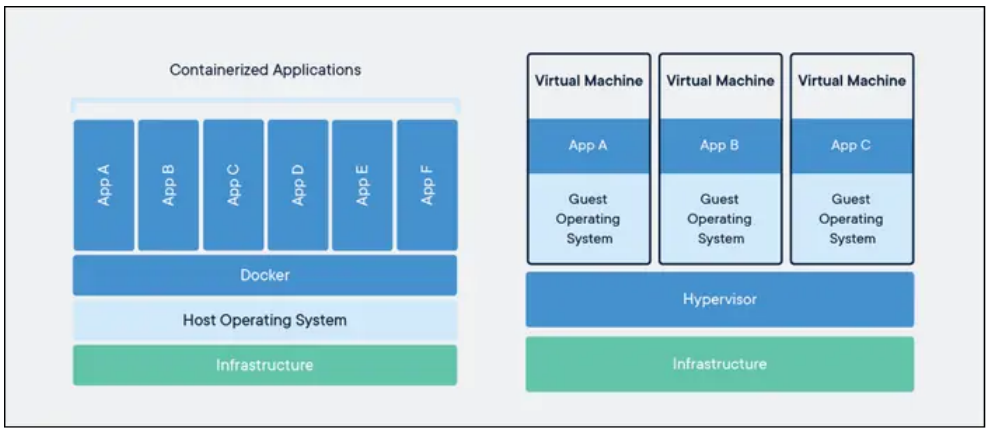

Figura 4.2: Comparación entre Docker y máquinas virtuales [Documentation,].

4.3.2. Docker compose

Docker compose[6] es una herramienta que funciona sobre Docker, y permite agrupar varios contenedores y lanzarlos simultáneamente mediante un único comando. Para realizar esta agrupación, se utiliza el *Composer file*, el cual es un fichero parecido al *Dockerfile*, mediante el cual se escogen cuales de los *Docker* definidos se van a ejecutar, en que puertos, con que parámetros... Gracias a esto, los servicios puede ser fácilmente escalados y portados a otros dispositivos, de esta manera es mucho más sencillo desplegarlos en un servidor, y de esta manera, los servicios desarrollados en este proyecto, pueden ser reutilizados en más aplicaciones y/o proyectos a parte de la aplicación implementada en este TFG.

[6]Docker: https://docs.docker.com/compose/.

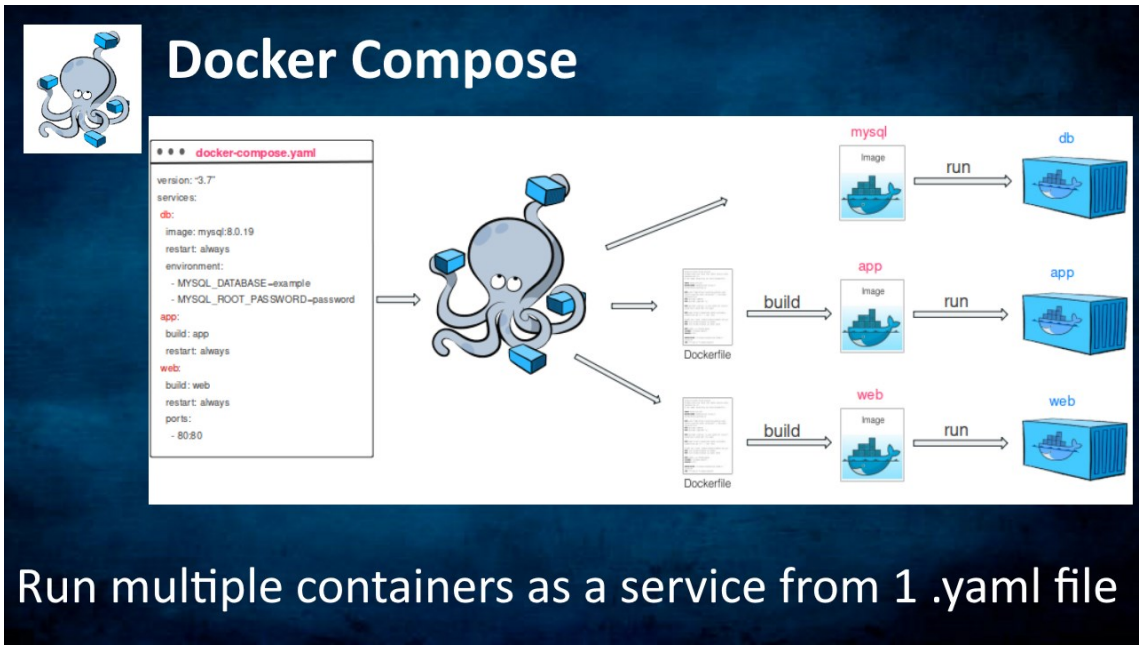

Figura 4.3: Ejemplo de Docker compose [Azure, 2020].

4.3.3. Express

Express [7] es el framework más utilizado de Node para la creación de aplicaciones web, mediante el cual se pueden crear servidores web de manera rápida y sencilla. Juntando Express y Puppeteer, es como se ha creado el servicio web que tiene las tareas de realizar web-scrapping. Al ser un framework de Node, se utiliza JavaScript, por lo cual habiendo aprendido a utiliza Node, es realmente sencillo utilizar Express para crear los servicios necesarios

4.3.4. Puppeteer

Puppeteer[8] es una librería de Node.js que proporciona una API para controlar Crome/-Chromium de manera autónoma. Se utiliza para simular las acciones que haría una persona en una página web, y en este proyecto se ha utilizado junto a Express para crear un servicio el cual se encarga de realizar el web-scrapping de la información necesaria de las extensiones.

[7] Express: https://expressjs.com/es/.
[8] Puppeteer : https://github.com/puppeteer/puppeteer.

5. CAPÍTULO

Diseño

En este capítulo se hablará de las decisiones tomadas a la hora de realizar el diseño de la aplicación: tanto por parte del programa como de los servicios que son necesarios para su funcionamiento.

5.1. Cognitive walkthrough

Cognitive Walkthrough o Recorrido Cognitivo[1] [Modroño, 2021] es un método de inspección de la usabilidad utilizado para probar un producto en una situación de uso simulado.

Para realizar un *Cognitive Walkthrough*, es necesario lo siguiente:

- Definir qué usuarios van a utilizar el producto.
- Definir qué tareas son las más apropiadas para el *Walkthrough*.
- Proveer una representación de las interfaces.
- Por cada tarea, realizar las siguientes preguntas.
    - ¿El usuario intentará conseguir el propósito?
    - ¿Es visible la acción?
    - ¿El usuario asociará la acción al resultado esperado?
    - ¿El usuario entenderá el feedback?

[1] Cognitive Walkthrough: https://www.usabilitybok.org/cognitive-walkthrough.

- Anotar el resultado de las anteriores preguntas, y realizar los cambios que sean necesarios.

De esta manera, se consigue obtener una idea sobre la interfaz desarrollada y se puede saber qué es necesario cambiar sin necesidad de usuarios de testeo. Una vez hecho el prototipo mediante Mockuplus, se procedió a realizar el *Walkthrough* de las diferentes páginas de la aplicación.

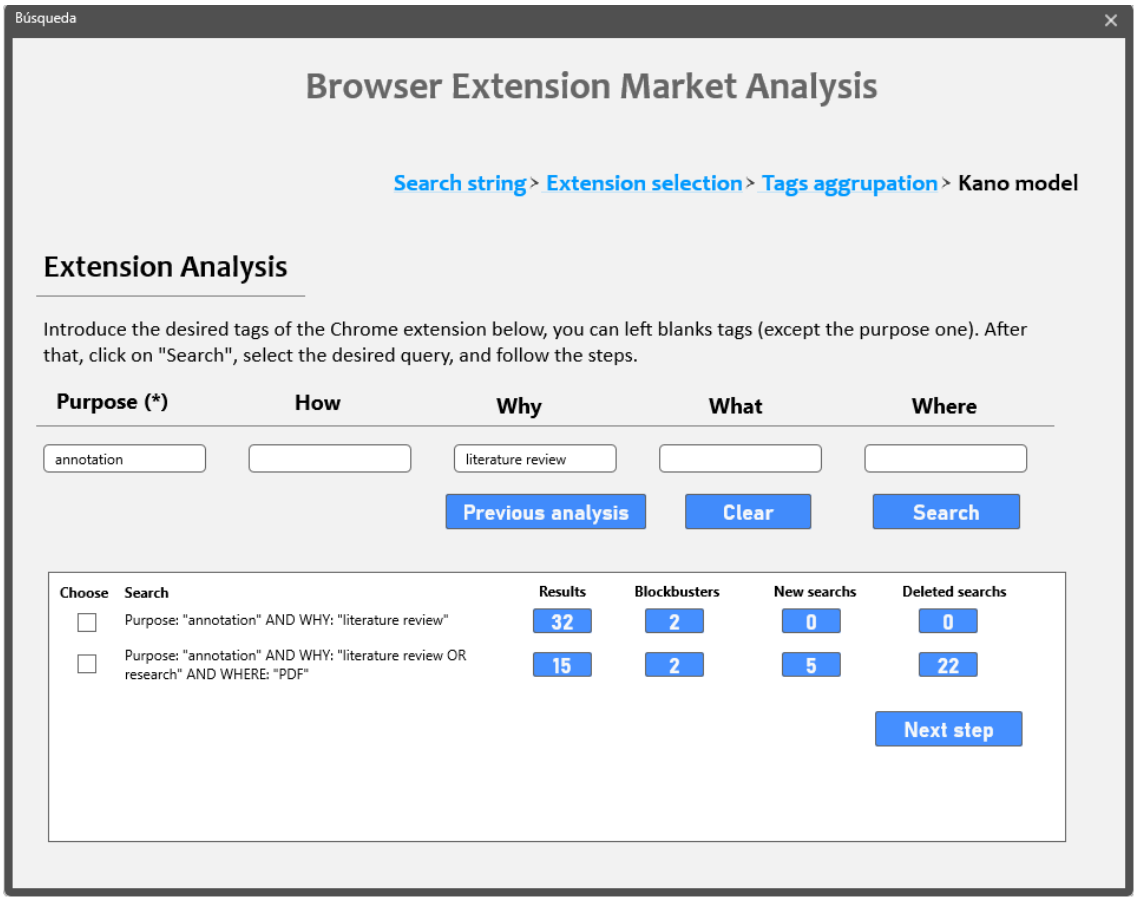

Figura 5.1: Mockup página 1 de la aplicación.

En esta página se evalúa la tarea de rellenar los ïnputz seleccionar la búsqueda deseada.

- ¿El usuario intentará conseguir el propósito? Si, la acción no requiere grandes conocimientos para llevarla acabo.
- ¿Es visible la acción? Si, hay un texto avisando y dando instrucciones de como realizar la acción.

- ¿El usuario asociará la acción al resultado esperado? Si, el menú cambiará al siguiente paso.
- ¿El usuario entenderá el feedback? Si, el menú cambiará al siguiente paso y la parte de arriba a la derecha (llamado breadcrumb) también.

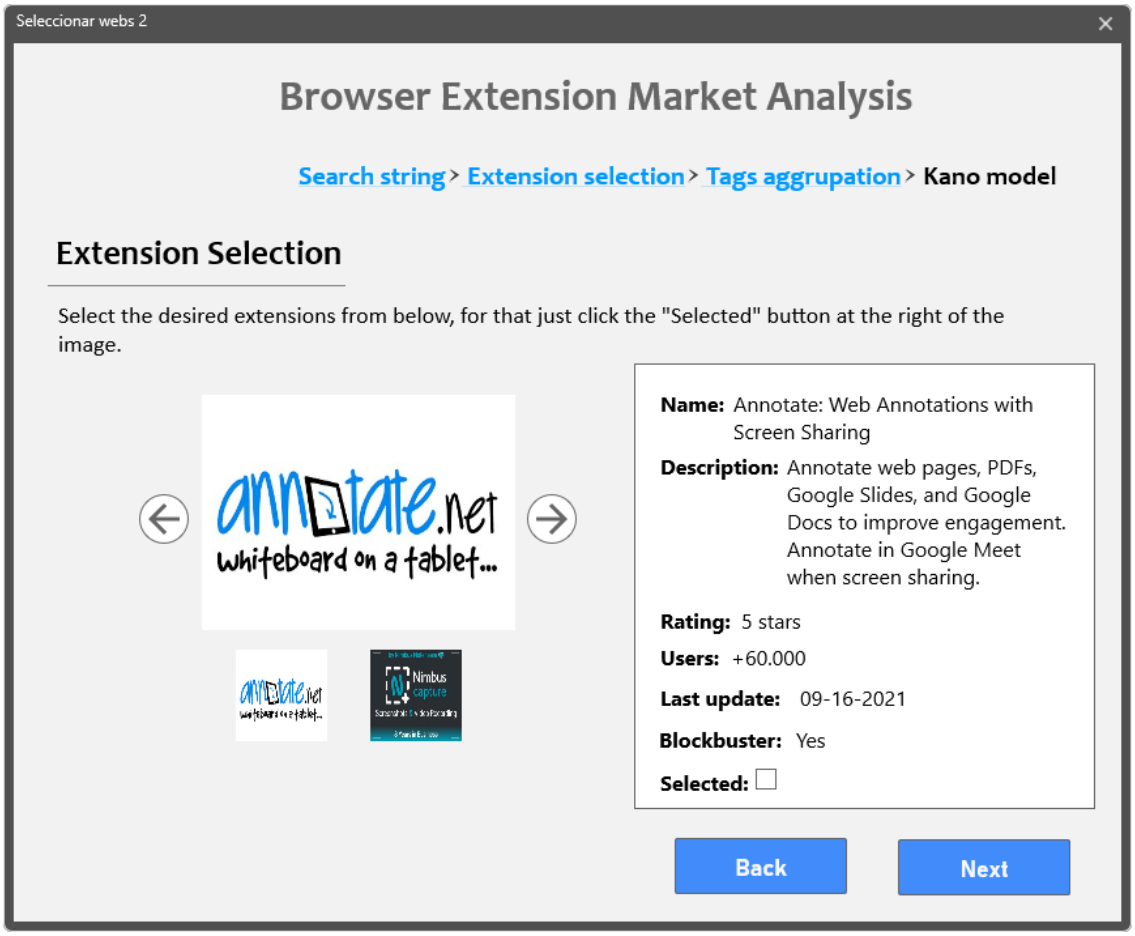

Figura 5.2: Mockup página 2 de la aplicación.

En esta página se evalúa la tarea de seleccionar las extensiones deseadas.

- ¿El usuario intentará conseguir el propósito? Si, la acción no requiere grandes conocimientos para llevarla acabo, consiste en marcar la casilla de ”Selected.[en] las extensiones deseadas.
- ¿Es visible la acción? Si, hay un texto dando instrucciones de como realizar la acción.

- ¿El usuario asociará la acción al resultado esperado? Si, el menú cambiará al siguiente paso.
- ¿El usuario entenderá el feedback? Si, una vez terminada la tarea, el menú y el breadcrumb cambiarán.

En esta página se evalúa la tarea de agrupar los *tags* (también conocido como *keywords*).

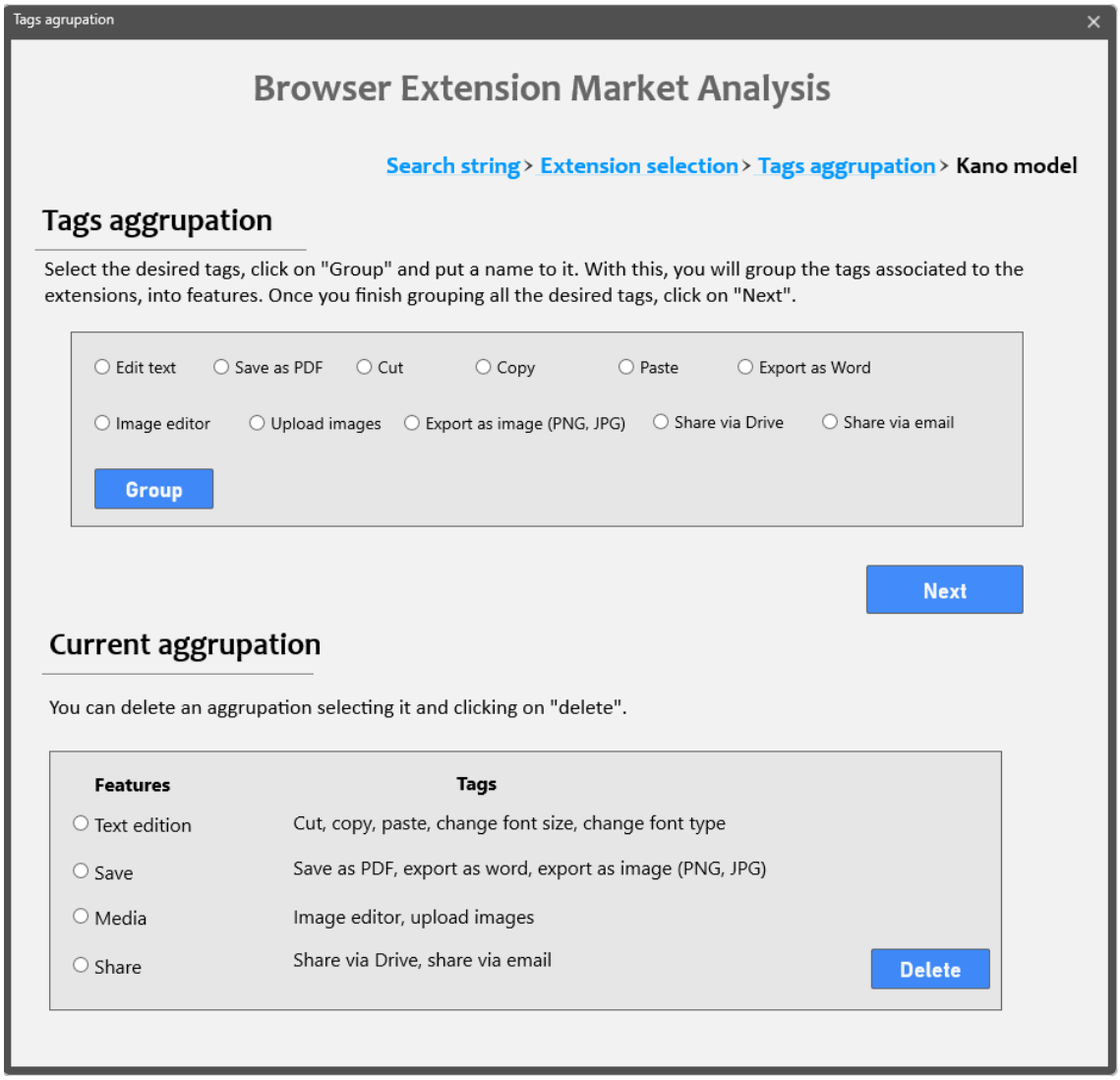

Figura 5.3: Mockup página 3 de la aplicación.

- ¿El usuario intentará conseguir el propósito? Si, la tarea es sencilla y consiste en seleccionar las *tags* deseadas y agruparlas.
- ¿Es visible la acción? Si, hay un texto dando instrucciones de como realizar la acción.

- ¿El usuario asociará la acción al resultado esperado? Si, debido a que al crear las agrupaciones, estas serán visibles en un apartado en la zona inferior de la página.
- ¿El usuario entenderá el *feedback*? Si, una vez terminada la tarea, el menú y el *breadcrumb* cambiarán.

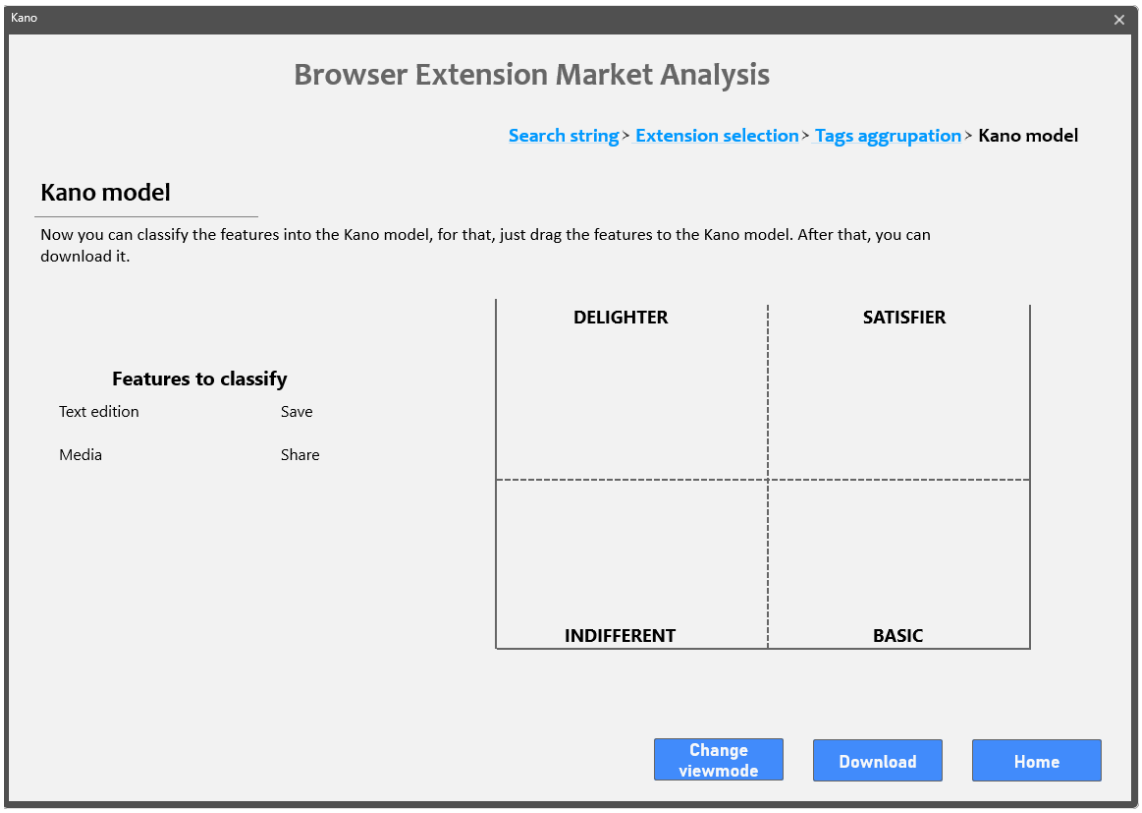

Figura 5.4: Mockup página 4 de la aplicación.

En esta página hay dos tareas a realizar: la clasificación de las funcionalidades y la descarga del archivo.

- Clasificación de las funcionalidades:
  - ¿El usuario intentará conseguir el propósito? Si, solo es necesario arrastrar las funcionalidades al modelo Kano.
  - ¿Es visible la acción? Si, hay un texto avisando y dando instrucciones de como realizar la acción.
  - ¿El usuario asociará la acción al resultado esperado? Si, la acción es parte del resultado.

- ¿El usuario entenderá el *feedback*? Si, el *feedback* es la misma acción de mover las funcionalidades.

- Descarga del archivo:

  - ¿El usuario intentará conseguir el propósito? Si, solo es necesario darle clic a un botón.
  - ¿Es visible la acción? Si, hay un texto avisando y dando instrucciones de como realizar la acción.
  - ¿El usuario asociará la acción al resultado esperado? Si, se descargará un archivo.
  - ¿El usuario entenderá el *feedback*? Si, saldrá una ventana avisando de la descarga del archivo.

Como se puede observar mediante el *Walkthrough*, el prototipo creado mediante Mockplus es bastante consistente, y se pueden sacar las siguientes conclusiones: es recomendable poner textos qué describan como realizar las tareas, para que los usuarios nuevos sepan como utilizar la aplicación; es importante darle *feedback* al usuario así sabrá si esta realizando la acción correctamente; y las tareas deben ser simples de realizar, de esta manera el usuario podrá llevarlas a cabo sin esfuerzo. Por ello, el diseño de la aplicación estará basado en gran parte en este prototipo, aunque puede recibir algún cambio.

5.2. Caso de estudio

En cuanto al diseño de la aplicación, se comprobó que el prototipo realizado mediante *Cognitive walkthrough* era sencillo e intuitivo de utilizar, por lo cual, las interfaces gráficas se implementaron de la siguiente manera, se mostrarán los resultados mediante un ejemplo de uso de la aplicación. En este ejemplo se va a trabajar sobre el *purpose* de *sustainability*.

5.2.1. Search string

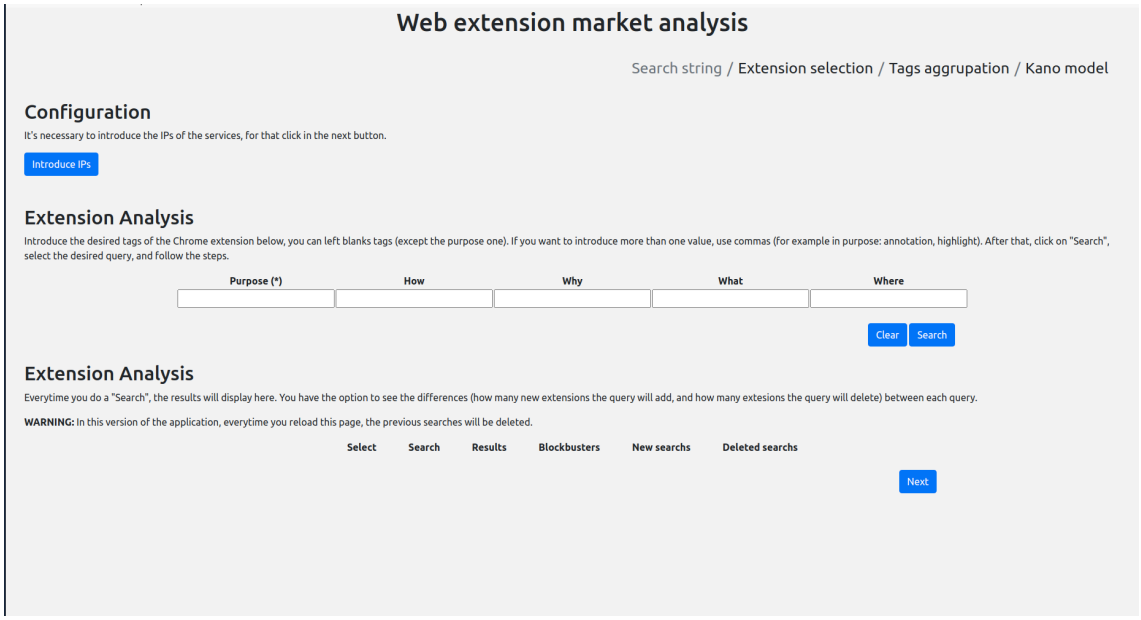

Figura 5.5: Página *search string*.

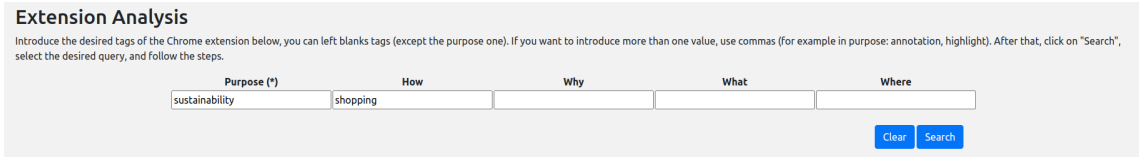

Figura 5.6: Parte superior de la página *search string*.

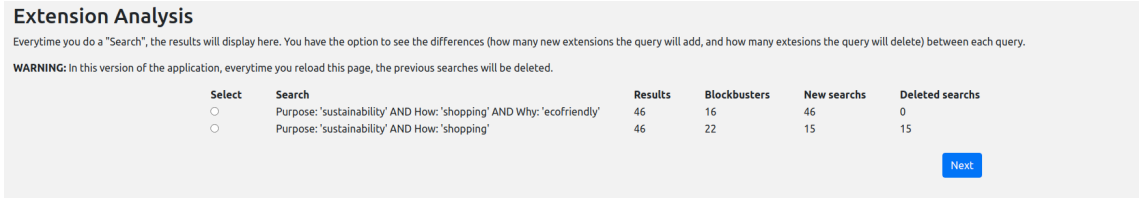

Figura 5.7: Parte inferior de la página *search string*.

La primera página dispone de los siguientes apartados:

- **Configuration:** es donde se introducen las IPs de los servicios.
- **Extension Analysis superior:** es donde se introduce la búsqueda deseada.

- **Extension Analysis inferior:** es donde se muestra el resultado de la búsqueda y donde el usuario selecciona la deseada.

5.2.2. Extension selection

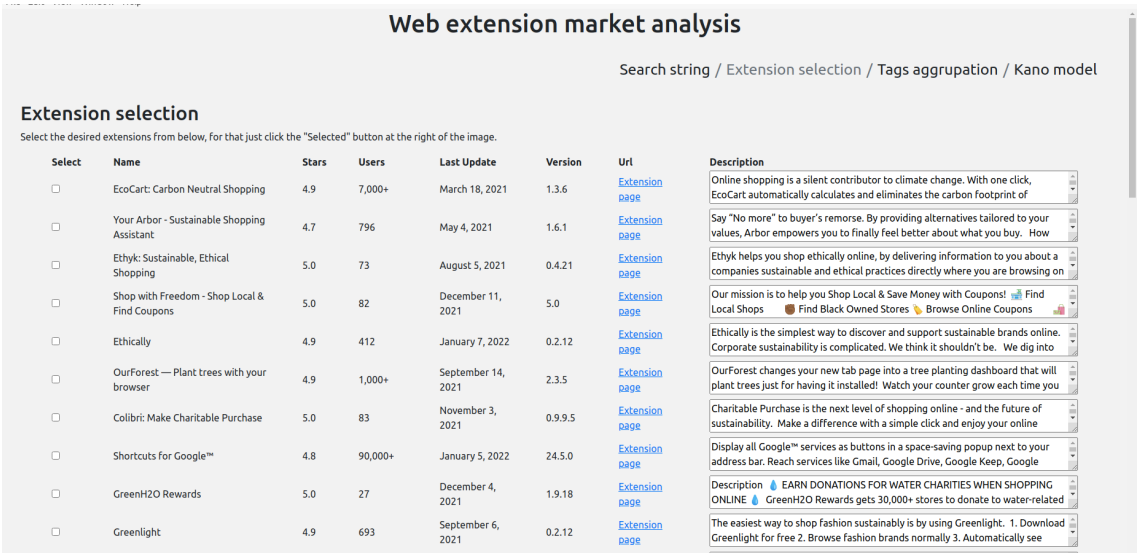

Figura 5.8: Parte superior de la página de *extension selection*.

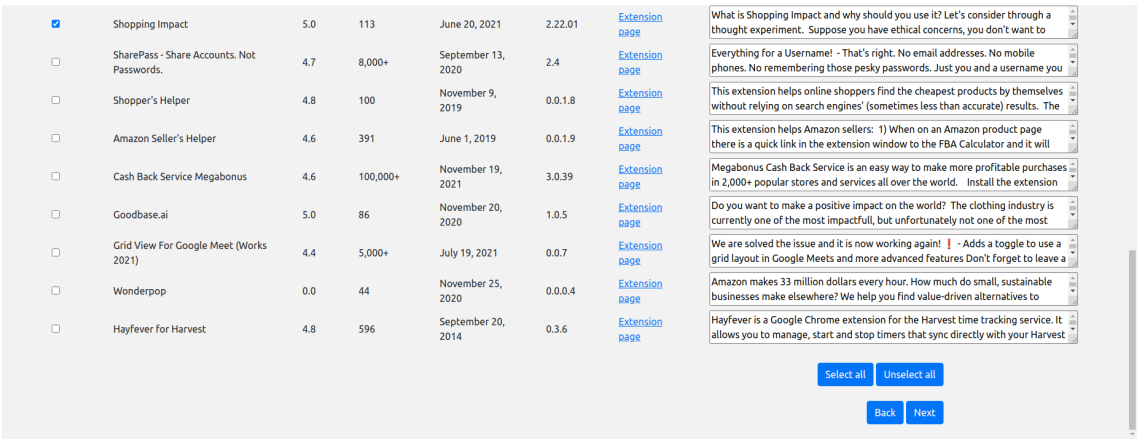

Figura 5.9: Parte inferior de la página *extension selection*.

La segunda página sirve para mostrar información sobre las extensiones comprendidas en la búsqueda y que el usuario pueda elegir las extensiones deseadas. En este ejemplo se ha escogido una extensión.

5.2.3. Tags aggrupation

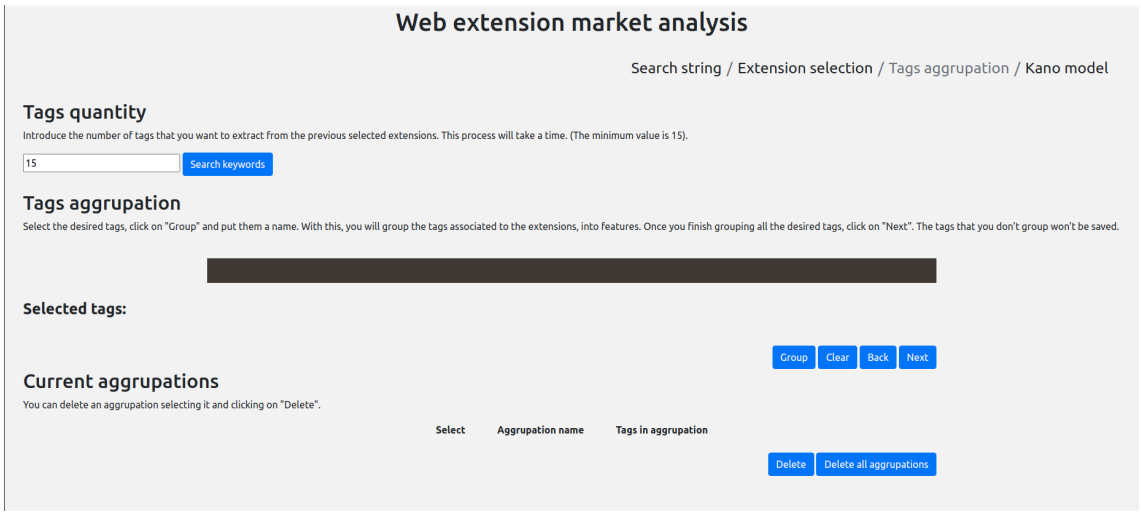

Figura 5.10: Página de *tags aggrupation*.

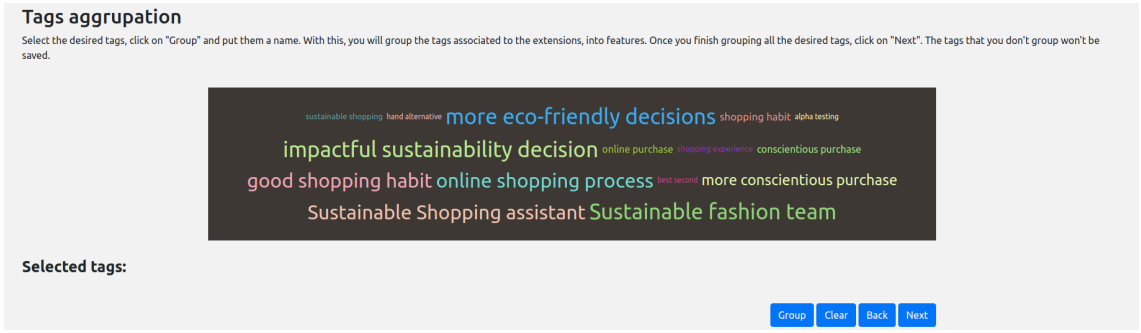

Figura 5.11: Página de *tags aggrupation* tras extraer los tags.

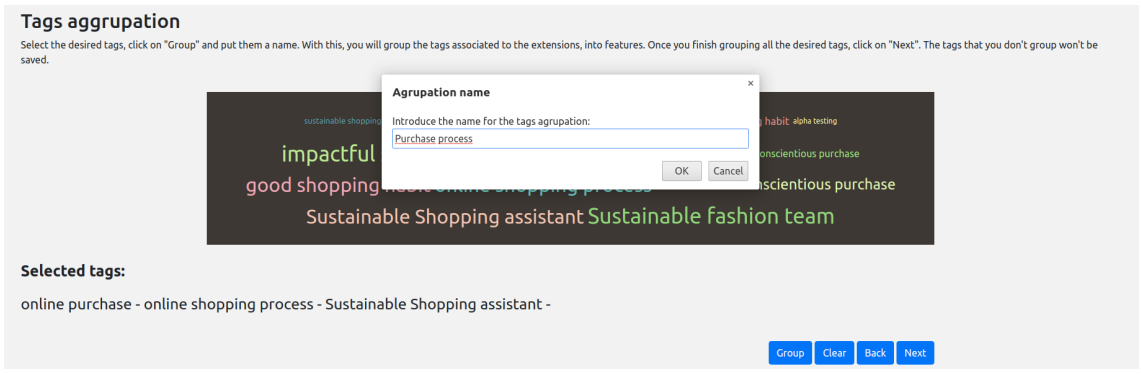

Figura 5.12: Página de *tags aggrupation* creando una agrupación.

Figura 5.13: Agrupaciones creados en la página *tags aggrupation*.

La tercera página tiene los siguientes apartados:

- **Tags quantity:** para escoger la cantidad de palabras clave a mostrar.
- **Tags aggrupation:** mediante esto, el usuario puede crear las agrupaciones de las palabras clave.
- **Current aggrupations:** se muestran las agrupaciones actuales, y en caso de desearlo, el usuario podrá eliminarlas.

5.2.4. Kano model

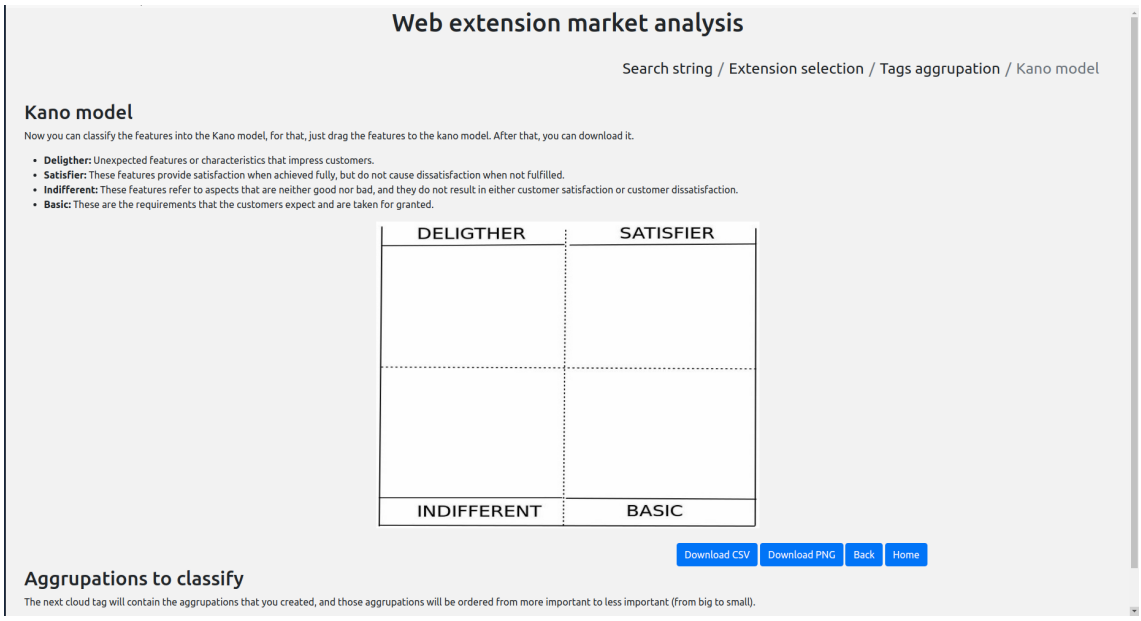

Figura 5.14: Página de *Kano model*.

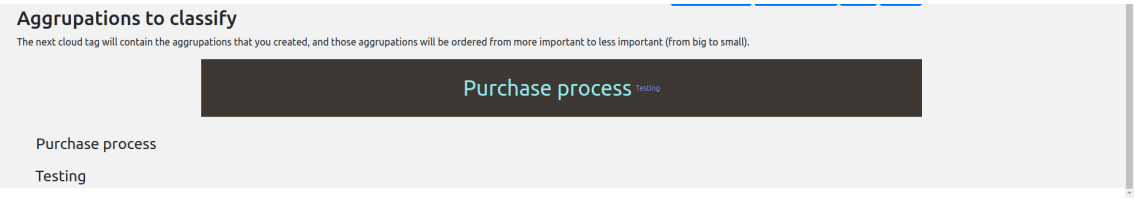

Figura 5.15: Total de agrupaciones creadas previamente.

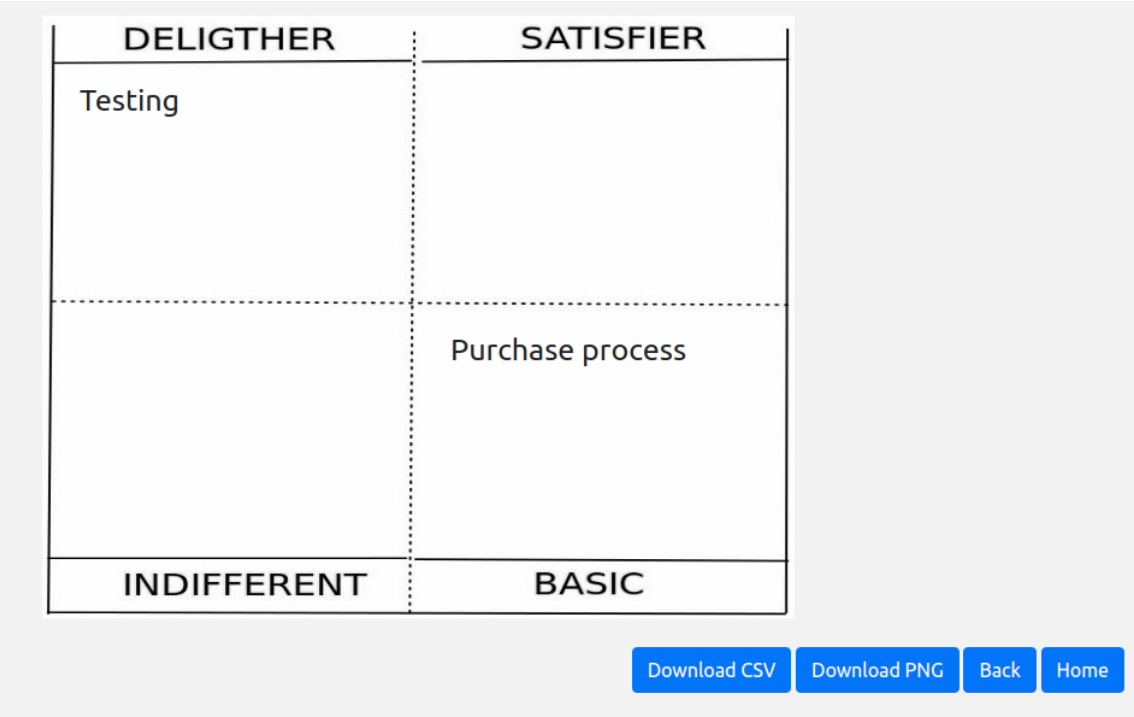

Figura 5.16: Agrupaciones colocadas en el modelo Kano.

La última página muestra el modelo Kano y las agrupaciones creadas por el usuario. Estas agrupaciones pueden ser movidas y colocadas en el modelo Kano. Tras esto, existe la opción de descargar el modelo Kano en distintos formatos.

5.3. Diseño servicios

Los servicios han pasado por varios diseños a lo largo del desarrollo de la aplicación, por diferentes motivos, los cuales se tratarán en el apartado de 6. Los servicios se separan en cuatro: el encargado de realizar el *scrapping*, es decir, el encargado de extraer la información y comentarios de las extensiones; el servicio de *feature detect* o detección de características, el cual tiene el trabajo de detectar si un comentario está solicitando una funcionalidad nueva; y por último el *keyword extract* o extracción de palabras clave, cuya

tarea es extraer los *tags* de los comentarios. En este apartado se hablará de los distintos diseños adoptados a lo largo del desarrollo.

5.3.1. Incluidos en la aplicación

El primer diseño de los servicios era el de incluirlos en la aplicación, de esta manera la aplicación sería *all-in-one*, es decir, autosuficiente. Este diseño tenía las siguientes ventajas y desventajas:

- Ventajas
  - La aplicación no necesita de servicios externos para funcionar.
- Desventajas
  - Los servicios no podrán ser reutilizables.
  - Ejecutar los servicios dentro de la aplicación podrá ralentizar el funcionamiento de la misma.

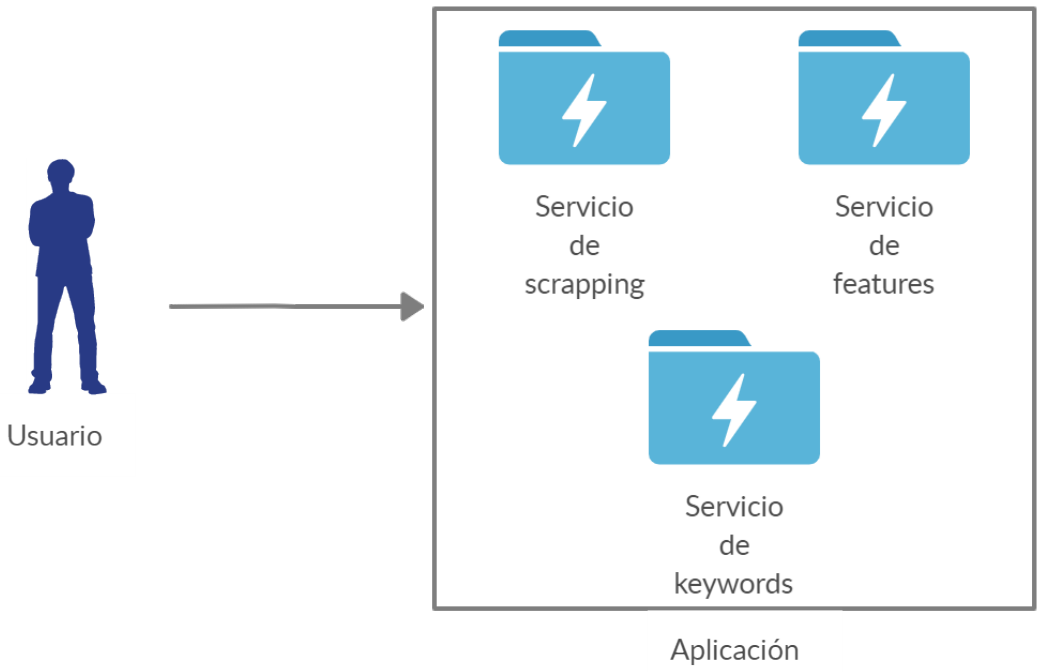

Figura 5.17: Servicios incluidos en la aplicación.

5.3.2. Node Express + Docker

El segundo diseño fue sacar los servicios fuera de la aplicación, el de *scrapping* se ejecutaría mediante Node, y los otros dos mediante Docker.

- Ventajas
  - Los servicios pueden ser reutilizados.
  - Los servicios se ejecutan de manera individual sin afectar al rendimiento del resto, ni de la aplicación.
- Desventajas
  - Los servicios deben ser desplegados en algún servidor, o en local.
  - Es necesario tener Docker y Node para utilizar este método.

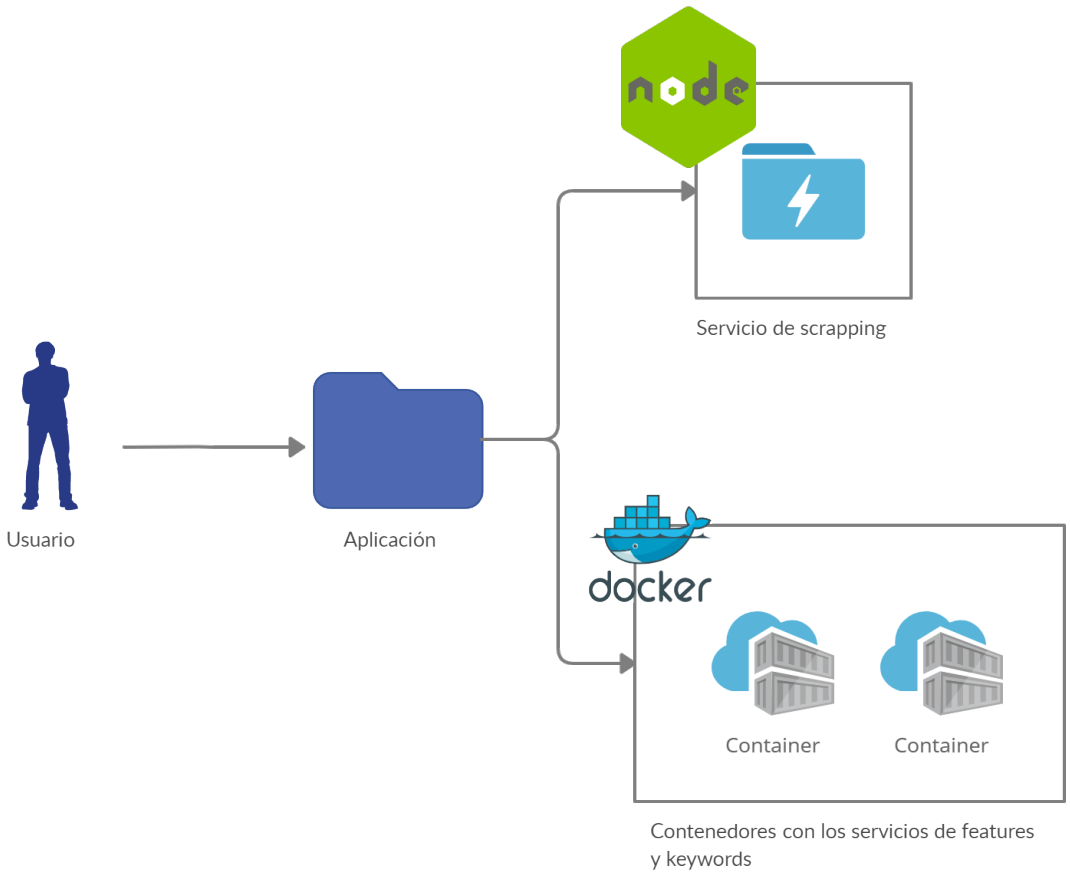

Figura 5.18: Servicios mediante Node Express + Docker.

5.3.3. Docker compose

El último diseño era parecido al anterior, solo qué ejecutando todos los servicios mediante Docker, y desplegándolos simultáneamente mediante Docker compose.

- Ventajas
  - Los servicios pueden ser reutilizados.
  - Los servicios se ejecutan de manera individual sin afectar al rendimiento del resto, ni de la aplicación.
  - Los servicio se pueden desplegar simultáneamente mediante el uso de un solo comando.
  - Los servicios son fácilmente escalables, por lo cual se pueden desplegar tantas veces como sea necesario.
- Desventajas
  - Los servicios deben ser desplegados en algún servidor, o en local.
  - Es necesario Docker y Docker compose para este método.

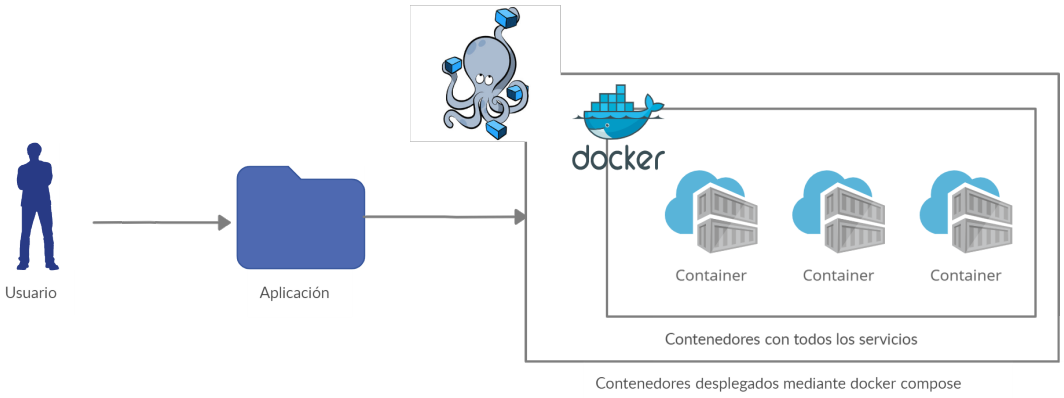

Figura 5.19: Servicios mediante Docker compose.

6. CAPÍTULO

Desarrollo

En este capítulo se hablará del desarrollo de los servicios y la aplicación, y los problemas ocurridos durante dichos desarrollos.

6.1. Desarrollo aplicación

La aplicación ha sido desarrollada mediante HTML, CSS y JavaScript. La implementación de la aplicación se puede separar en dos partes, la del *front end* que son las GUIs que va a ver el usuario, las cuales se han hecho mediante HTML y CSS. Por otra parte, el *back end*, que es el código escrito en JavaScript, y es el encargado de realizar las consultas a los servicios, cargar los resultados, guardar la información necesaria...

En cuanto al *front end*, no hay mucho que comentar que no se haya hecho ya. La aplicación dispone de cuatro páginas y tienen los siguientes componentes:

- **Search webs:** Dispone de botones e *inputs* para recibir la búsqueda del usuario. Tras esto, muestra los resultados en una *table*.
- **Extension selection:** Dispone de una *table* donde se van a mostrar todos los resultados obtenidos de la búsqueda del usuario. Cada fila de la tabla, dispone de un *select* para que el usuario seleccione las extensiones que le interesen.
- **Tags aggrupation:** Muestra un *input* pidiendo cuantas palabras claves se desea extraer. Tras esto hay un *cloud tag* con, el cual mostrará todas las palabras clave extraídas. Mediante el uso de los botones incluidos, el usuario realizará las agrupaciones que el vea necesarias.

- **Kano model:** Muestra una imagen del modelo Kano y un *cloud tag* con todas las agrupaciones creadas por el usuario. También se muestran unos *draggables*, los cuales son los que el usuario puede mover al modelo Kano.
- **Página de configuración:** Dispone de varios *inputs* mediante los cuales recoge las IPs de los servicios.

Por otro lado, tenemos el *back end*, del cual se van a mencionar los apartados más importantes del código, ya que hay algunas funciones como podrían ser: limpiar los *inputs*, añadir resultados a tablas, lanzar mensajes de feedback... Las cuales son funciones simples de comprender tan solo leyendo el código. El desarrollo del back end de la aplicación se detalla en el Anexo A.

6.2. Desarrollo servicios

Como se ha comentado en el apartado 5.3, se disponen de tres servicios, de los cuales dos ya estaban previamente creados (el de extracción de palabras clave y el de detección de funcionalidades), por lo cual no fue necesario su desarrollo, pero sí su inclusión con *Docker*, la cual se hablará en el Anexo B.3. Por lo tanto, en esta parte se hablará principalmente del desarrollo del servicio encargado de extraer la información de las extensiones mediante el *scrapping* y los problemas en su desarrollo. El desarrollo de los servicios se detallan en el Anexo B.

7. CAPÍTULO

Pruebas

En este capítulo se hablarán sobre las pruebas realizadas, tanto las funcionales durante el desarrollo del proyecto, como las pruebas con usuarios, una vez terminado el proyecto.

7.1. Pruebas funcionales

A lo largo del desarrollo de la aplicación, se han ido creando y ejecutando distintas pruebas para comprobar el correcto funcionamiento de la aplicación. Estas pruebas se realizaron de manera manual, ya que no suponían demasiado esfuerzo en su comprobación, debido a que la aplicación solo dispone de 4 páginas. El objetivo de estas pruebas fueron: verificar que la aplicación funciona correctamente; identificar posibles errores y solucionarlos.

7.1.1. Verificación funcionalidades generales

En estas pruebas, se analizan las funcionalidades generales de la aplicación, como puede ser la navegación entre páginas, la recolección de los datos... En la Tabla 7.1 se muestran las pruebas asociadas a dichas funcionalidades.

Grupo	Prueba	Descripción	Salida esperada	Salida real	Observaciones
Enrutamiento	1.1	Redirección entre las distintas páginas de la aplicación.	Solo es posible moverse entre las oáginas mediante los botones de 'Next' y 'Back'.	Solo es posible moverse entre las oáginas mediante los botones de 'Next' y 'Back'.	Correcto
	1.2	Recarga de páginas.	Al volver a una página previamente utilizada, no se pierden datos.	Al volver a una página previamente utilizada, no se pierden datos, a excepción de la primera página.	Mejorable
	2.1	Datos vacíos introducidos	En caso de que el 'Purpose' esté vacío, se informa al usuario y no se realiza la búsqueda.	En caso de que el 'Purpose' esté vacío, se informa al usuario y no se realiza la búsqueda.	Correcto

Recogida de datos	2.2	Introducir varios datos	En caso de que se introduzcan varios valores, se realizará una búsqueda utilizando todos los datos introducidos.	En caso de que se introduzcan varios valores, se realizará una búsqueda utilizando todos los datos introducidos.	Correcto
	3.1	Presentación de las palabras clave	Se muestran tantas palabras clave como haya indicado el usuario.	Se muestran tantas palabras clave como haya indicado el usuario.	Correcto
	3.2	Presentación de las agrupaciones	Se muestran las agrupaciones que ha creado el usuario.	Se muestran las agrupaciones que ha creado el usuario.	Correcto

Presentación datos	3.3	Presentación de las extensiones	Se muestra la siguiente información de las extensiones: Nombre, Valoración, Usuarios, Última actualización, Versión, URL y la descripción.	Se muestra la siguiente información de las extensiones: Nombre, Valoración, Usuarios, Última actualización, Versión, URL y la descripción.	Correcto

Tabla 7.1: Conjunto de pruebas para la verificación de las funcionalidades generales de la aplicación.

Resultados mejorables:

- Prueba 1.2: Al volver a la primera página (la página de 'Search String'), se pierden los datos de las búsquedas previas realizadas.

7.1.2. Verificación de manejo de datos

A parte de las pruebas previas, también se han realizado pruebas para la verificación del manejo de datos, es decir, si los datos se extraen correctamente, si los datos se mantienen coherentes durante toda la ejecución de la aplicación... Estas pruebas se representan en la Tabla 7.2.

Grupo	Prueba	Descripción	Salida esperada	Salida real	Observaciones
Guardado de datos	1.1	Los datos extraídos son guardados correctamente.	Los datos se guardan mediante el uso de 'localStorage'.	Los datos se guardan mediante el uso de 'localStorage'.	Correcto
Consulta de datos	2.1	Ver los datos recogidos hasta ahora.	Los datos solo se mostrarán en la página correspondiente.	Los datos solo se mostrarán en la página correspondiente.	Correcto
	3.1	Extraer datos de las extensiones	Se extraen los datos necesarios de las extensiones, los comentados en la prueba 3.3 del apartado anterior, más los comentarios de las mismas.	Se extraen los datos necesarios de las extensiones, los comentados en la prueba 3.3 del apartado anterior, más los comentarios de las mismas.	Correcto
	3.2	Extraer comentarios de las extensiones.	Se extraen todos los comentarios de las extensiones seleccionadas.	Se extraen todos los comentarios de las extensiones seleccionadas.	Correcto

Extracción datos	3.3	Filtrado de los comentarios que solicitan una característica.	Se analizan todos los comentarios extraídos.	Se analizan un máximo de 70 comentarios.	Incorrecto
	3.4	Extracción palabras clave	Se extraen la cantidad palabras clave que el usuario haya escogido de los comentarios filtrados como solicitud de características.	Se extraen la cantidad palabras clave que el usuario haya escogido de los comentarios filtrados como solicitud de características, con un máximo de 70 comentarios.	Mejorable
Exportación datos	4.1	Guardado modelo Kano.	El resultado del modelo Kano se puede guardar tanto como imagen o como CSV.	El resultado del modelo Kano se puede guardar tanto como imagen o como CSV.	Correcto

Tabla 7.2: Conjunto de pruebas para la verificación de la extracción de los datos de la aplicación.

Resultados incorrectos:

- Prueba 3.3: Debido a un bug no identificado, solo es posible realizar el análisis de un máximo de 70 comentarios, ya que el servicio de detección de características falla en caso contrario.

Resultados mejorables:

- Prueba 3.4: Este resultado viene ligado al anterior, en cualquier caso solo se podrán extraer las palabras clave de un máximo de 70 comentarios.

7.2. Pruebas con usuarios

Para las pruebas con usuarios, se escribió una guía de uso (la cual también servirá en un futuro para los posibles usuarios de la aplicación (Anexo D)). Estas pruebas se realizaron con integrantes del grupo de investigación Onekin. El objetivo de estas pruebas, era comprobar como funcionaba tanto la aplicación como los servicios en distintos sistemas operativos y dispositivos. En la Tabla 7.3 se muestran los errores identificados, y en la Tabla 7.4 sugerencias de cambios.

Error	Solución	Coste en tiempo
El servicio de scrapping daba un error de 'nullContent'	Cambiar los 'timeOuts' por 'waitForSelector'. De esta manera a la hora de que puppeteer recoja información, se esperará hasta que la página esté cargada, ya que este tiempo varía entre los dispositivos.	Bajo
El servicio de detección de características no realizaba su función correcamente	Al subir los archivos a GitHub no se subía un .jar que era necesario debido al .gitignore. Por lo cual se modificó el .gitignore para que se subiera todo el servicio a GitHub.	Bajo
El servicio de detección de características da un error cuando se le pasan más de 70 comentarios.	No se ha podido averiguar el error exacto por el que esto sucede, pero se cree que ha sido a raíz de pasar los servicios a 'Docker compose'. Para que el programa funcione hasta descubrir porque sucede este bug, se ha procedido a limitar la cantidad de comentarios que se analizan a 70.	Medio

Tabla 7.3: Tabla de errores identificados por usuarios.

Comentario	Solución	Coste en tiempo
El feedback para el usuario no es suficientemente visible	Se aumentó el feedback. Se utilizaron más alertas avisando al usuario de cuantos comentarios se han extraído, cuantos de esos comentarios solicitan características...	Bajo

Tabla 7.4: Tabla de sugerencias de usuarios.

8. CAPÍTULO

Seguimiento y control del proyecto

Durante el desarrollo del proyecto, se ha gestionado el control y seguimiento del mismo, supervisando las tareas realizadas para que estas se desarrollaran bajo los marcos de tiempo definidos en la planificación. En este apartado se comentarán tanto los cambios realizados en la planificación, como la desviación den tiempo y las incidencias ocurridas, entre otras cosas.

8.1. Gestión del alcance

El alcance del proyecto no ha sufrido grandes cambios, ya que desde el principio era un alcance acotado, y el cual se podía sacar adelante en el tiempo establecido. Aún así, debido a algunas incidencias las cuales se detallan en la sección 8.2, el diagrama de Gantt ha sufrido alguna modificación. A parte de esto, como los cambios que se iban realizando en la aplicación se mostraban mediante las reuniones, no fue necesario la entrega de un prototipo, por lo que el diagrama EDT también ha sufrido algún cambio menor.

8.1.1. Cambios en el diagrama EDT

Como se ha comentado previamente, no fue necesaria la entrega del prototipo, por lo cual este paquete de trabajo desaparece tanto del EDT como del diagrama de dependencias entre las tareas. El resto de tareas se han mantenido iguales, los únicos cambios a parte de este, han sido los tiempos previstos, los cuales se detallan en la sección 8.3.

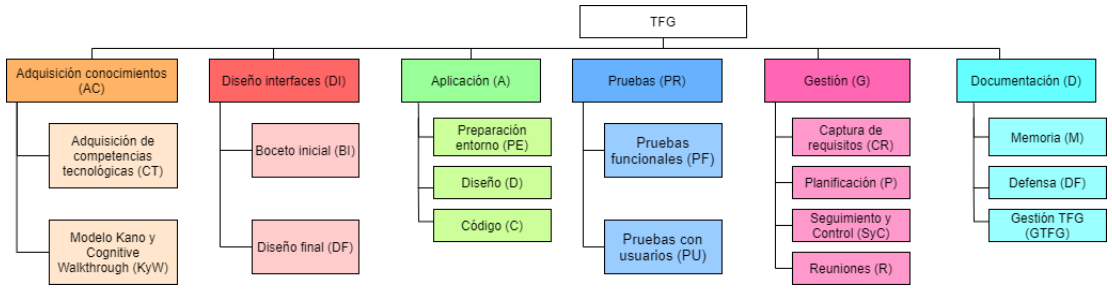

Figura 8.1: Diagrama EDT tras la finalización del proyecto.

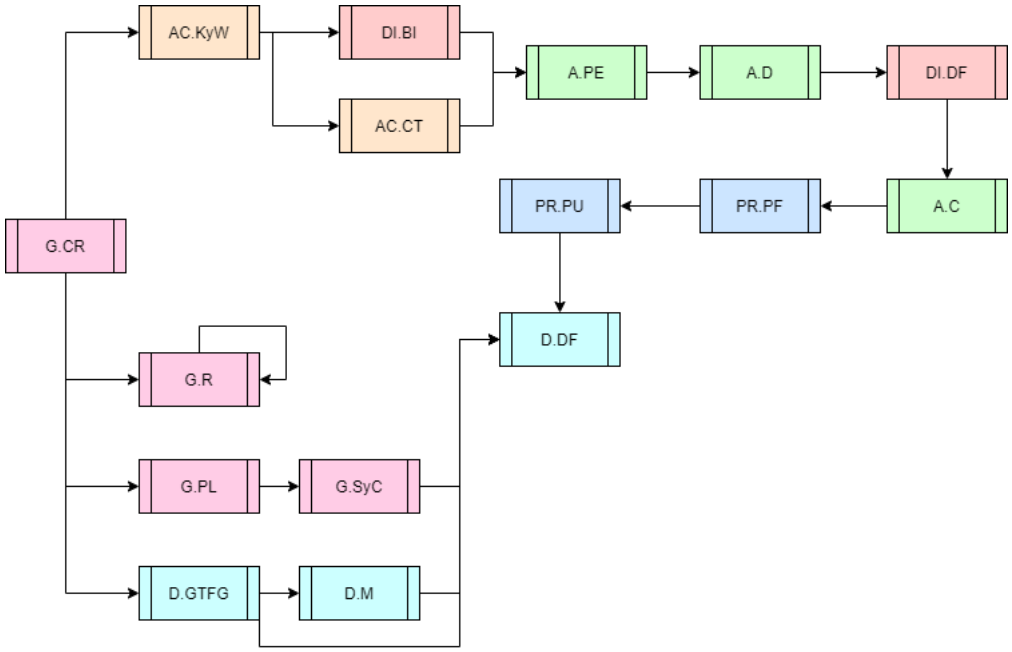

Figura 8.2: Diagrama de dependencias tras la finalización del proyecto.

8.1.2. Cambios en el diagrama Gantt

Como se puede observar en el diagrama de Gantt, se ha eliminado el paquete del prototipo, y se han unido las estimaciones de tiempo del prototipo con el de la aplicación final. A parte de esto, las pruebas con usuarios y la entrega de la aplicación se han atrasado debido a los problemas que se explican en la sección 8.2. Por un lado, la gestión del TFG se ha comenzado antes de lo previsto. Debido al retraso de la entrega de la aplicación, se han aumentado los tiempos destinados a las reuniones, al seguimiento y control y a las pruebas funcionales.

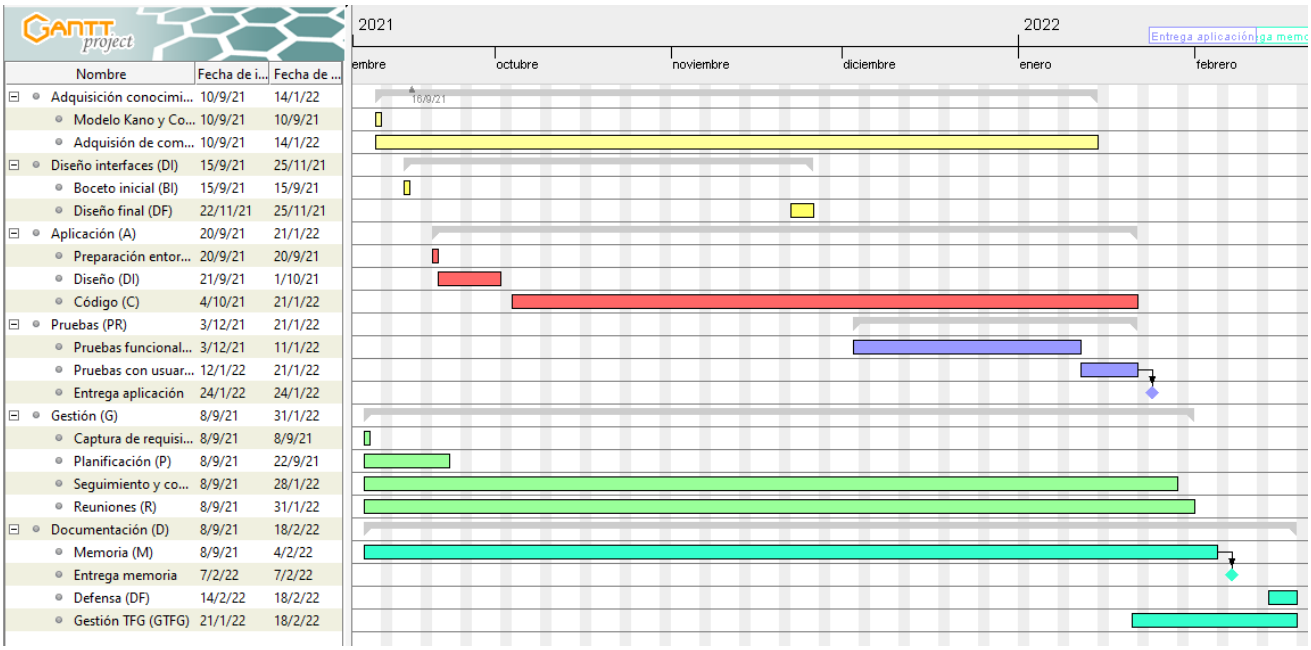

Figura 8.3: Diagrama de Gantt tras la finalización del proyecto.

8.2. Gestión de incidencias y riesgos

En la planificación se definieron unos riesgos con sus respectivas soluciones o maneras de minimizarlos en el caso de que sucedieran a la hora del desarrollo del proyecto. Aún habiendo hecho una previsión de riesgos, algunos de estos se han acabado convirtiendo en incidencias, las cuales han retrasado las tareas.

A continuación se detallan dichas incidencias, el impacto que han tenido sobre el desarrollo del proyecto y la solución llevada a cabo.

8.2.1. R3-Uso de tecnologías desconocidas

Desde el comienzo del TFG, se sabía que se iban a utilizar herramientas desconocidas por el alumno, de las cuales iba a ser necesario un tiempo de aprendizaje y de adaptación a las mismas.

La mayoría de las herramientas no dieron mucho problema, o en caso de darlo, como fue al principio del proyecto dichos problemas estaban dentro de los previamente planificado. La incidencia ocurrió con *Docker compose*, ya que era la herramienta más desconocida para el alumno, y su uso fue casi al final del desarrollo de la aplicación, por lo cual debido al desconocimiento de la misma,

Esta incidencia afectó tanto al comienzo de pruebas con usuarios, como a la entrega de la aplicación, ya que se tardó más de lo previsto en tener preparados los servicios en *Docker compose*, y esto retraso las dos tareas previamente comentadas. De todas maneras, estos desvíos no han sido tan graves, y se han podido mitigar gracias a la ayuda y consejos de algunos integrantes del grupo Onekin.

8.2.2. R5-Pérdida del sistema de información

Este riesgo tenía una probabilidad de ocurrir muy baja, pero acabó sucediendo y convirtiéndose en una incidencia. El alumno ha utilizado el sistema operativo Ubuntu para llevar a cabo el TFG, y un día, por motivos desconocidos (probablemente una actualización que no se instaló correctamente), todo el sistema operativo quedó inutilizable. Para solucionar esto, fue necesario reinstalar el sistema operativo, y por consiguiente, todas las copias y archivos almacenados en el dispositivo fueron eliminadas.

Esta incidencia puede sonar como algo muy grave, pero gracias al control de riesgos, y a que se disponía de varias copias de seguridad (tanto en Drive como en GitHub), fue sencillo recuperar todo el proyecto, sin ninguna pérdida de información ni pérdida en el proceso del TFG. El impacto de esta incidencia fue mínimo en pérdida de información, pero si que fue necesario emplear tiempo para solucionar el problema y poner todas las herramientas en funcionamiento de nuevo. Por lo cual, este es otro de los motivos por el que las pruebas con usuarios fueron retrasadas.

8.3. Gestión del tiempo

Debido a las incidencias y cambios en las tareas previamente comentados, el proyecto ha sufrido desviaciones, tanto en en periodos de realización de tareas y de tiempos previstos.

8.3.1. Desviación en las fases

Tanto las incidencias como los cambios en relación al prototipo, han dado como resultado que las fechas de finalización de las tareas también varíen. Estas variaciones se muestran en la Tabla 8.1 siendo los colores de la siguiente manera: verde en caso de estar correcto o adelantado; naranja en caso de retrasarse menos de un mes; y rojo en caso de retrasarse más de un mes.

Tarea	Fecha fin prevista	Fecha fin real	Desviación
Adquisición de conocimientos (AC)	20/12/21	14/1/22	+3 semanas
Modelo Kano y Cognitive Walkthrough	10/9/21	10/9/21	
Adquisión de competencias tecnologías	20/12/21	14/1/22	+3 semanas
Diseño de interfaces (DI)	25/11/21	25/11/21	
Boceto inicial	15/9/21	15/9/21	
Diseño final	25/11/21	25/11/21	
Aplicación (A)	10/12/21	21/1/22	+1 mes y medio
Preparación entorno	21/10/21	20/9/21	-1 mes
Diseño	25/10/21	1/10/21	-3 semanas
Código	10/12/21	21/1/22	+1 mes y medio
Pruebas (PR)	27/12/21	21/1/22	+1 mes y medio
Pruebas funcionales	20/12/21	11/1/22	+3 semanas
Pruebas con usuarios	21/12/21	21/1/22	+1 mes y medio
Gestión (G)	28/12/21	31/1/22	+1 mes
Captura de requisitos	8/9/21	8/9/21	
Planificación	22/9/21	22/9/21	
Seguimiento y control	24/12/21	28/1/22	+1 mes
Reuniones	28/12/21	31/1/22	+1 mes
Documentación (D)	18/2/22	18/2/22	
Memoria	4/2/22	4/2/22	
Defensa	18/2/22 (aprox.)	18/2/22 (aprox.)	
Gestión TFG	18/2/22 (aprox.)	18/2/22 (aprox.)	

Tabla 8.1: Tabla comparativa de las fechas previstas y las fechas reales de la finalización de las tareas.

8.3.2. Desviación en los paquetes

Como ya se ha comentado previamente, las tareas de Prototipo y Aplicación se han unido en una misma, llamada Aplicación, por lo cual las horas estimadas y totales se han sumado, es decir, si de estimación el Prototipo tenía un estimado de 30 horas, y la Aplicación un estimado de 50 horas, el total de la Aplicación pasa a ser 80 horas.

Para representar estas desviaciones se utilizará la Tabla 8.2, mediante la cual se muestra la comparativa entre las horas estimadas y las reales. A parte de esta tabla, se utilizarán dos gráficos para mostrar mejor los resultados.

BLOQUE DE TRABAJO	TAREA	ESTIMACIÓN	DEDICACIÓN REAL	DIFERENCIA
DISEÑO INTERFACES (DI)	Boceto inicial (BI)	4	4	0
	Diseño final (DF)	16	10	-6
	SUBTOTAL	**20**	**14**	-6
APLICACIÓN (A)	Preparación entorno (PE)	8	6	-2
	Diseño (D)	20	12	-8
	Código (C)	110	137	+27
	SUBTOTAL	**138**	**155**	+17
PRUEBAS (PR)	Pruebas funcionales (PF)	36	26	-10
	Pruebas con usuarios (PU)	10	12	+2
	SUBTOTAL	**46**	**38**	-8
GESTIÓN (G)	Captura de requisitos (CR)	4	2	-2
	Planificación (P)	20	23	+3
	Seguimiento y Control (SyC)	6	8	+2
	Reuniones (R)	14	12	-2
	SUBTOTAL	**44**	**45**	+1
DOCUMENTACIÓN (D)	Memoria (M)	50	72	+22
	Defensa (DF)	6	5	-1
	Gestión TFG (GTFG)	4	6	+2
	SUBTOTAL	**60**	**83**	+23
ADQUISICIÓN CONOCIMIENTOS (AC)	Adquisión de competencias tecnologías (CT)	20	33	+13
	Modelo Kano y Cognitive Walkthrough (KyW)	4	7	+3
	SUBTOTAL	**24**	**40**	+16
PROYECTO		**332**	**375**	+43

Tabla 8.2: Comparativa de las horas estimadas y las horas reales de las tareas

Como se puede observar, que el proyecto se ha desviado alrededor de un **13 %** respecto a la planificación. Esto se debe a las incidencias ocurridas y a que algunas tareas han llevado más tiempo del previsto. De todas maneras, esta desviación no es enorme, y se puede afirmar que la planificación ha sido efectiva a la hora de llevar adelante el proyecto. Aún así, se espera que estas desviaciones se vayan reduciendo en futuros proyecto a medida que el alumno adquiera más experiencia en realizar planificaciones.

Para finalizar se adjuntan los gráficos mediante los cuales se espera una mejor visualización de las diferencias entre las horas totales y las estimadas.

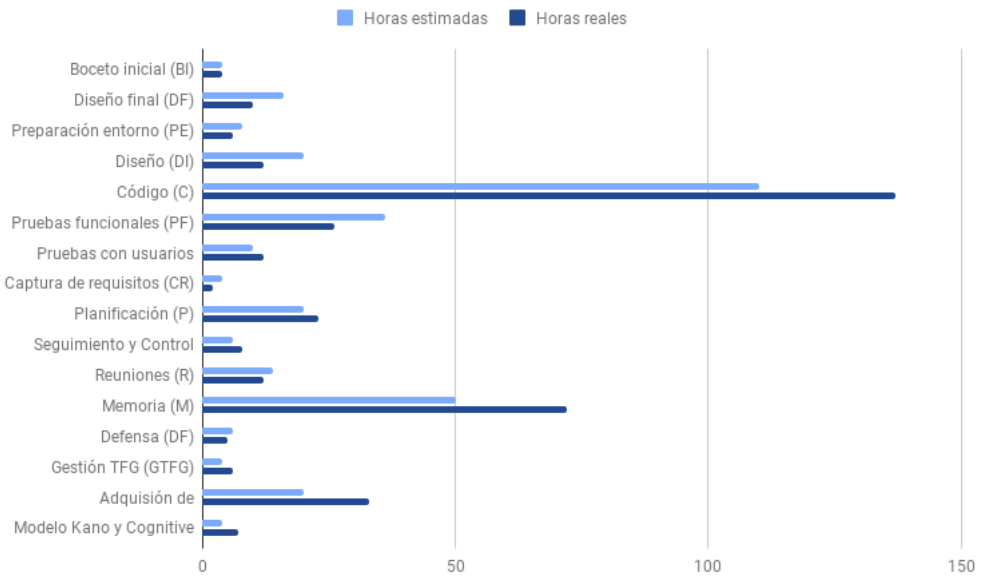

Figura 8.4: Gráfico comparación horas estimadas y reales de cada tarea.

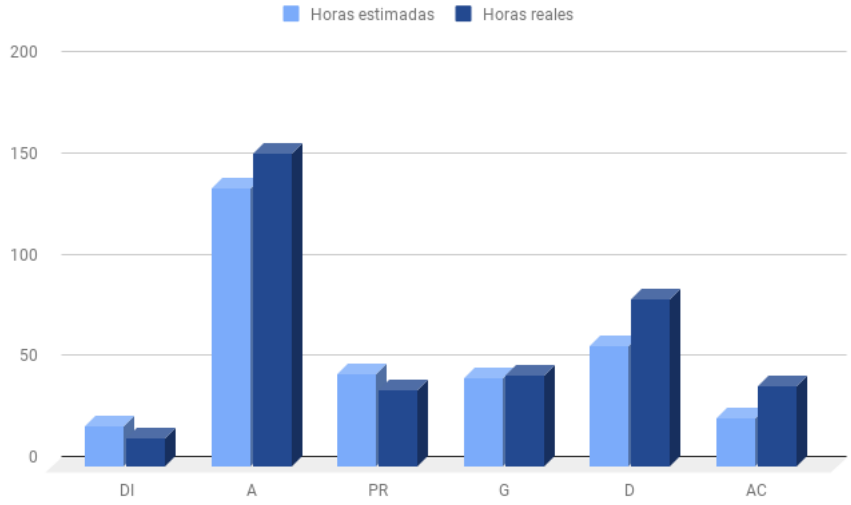

Figura 8.5: Gráfico comparación horas estimadas y reales de cada paquete.

9. CAPÍTULO

Conclusiones y líneas futuras

En este último capítulo se comenzará hablando sobre las conclusiones (9.1). Tras estas conclusiones también se enumeran las posibles mejoras (9.2), estas mejoras son características que se han ido identificando a lo largo del proyecto, pero que no se han podido incluir por falta de tiempo. Para finalizar, se hará un repaso de las posibles líneas futuras de trabajo (9.3).

9.1. Conclusiones

El objetivo principal de este TFG, era desarrollar la aplicación que realiza análisis de mercado de la Web Store de Google Chrome. Una vez finalizado este proyecto, se puede afirmar qué las expectativas se han cumplido. En este apartado se desglosaran las conclusiones en dos apartados: tanto a nivel técnico como personal.

9.1.1. Conclusiones a nivel técnico

En general, la valoración técnica del proyecto ha sido positiva. Se han conseguido desarrollar los requisitos principales del proyecto, aunque los secundarios no ha sido posible desarrollarlos. Se han alcanzado los siguientes objetivos técnicos:

- **Aplicación para análisis de extensiones:** se ha implementado una aplicación mediante la cual, sea posible realizar un análisis de mercado de las extensiones de Google Chrome. Mediante este análisis el usuario de la aplicación puede conocer más sobre las necesidades y solicitudes de los usuarios de extensiones, en relación

a las funcionalidades de dichas extensiones. De esta manera, el usuario puede utilizar el resultado del análisis para implementar su propia extensión con la cual pueda satisfacer al mayor número de usuarios.

- **Servicios para análisis de extensiones:** se ha implementado un *pack* de servicios, mediante *docker* y *docker compose*, con el cual se puede extraer información de las extensiones. La idea de este *pack* es que pueda ser reutilizado por cualquier persona que necesite realizar *scrapping* de las extensiones de Google Chrome, ya que como se ha mencionado previamente en este documento, Chrome no ofrece una API mediante la cual extraer la información de las extensiones sin necesidad de realizar *scrapping*. A parte de esto, dos de los servicios pueden ser utilizados en otras herramientas en las qué se necesite catalogar comentarios como positivos o negativos, y extraer las palabras claves de dichos comentarios.
- **Interoperabilidad de herramientas:** se ha conseguido juntar el funcionamiento de varias herramientas, para crear una única aplicación y unos servicios, que las utilice y unifique.

9.1.2. Conclusiones a nivel personal

En este apartado se hablarán sobre las lecciones aprendidas durante el desarrollo del proyecto, y las cuales serán útiles en el futuro ámbito laboral del estudiante. Los objetivos personales alcanzados durante el TFG son los siguientes:

- **Aprendizaje de herramientas para el desarrollo de aplicaciones:** para el desarrollo de este proyecto, ha sido necesario utilizar distintas herramientas de desarrollo de aplicaciones, las cuales son muy utilizadas en el ámbito laboral (detalladas en el Capítulo 4).
- **Satisfacción del cliente:** mediante el entendimiento del modelo Kano, también se ha aprendido sobre la importancia de la satisfacción del cliente, y lo necesario que es cumplir unos requisitos mínimos para poder complacer al cliente. De la misma manera que estos requisitos mínimos, se ha aprendido sobre las funcionalidades que pueden satisfacer más a los usuarios y el interés a la hora de identificarlas, y de esta manera para poder implementar una aplicación con más éxito entre los usuarios.
- **Necesidad de prototipos:** previo al desarrollo de la aplicación, se realizó un prototipo, el cual fue muy útil a la hora de realizar el diseño final, pues ya estaba gran

parte de las interfaces definidas. A parte de esto, se ha aprendido sobre un método de inspección de GUIs, con el cual se pueden diseñar y evaluar la usabilidad de las interfaces gráficas sin necesidad de usuarios para probarlas.

- **Gestión y desarrollo de un proyecto real:** durante la carrera se han desarrollado multitud de proyectos, tanto en grupo como de manera individual. Aún así, este proyecto es distinto a los mencionados, ya que es un proyecto que inicia desde 0 con un análisis de requisitos y funcionalidades, a diferencia de los desarrollados durante la carrera que ya suelen tener un objetivo fijo. A parte de esto, ha sido necesario realizar una planificación previa al desarrollo del TFG, y un seguimiento y control durante el desarrollo del mismo. Además, se han tenido incidencias y contratiempos que no suelen estar presentes en los proyectos de carrera, pero si en los proyectos laborales.

9.2. Posibles mejoras

El proyecto se ha finalizado cumpliendo los requisitos principales solicitados en el análisis de requisitos. Sin embargo, no se han podido cumplir los requisitos secundarios, e incluso los implementados, pueden mejorarse para conseguir una mejor aplicación. Ahora se enumerarán las posibles mejoras que se le pueden realizar al programa:

- **Mejorar las GUIs:** el diseño es muy simple e intuitivo, por lo cual es fácil de utilizar, pero al ser tan simple queda con un acabado visual muy poco profesional. Al ser una prueba de concepto, se ha preferido no invertir más tiempo del necesario en las interfaces, pero aun así, sería interesante mejorar las GUIs visualmente, procurando no dificultar el uso de la aplicación.

- **Historial:** la aplicación actualmente solo muestra el historial de las búsquedas realizadas en una misma sesión. Sería interesante que la aplicación tuviera la opción de recuperar análisis previos, y realizar comparaciones entre los mismos.

- **Guardado en la nube:** poder subir el resultado del análisis a la nube (Drive, Dropbox...).

- **Formatos de exportación:** añadir más formatos a la hora de exportar el resultado del modelo Kano.

9.3. Líneas futuras

En este último apartado, se detallan las líneas futuras que se podrían llevar a cabo para mejorar la aplicación. La diferencia con las mejoras previamente mencionadas es que para llevar estas a cabo, supondría un desarrollo más complejo y costoso.

- **Mejorar los servicios:** el servicio de extracción de palabras clave, dependiendo de cuantas extensiones y comentarios haya en la búsqueda, puede extraer palabras con poco sentido o sin coherencia para el análisis que se esté realizando. Esto se debe a la implementación de dicho servicio, ya que da importancia a palabras en otros idiomas, entre otras cosas. Por lo cual, una mejora sería modificar este servicio para que disponga de mayor fiabilidad y extraiga el menor número posible de palabras redundantes. Esto también sucede con el servicio de detección de características, que en algunos casos puede dar falsos positivos o a la inversa, por lo cual sería interesante mejorar la clasificación de dicho servicio.
- **Automatización de modelo Kano:** la aplicación actualmente deja en manos del usuario todo el modelo Kano, por lo que una mejora sería que la misma aplicación realizara tanto las agrupaciones de las características, como la clasificación del modelo Kano. De esta manera, el usuario solo debería realizar la búsqueda y dejar el resto en manos de la aplicación.

Anexos

A. ANEXO

Desarrollo de la aplicación

A.1. Uso de localStorage

Al cambiar de página o incluso recargar la misma, los datos se pierden, para evitar esto, se usa *localStorage*, que es una variable global capaz de almacenar datos. Sus dos métodos principales son:

- **setItem(key, value):** Sirve para guardar el valor (value) bajo un ID (key).
- **getItem(key):** Devuelve el valor asignado a la ID (key).

En el caso de la aplicación, se utiliza entre otras cosas, para guardar las URLs de las extensiones una vez que se ha realizado la búsqueda, de esta manera no se pierden al navegar entre las páginas.

Código A.1: Ejemplo de uso guardado *localStorage()*

```
1 localStorage.setItem(queryForTable + 'URL', JSON.stringify(responseURLs));
2 localStorage.setItem(queryForTable + 'INFO', JSON.stringify(extensionsInfo)
      );
```

Con las líneas de código previas, se guardan todas las URLs y las descripciones de las extensiones en esta variable global, para que sea posible acceder a ellas en un futuro, cuando es necesario extraer los comentarios por ejemplo.

Otro uso de esta variable global, es a la hora de introducir las IPs de los servicios.

Código A.2: Guardando las IPs en *localStorage()*

```
// Coger las IPs de los servicios
document.getElementById('cors').value = cors;
document.getElementById('scrapping').value = scrapping;
document.getElementById('features').value = features;
document.getElementById('keywords').value = keywords;

// Guardar las IPs en localStorage
localStorage.setItem('CORS-ANYWHERE', cors);
localStorage.setItem('SCRAPPING-SERVICE', scrapping);
localStorage.setItem('FEATURE-DETECTION-SERVICE', features);
localStorage.setItem('KEYWORD-EXTRACTION-SERVICE', keywords);
```

Y luego cada vez que es necesario recuperar una IP para su uso, se haría de la siguiente manera:

Código A.3: Cogiendo las IPs de *localStorage()*

```
let featureDetectionIP = localStorage.getItem('FEATURE-DETECTION-SERVICE');
let corsProxyIP = localStorage.getItem('CORS-ANYWHERE');
```

Por lo cual, el uso de esta variable global es muy necesario y ayuda de enorme manera, ya que evita que tengamos que crear documentos o ficheros temporales para guardar la información o valores que serán necesarios más adelante en la ejecución del programa.

A.2. Uso de fetch()

Para poder realizar las peticiones a los servicios, ha sido necesario el uso de *fetch()*, que sirve para mandar dichas peticiones con la información necesaria. En total hay 5 funciones que hacen uso del fetch

Código A.4: Función *getURLs()*

```
async function getURLs(query) {
    let scrappingServiceIP = localStorage.getItem('SCRAPPING-SERVICE');
    return await fetch('http://' + scrappingServiceIP + '/searchExtensions?
    q=' + query)
      .then(res => res.text())
      .then(text => JSON.parse(text))
      .catch((error) => {
```

```
7           alert("Something went wrong, please restart the service and try
    again.");
8           console.log(error);
9        });
10   }
```

Lo primero que se hace es usar *localStorage* para recuperar la IP del servicio de *scrapping*. Tras esto se usa *fetch()* mediante una URL parametrizada (en la cual se pasa la búsqueda del usuario). Se espera a la respuesta, en caso de ser correcta, se devuelve dicha respuesta convertida a JSON, en caso de que se reciba un error, se imprime por la consola y se le avisa al usuario del error.

Código A.5: Función *getDescriptions()*

```
1   async function getDescriptions(query) {
2       let scrappingServiceIP = localStorage.getItem('SCRAPPING-SERVICE');
3       if (query.length !== 0) {
4         return await fetch('http://' + scrappingServiceIP + '/
    extractExtensionInfo', { // the body is an array of URLs
5           method: 'POST',
6           headers: {
7             'Content-Type': 'application/json'
8           },
9           body: JSON.stringify(
10            { query }
11          )
12        }).then(res => res.text())
13          .then(text => JSON.parse(text))
14          .catch((error) => {
15            console.log(error);
16            alert("Something went wrong, please restart the service and try
    again.");
17          });
18      }
19    }
```

Esta función es muy similar a la anterior, solo que en vez de usar una URL parametrizada, se envía un JSON con todas las URLs de las extensiones. Por este motivo, este método es un *POST* y no un *GET*, ya que para poder enviar un JSON es necesario usar el primero mencionado.

Código A.6: Función *getComments()*

```
async getComments(query) {
   let scrappingServiceIP = localStorage.getItem('SCRAPPING-SERVICE');
   return await fetch('http://' + scrappingServiceIP + '/extractComments',
    {
    method: 'POST',
    headers: {
        'Content-Type': 'application/json'
    },
    body: JSON.stringify(
        { query }
    )
    }).then(res => res.text())
    .then(text => JSON.parse(text))
    .catch((error) => {
        alert("Something went wrong, please restart the service and try
    again.");
        console.log(error);
    });
}
```

Esta función es prácticamente igual a la anterior, lo único que cambia es al servicio al que se le manda la petición, y el contenido de la misma.

Código A.7: Función *getFeaturedComments()*

```
async getFeaturedComments(extensionsComments) {
    extensionsComments = extensionsComments.map(function (
    extensionsComments) {
        return extensionsComments.map(function (comment) {
            var rComment = {};
            rComment['author'] = comment.author;
            rComment['review_id'] = 'ri';
            rComment['package_name'] = 'extension';
            rComment['rating'] = comment.stars;
            rComment['title'] = '';
            rComment['body'] = comment.text;
            return rComment;
        })
    })
    extensionsComments = extensionsComments.flat(1);
```

```
15
16      let featureDetectionIP = localStorage.getItem('
        FEATURE-DETECTION-SERVICE');
17      let corsProxyIP = localStorage.getItem('CORS-ANYWHERE');
18      let features = await fetch('http://' + corsProxyIP + '/' +
        featureDetectionIP + '/hitec/classify/domain/google-play-reviews/', {
19          method: 'POST',
20          headers: {
21              'Content-Type': 'application/json',
22              'Accept': 'application/json',
23              "X-Requested-With": "XMLHttpRequest"
24          },
25          body: JSON.stringify(extensionsComments)
26      }).then(res => res.text()
27          .then(text => JSON.parse(text)))
28          .catch((error) => {
29              console.log(error);
30              alert("Something went wrong, please try again.")
31          });
32
33      if (features != null) {
34          features = features.filter(comment => comment['
        cluster_is_feature_request'] === true || comment['cluster_is_bug_report
        '] === true);
35      }
36
37      if (features != null) {
38          return features
39      } else {
40          return null;
41      }
42  }
```

Este método comienza haciendo un *map*, para convertir los comentarios extraídos al formato que acepta el servicio de detección de características. Una vez hecho esto, se realiza un *flat* para tener un *array* de una única dimensión. Tras esto, se realiza la petición *fetch* al servicio, mandándole el *array* que se acaba de crear. Tras esto si la respuesta es correcta, se hace un filtrado (ya que este servicio devuelve que comentarios son solicitudes de características o reportes de bugs) y solo se mantienen los comentarios que aportan algo. En caso de que algún comentario aporte información, se devuelve el *array* formado por

dichos comentarios, en caso de que no haya ninguno se devuelve *null*.

Código A.8: Función *extractTopics()*

```
async extractTopics(featureComments, descriptions, quantity) {
    featureComments = featureComments.map(function (comment) {
        var rComment = {};
        rComment['author'] = comment.review_id;
        rComment['date'] = '';
        rComment['stars'] = comment.rating;
        rComment['text'] = comment.body;
        return rComment;
    })

    let average = featureComments.reduce((s, a) => parseInt(s) + parseInt(
    a.stars), 0) / featureComments.length;

    descriptions = descriptions.map(function (description) {
        var rDescription = {};
        rDescription['author'] = '';
        rDescription['date'] = '';
        rDescription['stars'] = average;
        rDescription['text'] = description;
        return rDescription;
    })
    featureComments = featureComments.concat(descriptions);

    let featureDetectionIP = localStorage.getItem('
    KEYWORD-EXTRACTION-SERVICE');
    let corsProxyIP = localStorage.getItem('CORS-ANYWHERE');
    let commentsKeywords = await fetch('http://' + corsProxyIP + '/' +
    featureDetectionIP + '/topics?items=' + quantity, {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json',
            'Accept': 'application/json',
            "X-Requested-With": "XMLHttpRequest"
        },
        body: JSON.stringify(featureComments)
    }).then(res => res.text()
        .then(text => JSON.parse(text)))
        .catch((error) => {
```

```
36              console.log(error);
37          });
38      return commentsKeywords;
39  }
```

Esta función comienza haciendo un *map*, para convertir los comentarios al formato que acepta el servicio. Tras esto, se saca la puntuación media de todos los comentarios, y se añade al *array* de estos comentarios las descripciones de las extensiones, poniendo dicha media como su ”valoración”. Esto se hace debido a que las descripciones de las extensiones también pueden aportar funcionalidades o información interesante, pero para que no se valoren por encima o por debajo de los comentarios, se les da este valor medio. Para terminar se realiza la petición al servicio, el cual devuelve un *array* con las palabras clave extraídas.

B. ANEXO

Desarrollo de los servicios

B.1. Desarrollo servicio de scrapping

Como ya se ha comentado previamente, para desarrollar este servicio fue necesario utilizar la herramienta *puppeteer*, la cual nos sirve para automatizar las acciones que haría un usuario en un navegador web. El problema de esta herramienta, es que debe ser ejecutada en el *server side*, es decir, necesita funcionar alojada en un servidor. Esto dificultó el desarrollo, ya que la primera idea para este servicio, era que estuviera alojado dentro de la misma aplicación, y se intentaron varias soluciones a este problema.

B.1.1. Solución 1: Uso de Iframes

La primera solución que se probó fue el uso de Iframes, que son componentes con los que se puede incluir contenido de otras páginas webs en la aplicación que se esté desarrollando. La idea de esto era sustituir *puppeteer* por estos Iframes, y cargar las páginas web de las extensiones dentro de la página de la aplicación, de esta manera se podría realizar el *scrapping* dentro de la aplicación, y no sería necesario de herramientas externas.

Para realizar esto, se debía incluir la siguiente línea en el apartado de HTML, de esta manera se incluía un *Iframe* en la página.

Código B.1: Línea de código para introducir el *Iframe*

```
1 <iframe width="1200px" height="250px" id="iframe"></iframe>
```

Y para que el *Iframe* cargara los resultados buscados por el usuario, era necesario incluir la siguiente línea:

Código B.2: Línea de código para cargar el *Iframe*

```
1 document.getElementById("iframe").setAttribute("src", query);
```

Esta línea le da al tributo *src* del *Iframe* un valor, en este caso la búsqueda del usuario, la cual se forma concatenando los diferentes *inputs*, hasta formar una URL la cual carga la Web Store con las extensiones que se buscan bajo los término introducidos por el usuario. Un ejemplo de estas URLs es la siguiente: https://chrome.google.com/webstore/search/annotationAND(highlight)AND(teacher)?_category=extensions.

Con esto, al realizar la búsqueda el resultado era el siguiente:

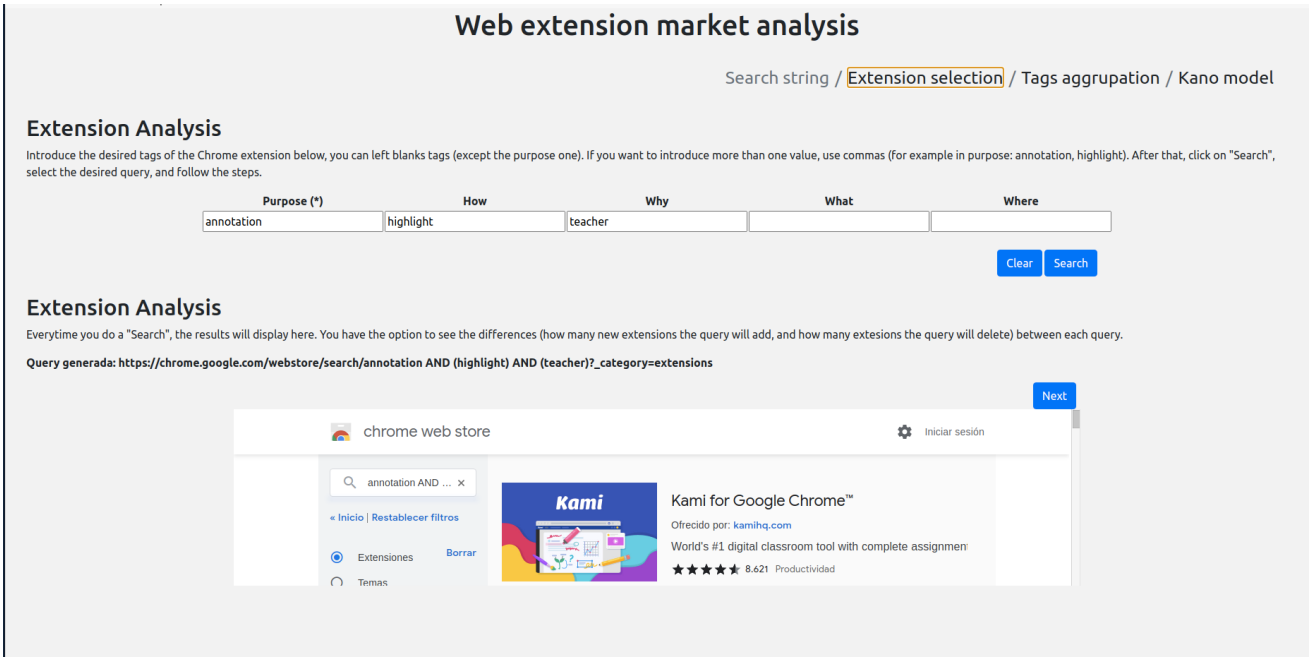

Figura B.1: Resultado con Iframes.

Puede parecer que la solución funciona, y hasta cierto punto es verdad, como se puede observar, carga correctamente la página. Pero esta solución no se puede utilizar por los siguientes motivos:

- No es posible acceder a los elementos dentro del *Iframe* de manera sencilla.
- El *Iframe* no carga las URLs de las extensiones. Es decir, al realizar *scrapping* lo que se necesita en el primer paso tras realizar las búsquedas, es recoger las URls de

las extensiones, mediante las cuales luego se va a acceder a las mismas, pero con este método, este campo no es accesible ni es posible visualizarlo.

Por lo cual, esta solución queda descartada por los motivos descritos previamente.

B.1.2. Solución 2: Uso de browser endpoint

Para esta solución es necesario instalar el paquete de *puppeteer-web* [1], a parte, también se necesita de tener un servidor funcionando, al cual se va a redirigir el funcionamiento de *puppeteer*. Para ello se utiliza la siguiente línea:

Código B.3: Línea de código para cargar el *Iframe*

```
1  const browser = await puppeteer.connect({
2        browserWSEndpoint: `ws://127.0.0.1:8080`, // <-- connect to a server
       running somewhere
3        ignoreHTTPSErrors: true
4      });
```

Realmente emplear esta solución ”solucionaba.[el] problema, ya que daba algún que otro error y problema, a parte que el paquete de *puppeteer-web* llevaba dos años sin recibir actualizaciones, y como era necesario tener un servidor, es más lógico tener un servidor con el servicio definido dentro de él, de esta manera se podría reutilizar el servicio en caso de necesitarlo, entre otras cosas. Por este motivo, esta solución fue descartada y se pasó a la solución final implementada.

B.1.3. Solución 3: Alojarlo en un servidor

Como se ha comentado previamente, esta fue la solución definitiva para arreglar el problema. Para implementar esta solución se utilizó *Express*, y cuando se comprobó su correcto funcionamiento, se incluyo en *Docker* con el resto de los servicios. El servidor programado dispone de tres funcionalidades: recoger las URLs de las extensiones relacionadas a la búsqueda del usuario, extraer la información necesaria de la extensión y extraer los comentarios de la extensión.

Código B.4: Definición de los paquetes necesarios

```
1  const express = require('express');
```

[1] Puppeteer web: https://www.npmjs.com/package/puppeteer-web.

```
const puppeteer = require('puppeteer');
const app = express();

var cors = require('cors');
app.use(cors());

app.use(express.json());

app.listen(4000, () => {
  console.log('listening')
})
```

Se definen los paquetes que van ser utilizados. Tras esto, se incluye el paquete *cors* el cual nos sirve para añadir las cabeceras de *CORS* y que se puedan realizar peticiones entre servidor y aplicación.

Código B.5: Método GET para extraer las URLs de las extensiones

```
app.get('/searchExtensions', async (req, res) => {
  const browser = await puppeteer.launch(
    {
      headless: true // launch headful mode
    }
  );
  console.log('Getting urls from the query: ' + req.query.q);
  const page = await browser.newPage();
  await page.goto(req.query.q);
  await page.waitForTimeout(500);

  // accept Google's conditions / cookies
  await page.waitForSelector('form > div > div > button > span');
  await page.click('form > div > div > button > span');

  let result = []

  await page.waitForSelector('a.h-Ja-d-Ac', {
    visible: true,
    });

  await page.waitForTimeout(500);
  result.push(...await getURLs(page));
```

```
  res.send(JSON.stringify(result));
  await browser.close();
})
```

Lo primero es lanzar *puppeteer*. Tras esto se abre una página de Chrome mediante *puppeteer* y se espera hasta que cargue. Una vez cargada, se aceptan las *cookies*. Tras esto se crea un *array* donde se van a ir introduciendo los resultados. A continuación, se llama a la función asíncrona, ya que para poder utilizar el comando *await* (que sirve para que el programa espere hasta terminar la acción, y de esta manera evitar que se ejecute sin tener los datos cargados), *getURLs()*, la cual recoge las URLs de las extensiones. Una vez terminado esto, se envía un array con todas las URLs de las extensiones.

Código B.6: Función getURLs()

```
async function getURLs(page) {
  return await page.evaluate(() => {

    let urls = [];
    let webs = document.querySelectorAll('.webstore-test-wall-tile');

    // for each web we get the URL
    // same as .map in navigator console
    webs.forEach(web => {
      urls.push(web.querySelector('a.h-Ja-d-Ac').href + '?hl=en');
    });
    return urls;
  })
}
```

Esta función recoge todos los elementos con el identificador *.webstore-test-wall-tile*, es decir, cada extension. Tras esto las itera, y por cada una extrae la URL y la inserta en el *array* creado previamente. Una vez terminado, devuelve dicho *array*. El comando *querySelector* es uno de los más utilizados a la hora de realizar *scrapping*, ya que lo que hace este comando es coger la información contenida en el componente de la página web que tenga el identificador introducido. Es decir, en este caso, dentro de la *web* se selecciona el componente con identificador 'a.h-Ja-d-Ac' y de él, se extrae el atributo *href*. Esto se puede comprobar mediante la función de inspeccionar elementos del navegador.

Figura B.2: Inspeccionar elemento.

Como se puede observar, al introducir el identificador 'a.h-Ja-d-Ac', se nos resalta una línea, la cual hace referencia a una extensión, y mediante el *.href* se extrae dicha URL.

Código B.7: Método POST para extraer la información de las extensiones

```
app.post('/extractExtensionInfo', async (req, res) => {

  let URLs = req.body.query;
  let result = []
  const browser = await puppeteer.launch(
    {
      headless: true // launch headful mode
    }
  );
  let firstTime = true;

  for (let index = 0; index < URLs.length; index++) { // for each URL get
    the info
    console.log('Extension ' + (index + 1) + ' of ' + URLs.length + '.
    Extracting info from: ' + URLs[index]);
    const page = await browser.newPage();

    // await page.goto(req.query.q); // old version
    await page.goto(URLs[index]);

    if (firstTime) {
      await page.waitForSelector('div.VfPpkd-RLmnJb', {
    visible: true,
    });
      // accept Google's conditions / cookies
```

```
      await page.waitForSelector('form > div > div > button > span')
      await page.click('form > div > div > button > span')
      firstTime = false;
    }

    await page.waitForSelector('div.e-f-pa', {
    visible: true,
    });

    result.push(...await getInfo(page));
    await page.waitForTimeout(150);
    await page.close();
  }
  await browser.close();
  res.send(JSON.stringify(result));
})
```

Este método recibe el *array* de las URLs mediante el *body* del *fetch*. En este caso es un *POST*, ya que recibe un JSON mediante el *body*, y en el caso anterior recibia una URL parametrizada. Lo primero es lanzar *puppeteer*. Luego se iteran todas las URLs, y en la primera iteración se aceptan las *cookies* de Chrome. Por cada iteración se abre una pestaña nueva la cual contiene la extensión que toque y se cierra la pestaña abierta anterior. Mediante el comando *waitForSelector*, se espera a que la página esté cargada, y de esta manera se evitan errores, en caso de que la página no cargue y el programa intente acceder a la información no visible todavía. Tras esto, se llama a la función *getInfo()*, la cual extrae la información de la extensión. Para finalizar, se cierra *puppeteer* y se devuelve un *array* con toda la información de las extensiones.

Código B.8: Función getInfo()

```
async function getInfo(page) {
  return await page.evaluate(() => {
    let info = [];

    extensionName = document.querySelector("h1.e-f-w").textContent;

    if (usersTotal = document.querySelector("span.e-f-ih") != null) {
      usersTotal = document.querySelector("span.e-f-ih").getAttribute("
    title").replace(" usuarios", '').replace(" users", '');
    } else {
```

```
    usersTotal = 0;
  }

  // rule of 3: 100% of width = 5 stars
  //            the width that we retrieve = x stars
  if (document.querySelector("div.t9Fs9c") != null) {
    starsString = document.querySelector("div.t9Fs9c").getAttribute("style").replace("width:", '').replace("%", '');
  } else {
    starsString = 0;
  }

  stars = ((parseFloat(starsString) * 5) / 100).toFixed(1);

  if (document.querySelector("pre.C-b-p-j-Oa") != null) {
    description = document.querySelector("pre.C-b-p-j-Oa").textContent.replace(/\n/g, ' ');
  } else {
    description = '';
  }

  if (document.querySelector("span.h-C-b-p-D-md") != null) {
    version = document.querySelector("span.h-C-b-p-D-md").textContent;
  } else {
    version = '';
  }

  if (document.querySelector("span.h-C-b-p-D-xh-hh") != null) {
    lastUpdate = document.querySelector("span.h-C-b-p-D-xh-hh").textContent;
  } else {
    lastUpdate = '';
  }

  info.push({
    "name": extensionName,
    "stars": stars,
    "users": usersTotal,
    "description": description,
    "version": version,
    "lastUpdate": lastUpdate,
```

```
48      });
49      return info;
50    })
51  }
```

En esta función, mediante el uso del *querySelector()* se van extrayendo los distintos elementos informativos de la extensión. Como se puede observar, cada elemento tiene su propio identificador, para saber cuales son estos identificadores, es necesario realizar un trabajo previo mediante la función de inspeccionar elementos y ver que identificador tiene cada elemento deseado. Para las estrellas se utiliza una regla del 3, ya que en la página de la Web Store, estas estrellas aparecen de 0 a 5 estrellas, pero su funcionamiento es una imagen superpuesta que se ensancha o encoge dependiendo de la puntuación que tenga la extensión. Una vez que se han extraído todos los elementos necesarios, se introducen en el *array* creado previamente y se devuelve dicho *array*.

Código B.9: Método POST para extraer los comentarios de las extensiones

```
1  app.post('/extractComments', async (req, res) => {
2
3    let URLs = req.body.query;
4    let result = [];
5    const browser = await puppeteer.launch(
6      {
7        headless: true
8      }
9    );
10
11   let firstTime = true;
12
13   for (let index = 0; index < URLs.length; index++) { // for each URL get
       the info
14     console.log('Extension ' + (index + 1) + ' of ' + URLs.length + '.
       Extracting comments from: ' + URLs[index]);
15     const page = await browser.newPage();
16
17     await page.goto(URLs[index]);
18
19     // accept Google's conditions / cookies
20     if (firstTime) {
21       await page.waitForSelector('div.VfPpkd-RLmnJb', {
```

```
22      visible: true,
23      });
24        // accept Google's conditions / cookies
25        await page.waitForSelector('form > div > div > button > span')
26        await page.click('form > div > div > button > span')
27        firstTime = false;
28      }
29
30      await page.waitForSelector('h1.e-f-w', {
31      visible: true,
32      });
33
34      let hasNext = true;
35      let maxPages = 3;
36      var comments2 = [];
37      var comments = {};
38
39      while (hasNext && maxPages > 0) {
40        comments = await extractComments(page);
41        comments2 = comments2.concat(comments);
42        hasNext = await page.evaluate(() => {
43          if (document.querySelector("body  a.dc-se").getAttribute("style") !
      = "display: none;") {
44            document.querySelector("body  a.dc-se").click();
45            return true;
46          }
47          return false;
48        });
49        await page.waitForTimeout(1050);
50        maxPages--;
51      }
52      if (comments2.length > 0) {
53        result.push(comments2);
54      }
55      await page.close();
56    }
57    await browser.close();
58    res.send(JSON.stringify(result));
59  })
```

Se extrae el *array* que contiene las URLs de las extensiones. Tras esto, se lanza *puppeteer*

y por cada URL se abre una nueva pestaña y se cierra la anterior, de la misma manera que en el método anterior. En caso de ser la primera iteración, se aceptan las *cookies*. Tras esto, se define un bucle, el cual va a iterar por cada página de comentarios de la extensión (caben un total de 25 por cada página). Mediante *getAttribute("style")* se comprueba que existan más páginas de comentarios, en caso de existir, se abre la siguiente, y se extraen los comentarios de dicha página. Una vez terminado con todas las extensiones, se devuelve el *array* con todos los comentarios extraídos.

Código B.10: Función *extractComments()*

```
async function extractComments(page) {
  return await page.evaluate(() => {
    let messages = [];
    let comments = document.querySelectorAll("body    div.ba-pa")

    // 25 of them, might be empty
    comments.forEach(comment => {
      if (comment.querySelector(".ba-Eb-ba") != null) {
        if (comment.querySelector(".rsw-stars") != null) {
          stars = comment.querySelector(".rsw-stars").ariaLabel.replace("
    stars", "").replace(" estrellas", "").replace(" star", "").replace("
    estrella", "");
        }
        text = comment.querySelector(".ba-Eb-ba").innerText;
        author = comment.querySelector(".comment-thread-displayname").
    innerText;
        date = comment.querySelector(".ba-Eb-Nf").innerText.replace("
    Modified ", "");

        messages.push({
          "author": author,
          "date": date,
          "stars": stars,
          "text": text,
        });
      }
    });
    return messages;
  })
}
```

Mediante *querySelectorAll("body div.ba-pa")* se recogen todos los comentarios, ya que el identificador de los comentarios es 'div.ba-pa'. Se itera por cada comentario extraído, y usando el *querySelector()* (conociendo previamente los identificadores de los valores que se desean conseguir) se guardan los valores de: el texto, el autor, la fecha y las estrellas (de las cuales solo se guarda el valor y se retira el texto). Una vez hecho esto, se guarda dichos valores en el *array* creado previamente, y una vez terminado con los comentarios, se devuelve el *array*.

B.2. Funcionamiento de los otros servicios

B.2.1. Servicio de detección de características

Como ya se ha mencionado previamente, este servicio ya estaba previamente definido. Este servicio recibe un *array* de comentarios mediante el método *POST* del *fetch*. Estos comentarios tienen el siguiente formato:

Código B.11: Formato comentarios

```
{
    'author' : 'Pepe',
    'review_id': 'ri',
    'package_name' : 'extension',
    'rating' : 3.5,
    'title' : '',
    'body' : 'I love this extension because I can save the result on Drive'
}
```

Y tras realizar el análisis de dichos comentarios, usando IA y estudios de palabras, es capaz de detectar que comentarios son positivos y cuales negativos. Una vez terminado este proceso, devuelve un *array* con el siguiente formato:

Código B.12: Formato comentarios

```
{
    'author' : 'Pepe',
    'body' : 'I love this extension because I can save the result on Drive'
    ,
    'cluster_is_feature_request' : true,
    'cluster_is_bug_report' : false,
```

```
    'package_name' : 'extension',
    'rating' : 3.5,
    'review_id': 'ri',
    'title' : ''
}
```

Por lo cual, este comentario sería utilizado para extraer sus palabras clave, ya que está aportando información sobre la extensión.

B.2.2. Servicio de extracción de palabras clave

Como ya se ha mencionado previamente, este servicio ya estaba previamente definido. Este servicio recibe un *array* de comentarios mediante el método *POST* del *fetch*. Estos comentarios tienen el siguiente formato:

Código B.13: Formato comentarios

```
{
    'author' : 'Pepe',
    'date': '26 Sep 2021',
    'stars' : 3.5,
    'text' : 'I love this extension because I can save the result on Drive'
}
```

Este servicio devuelve un *array* con las palabras clave de los comentarios extraídos, en caso del ejemplo anterior devolvería ”save on Drive”.

B.2.3. Cors-anywhere

Debido a una cabecera HTTP llamada CORS [2], la cual sirve para evitar el acceso a los dominios que sean distintos al origen. El problema es que estas cabeceras estaban evitando el poder recibir la respuesta de los servicios (a excepción del de *scrapping* por lo ya explicado previamente), por lo cual fue necesario del uso de un servidor proxy que redirige la solicitud de la aplicación al servicio necesario, añadiéndole la cabecera del CORS. Estas cabeceras se pueden añadir de forma manual, pero aún haciéndolo de esta manera, no era posible evitar el error, por lo cual se decidió utilizar el servidor proxy. Este servidor

[2]CORS: https://developer.mozilla.org/es/docs/Web/HTTP/CORS.

proxy ya estaba definido en GitHub [3], por lo cual no fue necesario implementarlo, pero si incluirlo en el *Docker*.

B.3. Dockerfiles y docker compose para los servicios

Como ya se ha mencionado en el Capítulo 4, para el despliegue de los servicios se ha utilizado *Docker* junto a *Docker compose*, para poder utilizar estas herramientas, es necesario definir unos ficheros, que se encargan de cargar todo lo necesario para que los servicios puedan ser utilizados.

Código B.14: *Dockerfile* del servicio *cors-anywhere*

```
1  FROM node:10.11-alpine
2
3  WORKDIR /usr/src/app
4  COPY package*.json ./
5  RUN npm install
6
7  COPY . .
8
9  EXPOSE 5000
10
11 CMD [ "node", "server.js" ]
```

Este *Dockerfile* es el encargado de ejecutar los comandos necesarios para desplegar el servicio del *cors-anywhere*. La primera línea le dice a *Docker* que versión de *Node* utilizar. Tras esto, se define un espacio de trabajo y se copia el fichero de *package.json* (el que contiene las dependencias del servidor entre otras cosas) a dicho directorio, y mediante el comando *RUN* se ejecuta *npm install*, el cual instala todas las dependencias. Una vez hecho esto, se abre el puerto 5000 y se inicia el servicio.

Código B.15: *Dockerfile* del servicio de *scrapping*

```
1  FROM node:12-slim
2
3  # Install latest chrome dev package and fonts to support major charsets (
       Chinese, Japanese, Arabic, Hebrew, Thai and a few others)
4  # Note: this installs the necessary libs to make the bundled version of
       Chromium that Puppeteer
```

[3] Cors-anywhere: https://github.com/Rob--W/cors-anywhere.

```
# installs, work.
RUN apt-get update \
    && apt-get install -y wget gnupg \
    && wget -q -O - https://dl-ssl.google.com/linux/linux_signing_key.pub | apt-key add - \
    && sh -c 'echo "deb [arch=amd64] http://dl.google.com/linux/chrome/deb/ stable main" >> /etc/apt/sources.list.d/google.list' \
    && apt-get update \
    && apt-get install -y google-chrome-stable fonts-ipafont-gothic fonts-wqy-zenhei fonts-thai-tlwg fonts-kacst fonts-freefont-ttf libxss1 \
      --no-install-recommends \
    && rm -rf /var/lib/apt/lists/*

# If running Docker >= 1.13.0 use docker run's --init arg to reap zombie processes, otherwise
# uncomment the following lines to have `dumb-init` as PID 1
# ADD https://github.com/Yelp/dumb-init/releases/download/v1.2.2/dumb-init_1.2.2_x86_64 /usr/local/bin/dumb-init
# RUN chmod +x /usr/local/bin/dumb-init
# ENTRYPOINT ["dumb-init", "--"]

# Uncomment to skip the chromium download when installing puppeteer. If you do,
# you'll need to launch puppeteer with:
#     browser.launch({executablePath: 'google-chrome-stable'})
# ENV PUPPETEER_SKIP_CHROMIUM_DOWNLOAD true

WORKDIR /usr/app
COPY index.js .

# Install puppeteer so it's available in the container.
RUN npm init -y &&  \
    npm i puppeteer && \
    npm i express && \
    npm i cors && \
    npm i -g nodemon \
    # Add user so we don't need --no-sandbox.
    # same layer as npm install to keep re-chowned files from using up several hundred MBs more space
    && groupadd -r pptruser && useradd -r -g pptruser -G audio,video
```

```
    pptruser \
38      && mkdir -p /home/pptruser/Downloads \
39      && chown -R pptruser:pptruser /home/pptruser \
40      && chown -R pptruser:pptruser ./node_modules \
41      && chown -R pptruser:pptruser ./package.json \
42      && chown -R pptruser:pptruser ./package-lock.json
43
44  # Run everything after as non-privileged user.
45  USER pptruser
46
47  CMD ["google-chrome-stable"]
```

Para poder ejecutar *puppeteer* dentro de *Docker*, había que definir un *Dockerfile* específico el cual está definido en el GitHub de *Docker* [4]. Básicamente este fichero, carga una versión de Chrome, en la cual se ejecutará *puppeteer*.

Código B.16: *Dockerfile* del servicio de extracción de palabras clave

```
1   FROM rstudio/plumber
2   MAINTAINER Docker User <docker@user.org>
3
4   RUN apt-get update --allow-releaseinfo-change && apt-get install -y
        libz-dev
5   RUN R -e "install.packages(c('tidyverse','dplyr','udpipe','jsonlite'), \
6   repos = 'http://cran.us.r-project.org')"
7   RUN R -e "udpipe::udpipe_download_model('english')"
8
9   COPY ./getTopics.R /plumber.R
10
11  CMD ["/plumber.R"]
```

Este *Dockerfile* instala las dependencias y paquetes necesarios para este servicio. Una vez instalados los paquetes necesarios, se inicia el servicio. En este caso no es necesario abrir el puerto, ya que este se define en el código del servicio.

Código B.17: *Dockerfile* del servicio de detección de características

```
1   FROM sellpy/python3-jupyter-sklearn-java
2
```

[4] GitHub de problemas de Docker: https://github.com/puppeteer/puppeteer/blob/main/docs/troubleshooting.md.

```
RUN pip3 install --upgrade pip \
    Flask==1.0.2 \
    pandas==0.23.4 \
    numpy==1.15.4 \
    scikit_learn==0.20.1 \
    gdown

RUN pip3 install nltk

RUN pip3 install gdown
RUN apt-get install unzip

ARG GDRIVE_DL_LINK

RUN gdown https://drive.google.com/uc?id=${GDRIVE_DL_LINK}
RUN unzip -d . stanford-postagger-full-2016-10-31.zip

RUN [ "python3", "-c", "import nltk; nltk.download('all')" ]

# Add local files and folders
ADD / /app/amazon-kinesis-client-python/

EXPOSE 9651

CMD [ "python3", "./starter.py" ]
```

Este *Dockerfile* instala las dependencias y paquetes necesarios para este servicio. Una vez instalados los paquetes necesarios, se descarga desde Drive el .zip necesario para ejecutar este servicio, y abre el puerto necesario. Tras todo el proceso de instalación, se inicia el servicio.

Código B.18: Fichero de *Docker compose*

```
version: '3'

services:
  topic-extraction:
    build: ./topicextraction-main
    ports:
      - "8000:8000"
  feature-detection:
```

```
    build: ./ri-analytics-classification-google-play-review-master
    ports:
      - "9651:9651"
  scraping-service:
    build: ./scraping-service-chrome-extension-analysis
    ports:
      - "4000:4000"
    cap_add:
      - SYS_ADMIN
    init: true
    command: nodemon
  cors-anywhere:
    build: ./cors-anywhere-service
    ports:
      - "5000:5000"
```

El fichero de *Docker compose* comienza definiendo la versión de dicho programa. Tras esto, se define *services:*, que van a ser los servicios que se van a desplegar. Por cada servicio se define la ruta al *Dockerfile* mediante la línea de *build*. Tras esto, con la línea de *ports* se define el mapeado de los puertos, en este caso se asigna el mismo puerto real y virtual. En el caso del servicio de *scrapping* ha sido necesario utilizar *nodemon*, mediante el cual se carga el *index.js*, que es el que contiene el servicio. Esto se debe a que el *Dockerfile* del servicio de *scrapping* carga lo necesario para poder ejecutar *puppeteer*, pero hay que decirle al contenedor que código necesita ejecutar.

C. ANEXO

Actas de reuniones

PLANTILLA DE ACTAS

Fecha:
Inicio:
Fin:
Lugar: Reunión presencial
Tipo de Reunión: Seguimiento y control
Asistentes:
Unai Ahedo de Luis
Oscar Díaz García

Orden del día

1.

Resumen de la reunión

Estado del proyecto

Decisiones

- **Decisiones adoptadas:**
  -
- **Asignación de tareas a realizar:**
  -
- **Próxima reunión:**

ACTA 1

Fecha: 08-09-2021
Inicio: 12:30
Fin: 13:15
Lugar: Reunión telemática
Tipo de Reunión: Primera reunión
Asistentes:
Unai Ahedo de Luis
Oscar Díaz García

Orden del día

1. Iniciar el TFG.
2. Definir el alcance del TFG.
3. Definir los requisitos del TFG.

Resumen de la reunión

Mediante unas presentaciones, Oscar le ha propuesto a Unai la realización de una aplicación de escritorio, la cual sirva para realizar un análisis de mercado de las extensiones de Google Chrome. Este análisis de mercado servirá para futuros desarrolladores de extensiones a la hora de escoger las mejores características para sus extensiones. Para esto se utilizará el modelo Kano para categorizar las características necesarias mediante las cuales se contente a los posibles futuros usuarios de la extensión. Durante la reunión se ha propuesto un alcance concreto, con opción a extenderlo en caso de ser posible.

Estado del proyecto

El proyecto está en su estado inicial. Se dispone de la idea a desarrollar, y las herramientas a utilizar.

Decisiones

- **Decisiones adoptadas:**
  - El ciclo de vida del proyecto será incremental.
  - La herramienta a utilizar para realizar el mockup queda a elección del alumno.
  - Para realizar la aplicación de escritorio, el alumno utilizará Electron y JavaScript.

  - Debido a que la Workshop de Google Chrome no tiene API, el alumno utilizará web scraping para conseguir los datos necesarios a la hora de realizar el análisis de mercado.
  - Para el diseño de la interfaz el alumno utilizará *Cognitive Walkthrough*.

- **Asignación de tareas a realizar:**
  - Familiarización de las tecnologías a utilizar. (Unai)
  - Buscar información sobre *Cognitive Walkthrough*. (Unai)
  - Realizar un *mockup* de la interfaz de la aplicación. (Unai)
  - Realizar una presentación para la próxima reunión. (Unai)
- **Próxima reunión:** Por definir.

ACTA 2

Fecha: 17-09-2021
Inicio: 13:00
Fin: 13:30
Lugar: Reunión presencial
Tipo de Reunión: Seguimiento y control
Asistentes:
Unai Ahedo de Luis
Oscar Díaz García
Haritz Medina Camacho

Orden del día

1. Presentación por parte de Unai del boceto inicial de las interfaces.
2. Propuesta por parte de Unai de la planificación inicial.
3. Indicar las siguientes tareas a seguir en el proyecto.

Resumen de la reunión

La reunión ha comenzado con Unai mostrando el boceto inicial hecho mediante Mockplus de las interfaces. Tanto Oscar como Haritz se han percatado de que faltaban funcionalidades, las cuales Unai pensaba que se debían hacer de manera automática, pero que deben quedar en manos del usuario. Es decir, la agrupación de los *tags* asociados a una extensión, la debe hacer el usuario, y de esa manera generar *features*. Sumado a esto Oscar ha propuesto mostrar las extensiones (a la hora de seleccionar las deseadas) utilizando un estilo parecido al visualizador de fotos de los Mac.

Tras esto Unai ha presentado el diagrama de Gantt asociado al proyecto. A esto último Oscar ha añadido que sería necesario tener para mediados de Octubre un prototipo de la aplicación.

Estado del proyecto

El proyecto sigue en una etapa inicial, aunque ya se dispone de una idea a seguir en cuanto a las interfaces.

Decisiones

- **Decisiones adoptadas:**

  - Hay que añadir las funcionalidades que va a utilizar el usuario en las interfaces.
  - Deberá haber una primera versión de la aplicación para mediados de Octubre, las interfaces en Electron e implementación de scripts.
  - Unai debe modificar la planificación para incluir la entrega del primer prototipo.

- **Asignación de tareas a realizar:**
  - Modificar el diseño de las interfaces, para que el usuario tenga la opción de agrupar *tags* y crear las *features* a raíz de ello.(Unai)
  - Modificar el diseño de las interfaces, para que el usuario pueda categorizar las *features* dentro del modelo de Kano. (Unai)
  - Haritz resolverá las dudas de Unai sobre los scripts que van a servir de punto de partida para el proyecto.
- **Próxima reunión:** Lunes 20-09-2021.

ACTA 3

Fecha: 20-09-2021
Inicio: 15:00
Fin: 15:15
Lugar: Reunión telemática
Tipo de Reunión: Seguimiento y control
Asistentes:
Unai Ahedo de Luis
Oscar Díaz García
Haritz Medina Camacho

Orden del día

1. Presentación por parte de Unai de los arreglos de las interfaces iniciales.
2. Indicar las pautas a seguir en el proyecto.

Resumen de la reunión

La reunión ha comenzado con Unai mostrando las modificaciones realizadas a los bocetos previamente presentados. Tanto Oscar como Haritz han dado el visto bueno, y ha dado comienzo la fase de desarrollo de la aplicación.

Estado del proyecto

El proyecto sigue en una etapa inicial, aunque ya se dispone de una idea a seguir en cuanto a las interfaces. Se va a comenzar el desarrollo de la aplicación mediante el uso de Electron + React.

Decisiones

- **Decisiones adoptadas:**
  - Unai comenzará a familiarizarse con Electron, Node y React.
  - Unai comenzará a desarrollar el prototipo.
- **Asignación de tareas a realizar:**
  - Unai deberá tener un prototipo con unas funcionalidades básicas para mediados de octubre.
- **Próxima reunión:** Mediados de octubre, cuando Unai termine el primer prototipo.

ACTA 4

Fecha: 14-10-2021
Inicio: 16:00
Fin: 16:15
Lugar: Reunión telemática
Tipo de Reunión: Presentación prototipo 1
Asistentes:
Unai Ahedo de Luis
Oscar Díaz García
Haritz Medina Camacho

Orden del día

1. Presentación por parte de Unai del prototipo y problemas asociados al mismo.
2. Indicar las pautas a seguir en el proyecto.

Resumen de la reunión

La reunión ha comenzado con Unai mostrando el prototipo de la aplicación, la cual ya dispone de los scripts necesarios para hacer el scrapping mediante el cual se van a sacar los datos para crear el modelo Kano. Estos scripts todavía no se han podido integrar en la aplicación, debido a que puppeteer tiene que ser ejecutado en un servidor, y la aplicación es solo de cliente. Oscar ha dado su aprobación al prototipo, aunque ha comentado que se debe encontrar una solución al problema con puppeteer.

Tras esto, Unai y Haritz han estado buscando y debatiendo alternativas posibles a puppeteer, u otra manera de utilizarlo con la que no hayan problemas.

Estado del proyecto

El proyecto se encuentra en la entrega del primer prototipo, el cual cumple las funciones básicas necesarias para el mismo.

Decisiones

- **Decisiones adoptadas:**
  - Para arreglar el problema de puppeteer se han propuesto dos soluciones: disponer de los scripts que se ejecutan con puppeteer como un servicio o utilizar IFrames.
- **Asignación de tareas a realizar:**

  - Unai y Haritz comprobarán la viabilidad de realizar la parte asociada a puppeteer de otra manera.
  - Unai implementará la solución al prototipo y comprobará su correcto funcionamiento.

- **Próxima reunión:** Por determinar.

ACTA 5

Fecha: 05-11-2021
Inicio: 13:00
Fin: 13:20
Lugar: Reunión telemática
Tipo de Reunión: Seguimiento y control
Asistentes:
Unai Ahedo de Luis
Oscar Díaz García

Orden del día

1. Presentación por parte de Unai de las implementaciones realizadas para poder utilizar puppeteer en la aplicación.
2. Indicar las pautas a seguir en el proyecto.

Resumen de la reunión

Unai ha presentado la solución a los problemas previos con puppeteer, solo que no está optimizada. Oscar ha solicitado que se debe optimizar para que tarde lo menos posible al realizar el scrapping. Tras esto, Oscar ha sugerido que se implemente un mockup sobre los dos últimos apartados de la aplicación, aunque no se disponga de la funcionalidad de hacer el scrapping, que estos mockups sirvan para visualizar las acciones que va a poder llevar a cabo el usuario.

Estado del proyecto

El proyecto se encuentra en una fase media. Ya ha alcanzado la mitad de sus funcionalidades, aunque algunas de estas se deban optimizar.

Decisiones

- **Decisiones adoptadas:**
  - Se debe optimizar el scrapping.
  - Se debe realizar un mockup del apartado de agrupación de tags y el modelo Kano
- **Asignación de tareas a realizar:**
  - Unai debe optimizar el apartado del scrapping.

- Unai debe implementar un mockup de los apartados de la agrupación de tags y el modelo Kano.

- **Próxima reunión:** 15-11-2021

ACTA 6

Fecha: 15-11-2021
Inicio: 13:00
Fin: 13:30
Lugar: Reunión telemática
Tipo de Reunión: Seguimiento y control
Asistentes:
Unai Ahedo de Luis
Oscar Díaz García

Orden del día

1. Mostrar los cambios realizados desde la reunión previa.

Resumen de la reunión

La reunión ha comenzado con Unai mostrando los cambios realizados desde la última reunión los cuales incluían: optimizar la búsqueda de extensiones (ahora tarda minuto y medio en buscar 50 extensiones, en comparación a los 10 minutos previos) e integrar la funcionalidad, por ahora solo visual, del apartado de agrupar tags y clasificarlos. Estos cambios han recibido el visto bueno por parte de Oscar, aunque ha hecho un apunte sobre los cloud tags: el color de las palabras dentro de los mismos también deberían ser representativos, es decir, que si los tags provienen de comentarios positivos, que estos dentro del cloud tag sean de color verde. Una vez vistos todos los cambios, Oscar le ha comentado a Unai que sería interesante que se reuniera con Juanan, el cual ha desarrollado el apartado que falta en torno a la funcionalidad, para integrarlo en la aplicación.

Estado del proyecto

A falta de implementar una sección para terminar la implementación de la aplicación.

Decisiones

- **Decisiones adoptadas:**
  - Se concretará una próxima reunión con Juanan, en la cual se hablará sobre la sección del código que se encargará de extraer los tags de los comentarios.
  - Buscar y utilizar librerías de cloud tags que permitan modificar el color de las palabra contenidas en el mismo.
- **Asignación de tareas a realizar:**

- Mejorar las GUI visualmente. (Unai)
- Modificar el tags cloud para que aporte más información (el color de las palabras tiene que ser significativo también, no solo el tamaño). (Unai)
- Reunirse con Juanan e integrar el apartado que se encargará de extraer los tags. (Unai).

- **Próxima reunión:** Por determinar.

ACTA 7

Fecha: 03-12-2021
Inicio: 12:00
Fin: 12:30
Lugar: Reunión presencial
Tipo de Reunión: Seguimiento y control
Asistentes:
Unai Ahedo de Luis
Oscar Díaz García

Orden del día

1. Mostrar los cambios realizados desde la reunión previa.

Resumen de la reunión

La reunión ha comenzado con Unai mostrando algunos de los cambios realizados desde la última reunión, de los cambios que de debían realizar se han podido realizar únicamente los siguientes: implementar el apartado que extrae los tags de las extensiones, y mejorar las GUI visualmente. No se ha podido modificar el cloud tag ya que Unai no ha encontrado ninguna librería que permita mostrar el cloud tag.

Estado del proyecto

A falta de dockerizar los servicios en un mismo endpoint y mejoras en la optimización de la aplicación. Comienzo de pruebas funcionales y con usuarios.

Decisiones

- **Decisiones adoptadas:**
  - Se deben dockerizar los servicios para que sea posible exportarlos y utilizarlos fuera de esta aplicación.
  - Se debe crear el ejecutable de la aplicación para poder comenzar con las pruebas con usuarios.
- **Asignación de tareas a realizar:**
  - Optimizar la aplicación. (Unai)
  - Dockerizar los servicios. (Unai)
  - Crear el ejecutable de la aplicación. (Unai)
- **Próxima reunión:** Por determinar.

D. ANEXO

Guía de uso

Se incluye una guía de uso del programa, para que sea más sencillo utilizarlo desde 0 sin conocimiento previo del mismo.

San Sebastián / Donostia

Guía de Uso

"***Aplicación*** **para análisis de mercado de extensiones de Google Chrome**"

Unai Ahedo de Luis

Universidad del País Vasco
Facultad de Ingeniería Informática
Ingeniería del Software

Índice

Lista de Figuras

1. Guía

1.1 Paso 0: Configurar servicios

1.1.1 Fuera de la aplicación

El paso previo a utilizar la aplicación es el despliegue de los servicios, para ello es necesario visitar el siguiente enlace de GitHub [1], que es donde se encuentran los servicios. Es necesario descargar y descomprimir el proyecto. Hay que instalar *Docker* [2] y *Docker compose* [3] Tras esto, se accede mediante consola a la carpeta extraída y se utilizan los siguientes comandos en orden:

```
docker-compose build --build-arg GDRIVE_DL_LINK=1GA2Idb9LhpepGwGIRCiqoPok6mYWWE3M
```

```
docker-compose up
```

Al ejecutar el primer comando, en la consola empezará a aparecer texto y comandos distintos, esto es parte de la configuración y puede llevar un rato, en torno a los 15 minutos (dependiendo de la máquina donde se ejecute), por lo cual hay que dejarlo funcionar. Cuando se configure, es necesario ejecutar el segundo comando, el cual tardará menos en funcionar y debería de mostrar lo siguiente por pantalla:

[1] GitHub de servicios: https://github.com/UnaiAhedo/docker-compose-chrome-extension-analysis-application.
[2] Instalar docker: https://docs.docker.com/engine/install/.
[3] Instalar docker compose: https://docs.docker.com/compose/install/.

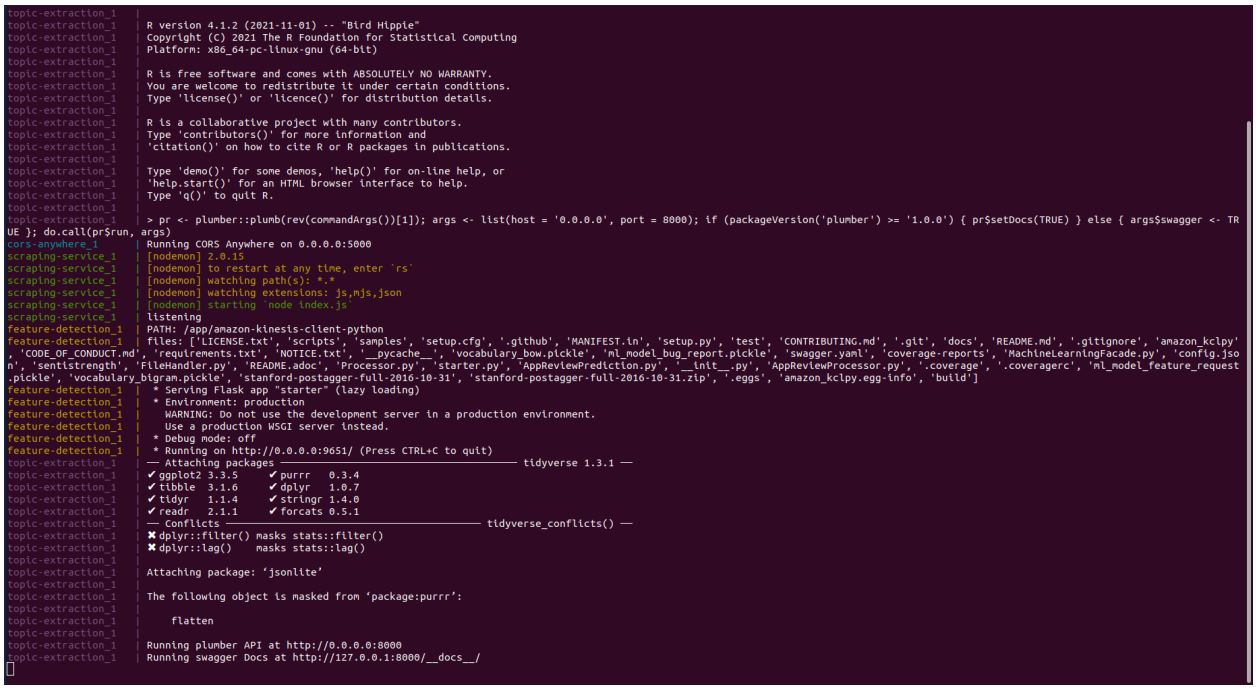

Figure 1.1: Consola tras docker compose.

Como se puede observar a la izquierda, hay textos de varios colores, haciendo referencia a los diferentes servicios (cors, scrapping...).

1.1.2 En la aplicación

Una vez desplegados los servicios, es necesario configurar las IPs dentro de la aplicación, para esto hay que visitar el siguiente enlace [4], que es el que contiene el ejecutable de la aplicación, y descargarla. Una vez hecho esto, se extrae y se ejecuta (mediante ./ en linux) el programa *extension-market-analysis*, el cual se encuentra en *dist/linux-unpacked*. Esto abrirá la aplicación. En esta primera pantalla, hay un apartado de configuración, al cual se accede presionando al botón de *Introduce IPs*.

[4]Drive de aplicación: https://drive.google.com/drive/folders/1T-yDY42Y7wKhkqAk5ZnyYkUSeievVg2K.

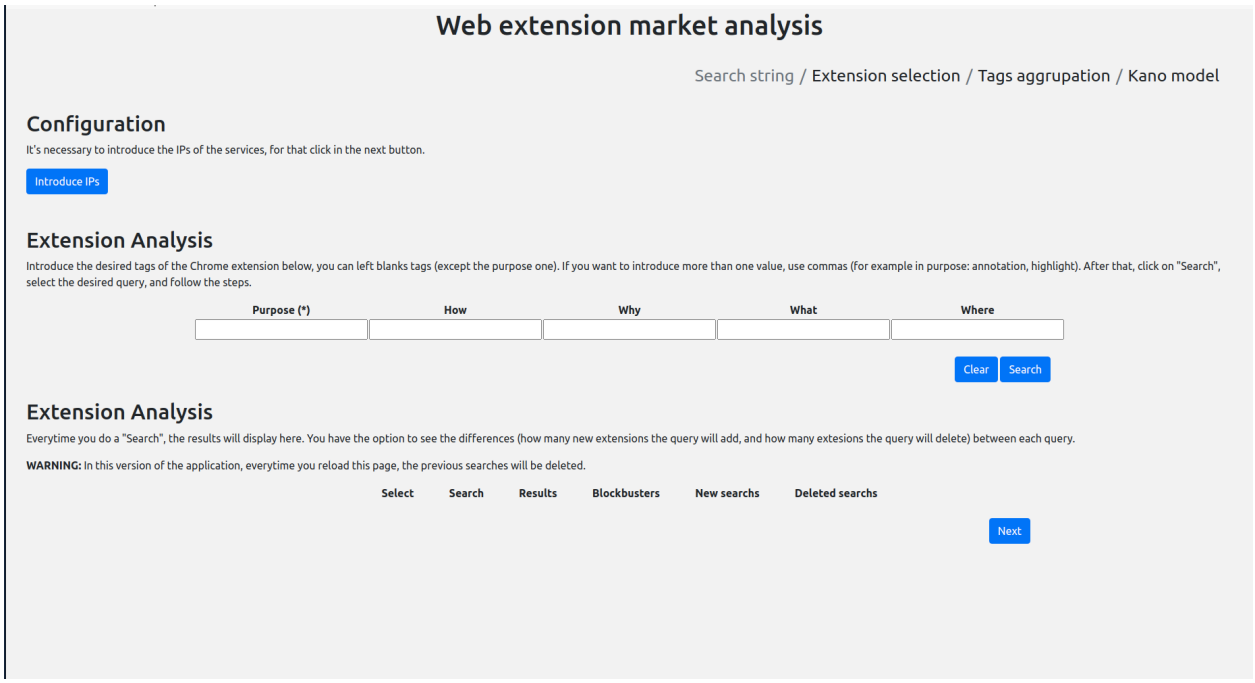

Figure 1.2: Primera página.

En ese menú se encuentra lo siguiente:

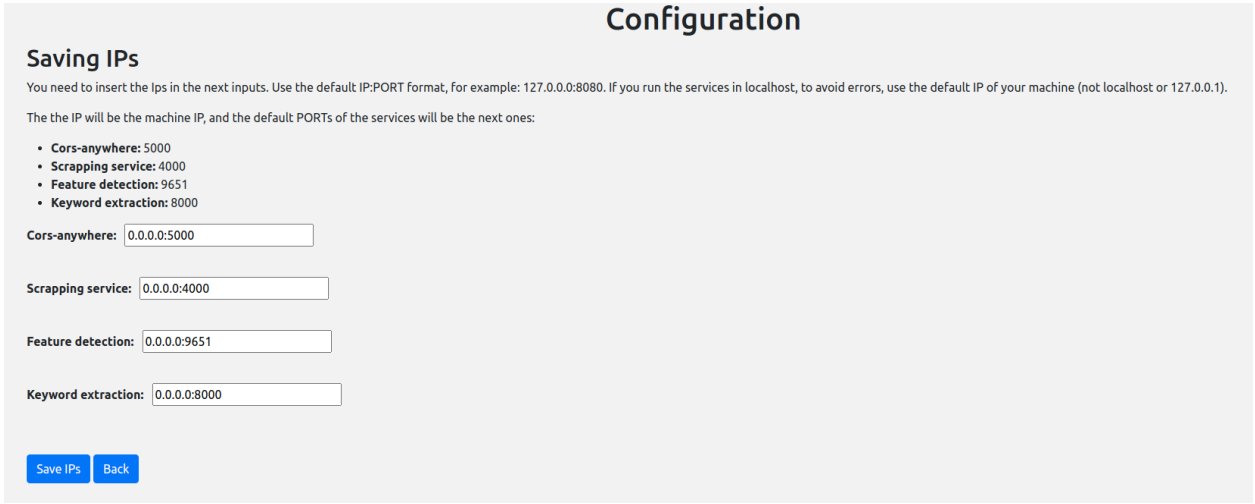

Figure 1.3: Apartado configuración.

Aquí se introducen las IPs de donde están desplegados los servicios, los cuales tienen unos puertos por defecto ya apuntados en la misma aplicación. Los puertos por defecto de dichos servicios ya están puestos en los *inputs*, en caso de cambiar los puertos por defecto del *docker-compose*, también será necesario cambiarlos de aquí. Para evitar cualquier tipo de error, es necesario introducir la IP privada de la máquina, aunque se esté ejecutando en *localhost*. Con las IPs configuradas, ya es posible empezar a utilizar el programa.

1.2 Paso 1: Realizar la búsqueda y elegir la deseada

En la primera página se dispone de los elementos de búsqueda, mediante los cuales se podrán encontrar las distintas extensiones deseadas.

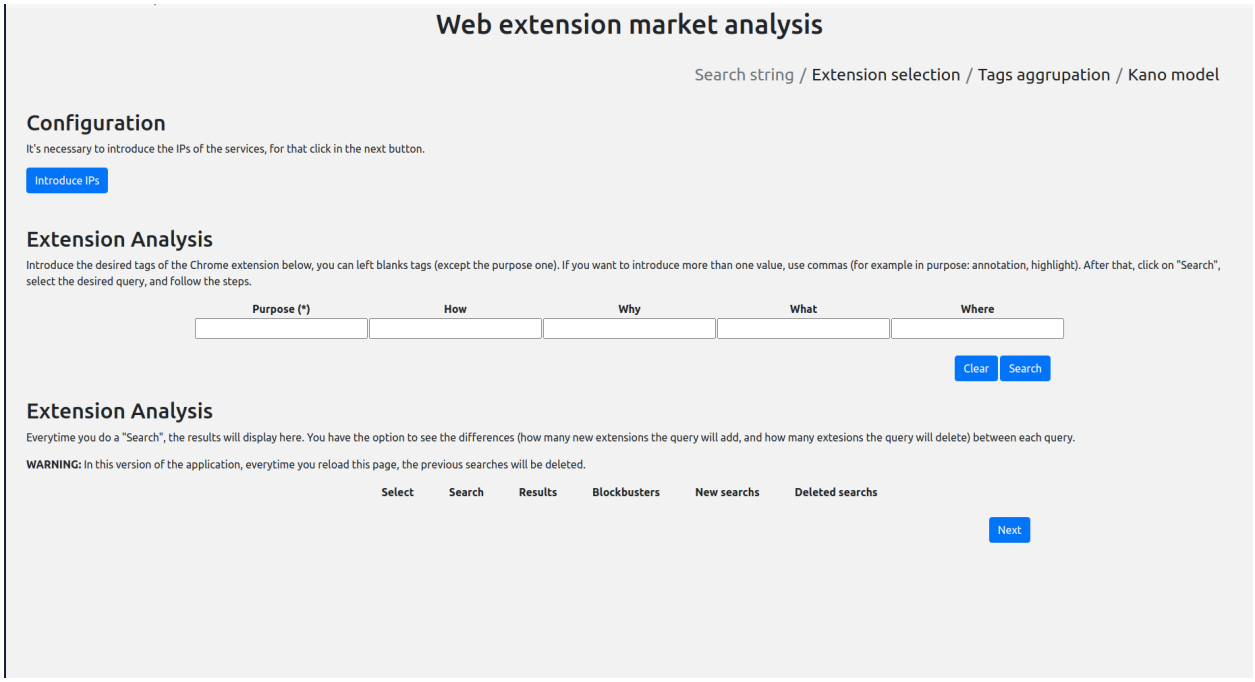

Figure 1.4: Primera página.

Para esto, es necesario introducir un *purpose* sobre lo que tratará la búsqueda, y algún campo más por si fuera necesario sesgar más aún dicha búsqueda. Al darle a buscar, la aplicación buscará dentro del mercado de extensiones de Google Chrome las que tengan que ver con dicha búsqueda.

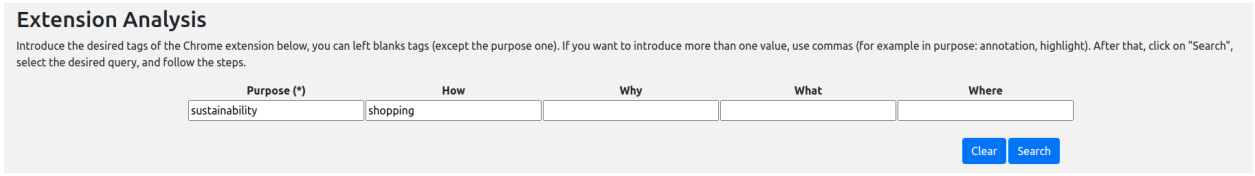

Figure 1.5: Apartado de búsqueda.

Cuando la búsqueda ha finalizado, se mostrará el resultado en el apartado inferior, por lo cual, se pueden realizar varias búsquedas, y escoger la que más convenza al usuario.

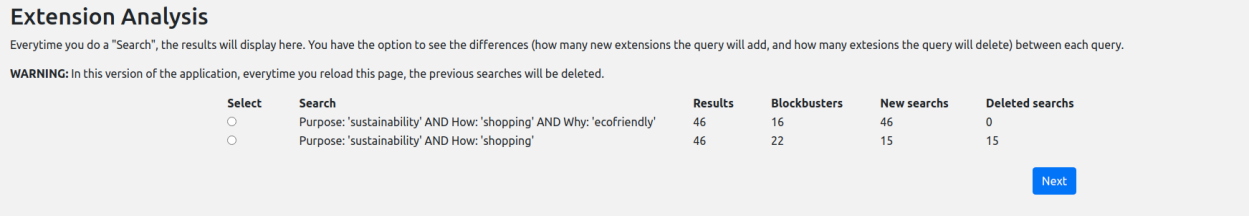

Figure 1.6: Apartado de resultados.

1.3 Paso 2: Elegir las extensiones deseadas

Una vez escogida la búsqueda deseada y pulsado el botón *Next*, se llegará a la página de escoger las extensiones deseadas. Para facilitar la decisión al usuario, se podrán visualizar las descargas, usuarios totales... Entre otras cosas, de esta manera el usuario podrá escoger mejor las extensiones deseadas. Para seleccionarlas, tan solo hay que pulsar la casilla a la izquierda de la extensión deseada.

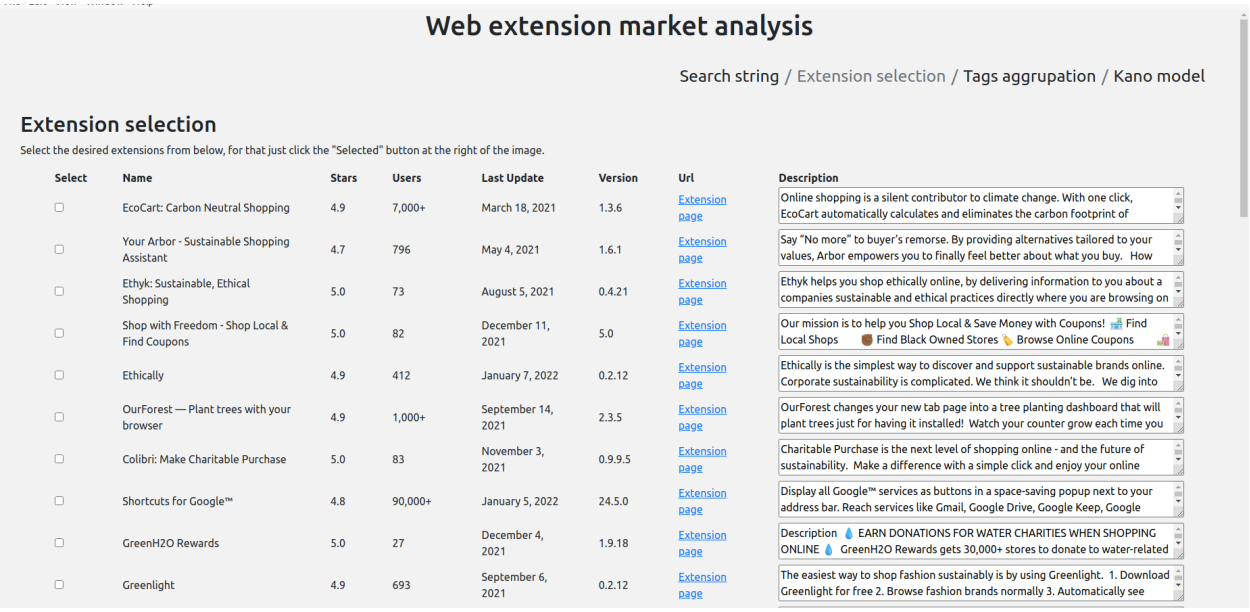

Figure 1.7: Escoger extensiones.

Una vez finalizada la selección, en la parte inferior de la página se encuentra el botón de *Next*.

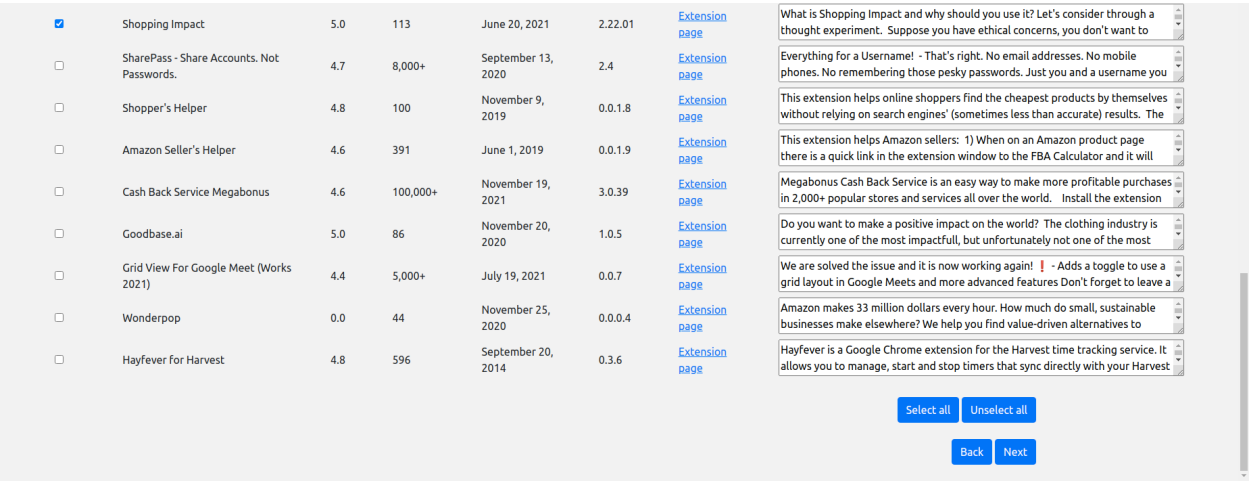

Figure 1.8: Escoger extensiones parte inferior.

1.4 Paso 3: Agrupar los tags

Una vez cargada esta página, se visualizará lo siguiente:

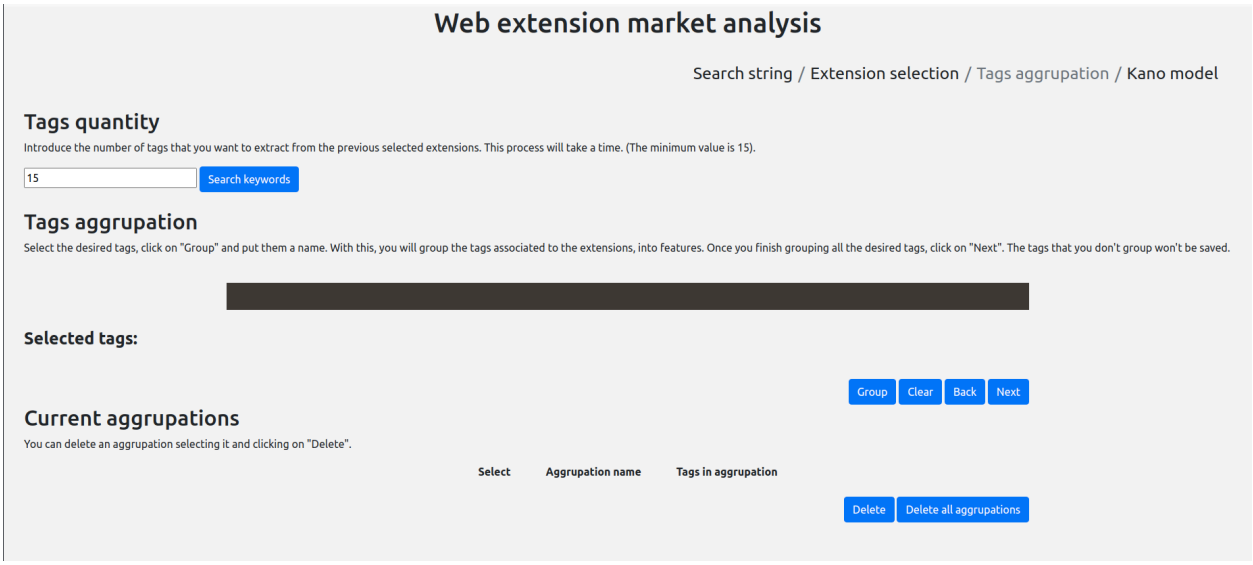

Figure 1.9: Cargar las palabras clave.

Para que el programa comience a extraer las palabras clave de los comentarios de las extensiones, es necesario introducir un número, el cual va a ser cuantas palabras clave o *tags* se van a extraer, y luego pulsar el botón *Search keywords*. **AVISO:** este proceso puede tardar, a más extensiones escogidas en el paso anterior, mayor será la carga, así que es necesario tener paciencia. Cabe destacar que este sistema de extracción de *keywords* pueda dar algunas palabras clave que no sean excesivamente significativas para el dominio que estés analizando.

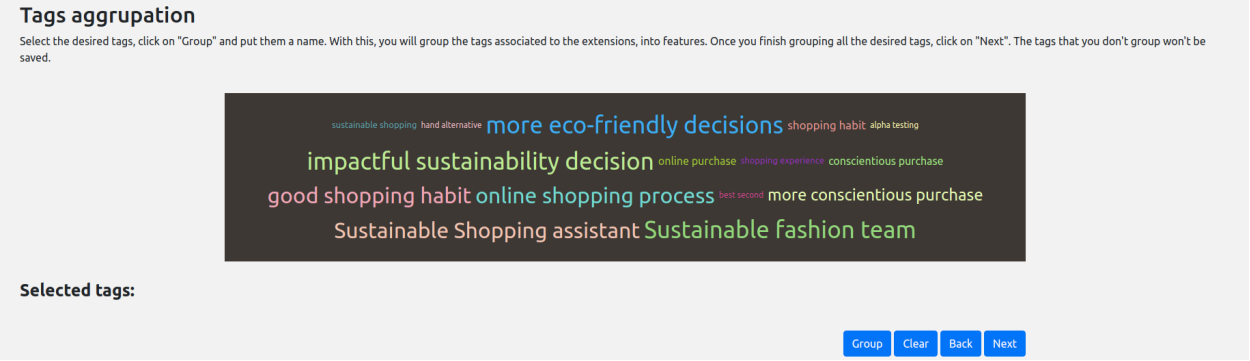

Figure 1.10: Palabras clave cargadas.

Tras esto, se puede comenzar a realizar las agrupaciones, lo cual es muy sencillo de hacer: se seleccionan las palabras clave deseadas, se pulsa el botón *Group*, se introduce el nombre de la agrupación y *Ok*.

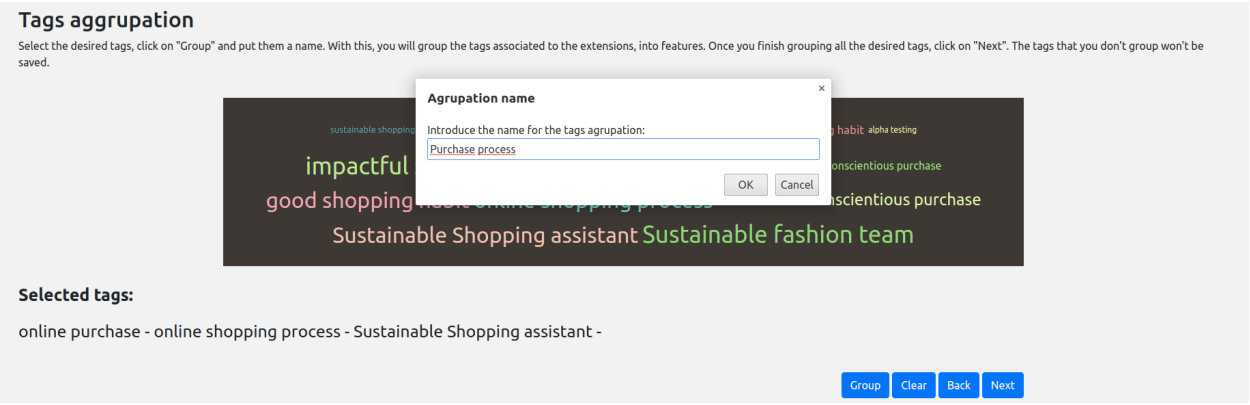

Figure 1.11: Crear agrupación.

De esta manera se van creando las agrupaciones, las cuales se van mostrando en la parte inferior. El programa dispone de la opción de borrar las agrupaciones en caso de crear una que no sea la deseada.

Figure 1.12: Visualizar agrupación.

Una vez que se han creado todas las agrupaciones deseadas, se prosigue pulsando el botón *Next*.

1.5 Paso 4: Realizar el modelo Kano

En esta última página se muestra el modelo Kano, el cual se debe rellenar con las agrupaciones hechas previamente.

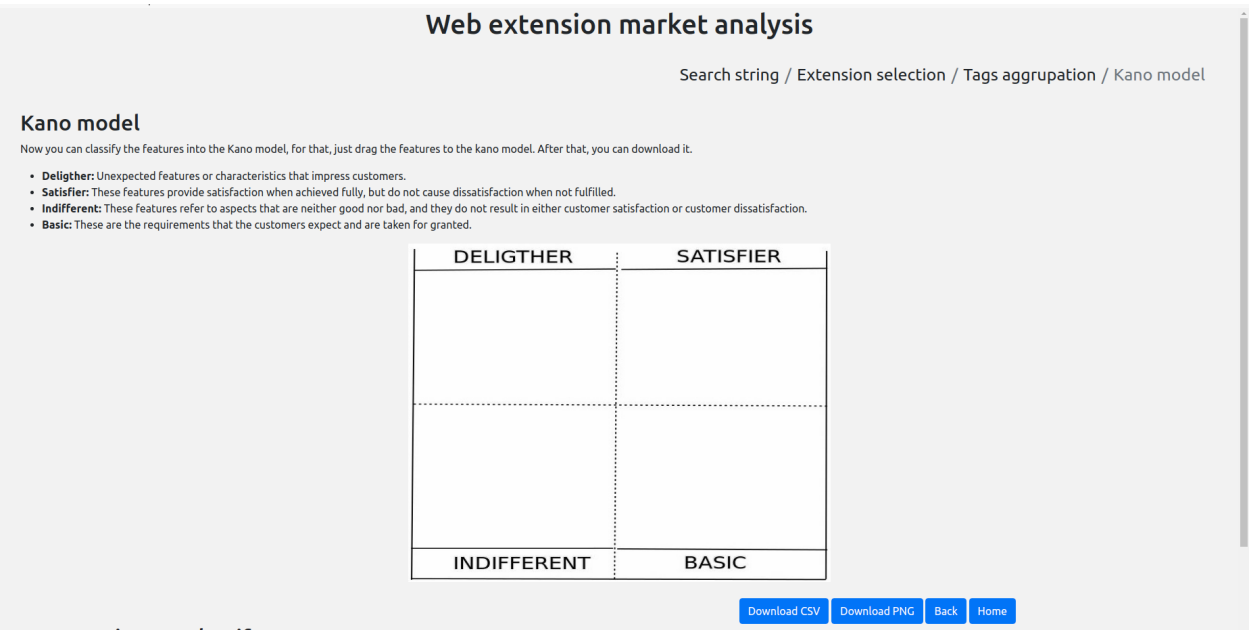

Figure 1.13: Última página.

Para esto, se debe bajar un poco mediante la rueda del ratón, y se podrá ver un *clod tag*, el cual muestra todas las agrupaciones hechas, y que también sirve para ayudar en la clasificación (las agrupaciones más grandes tienen más relevancia).

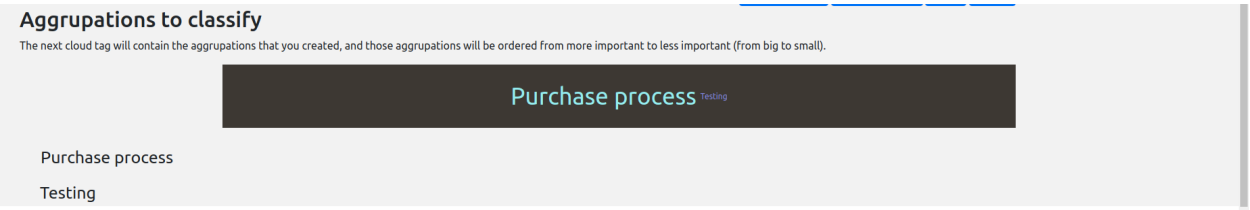

Figure 1.14: Agrupaciones.

En caso de pulsar una de las agrupaciones se muestra las palabras claves que contiene. Para organizar las agrupaciones, es tan simple como mover las que están debajo del *cloud tag* al modelo Kano. Y una vez hecho esto, se peude descargar una captura del modelo Kano, o un .csv de las agrupaciones, su peso y las palabras clave que contiene.

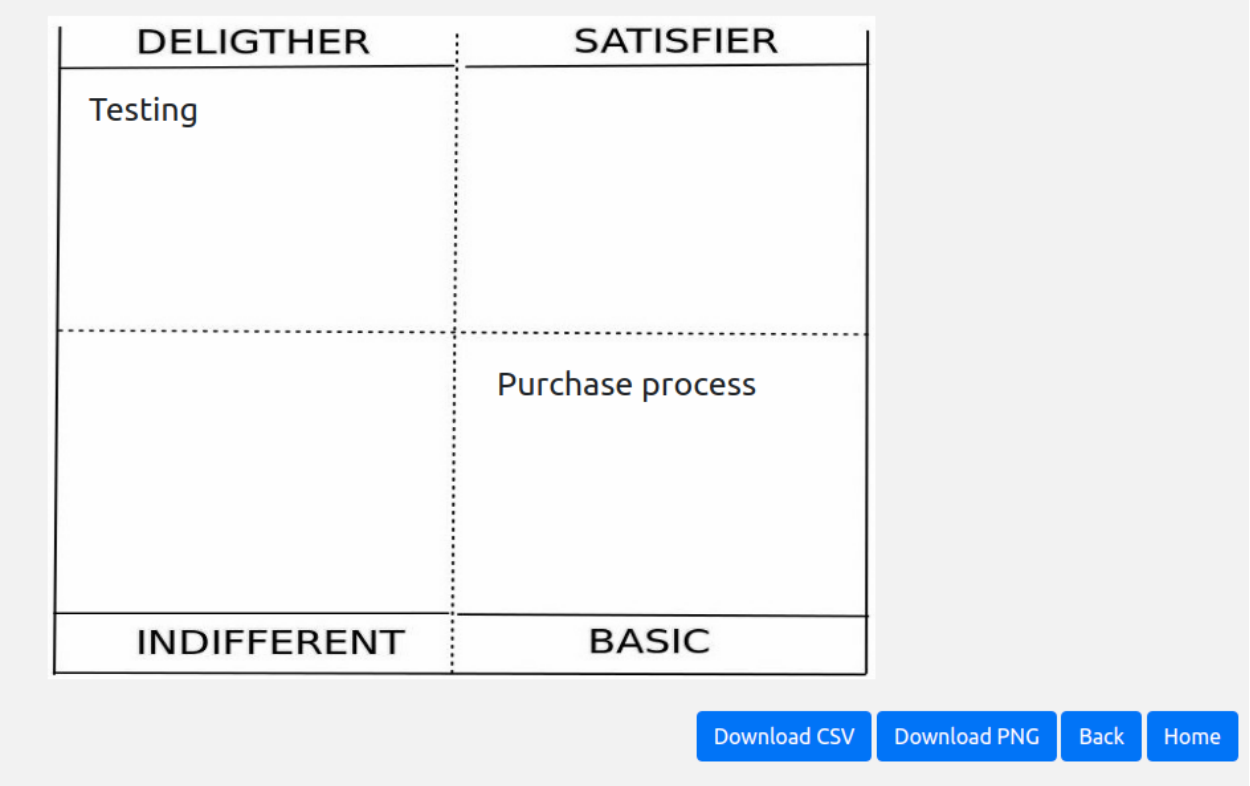

Figure 1.15: Modelo Kano terminado.

Bibliografía

[Azure, 2020] Azure (2020). Docker Compose Server. https://azuremarketplace.microsoft.com/es-es/marketplace/apps/cloud-infrastructure-services.docker-compose-ubuntu20?tab=overview/. [Online; accedido el 22-diciembre-2021].

[Chen and Li, 2019] Chen, Y. and Li, M. (2019). Smart tv terminal interface design for the elderly based on user experience. *IOP Conference Series: Materials Science and Engineering*, 573:012056.

[Dik, 2018] Dik, J. (2018). How will long tail affect the leisure branch? *Medium*.

[Documentation,] Documentation, D. What is a Container? | App Containerization | Docker. https://www.docker.com/resources/what-container. [Online; accedido el 17-diciembre-2021].

[IZO, 2020] IZO (2020). How can I use the Kano Model to Improve my Product? https://izo.es/en/como-aplicar-el-modelo-de-kano-en-el-diseno-de-productos/. [Online; accedido el 12-enero-2021].

[Modroño, 2021] Modroño, T. (2021). How to Conduct a Cognitive Walkthrough. *The Interaction Design Foundation*.

[Salas, 2017] Salas, A. R. (2017). La Esquina de la Gestión (Alberto Redondo Salas): Ciclo de vida de proyectos: Clásico, Iterativo y Ágil.

[Sekhon, 2020] Sekhon, S. (2020). How to create an Electron app?