Universidad de Valladolid

Escuela Técnica Superior

Ingenieros de Telecomunicación

Trabajo Fin de Grado

GRADO EN INGENIERÍA DE TECNOLOGÍAS DE TELECOMUNICACIÓN

Sistema RPA para gestión de consumos de licencias y sistemas internos

Autor:

D. Juan José Hernanz Fernández

Tutores:

D. Rubén Mateo Lorenzo Toledo

Valladolid, 01 de julio de 2021

Título:	**Sistema RPA para gestión de consumos de licencias y sistemas internos**
Autor:	**D. Juan José Hernanz Fernández**
Tutores:	**D. Rubén Mateo Lorenzo Toledo**
Departamento:	**Teoría de la Señal y Comunicaciones e Ingeniería Telemática**

Tribunal

Presidente:	**Evaristo J. Abril Domingo**
Vocal:	**Rubén M. Lorenzo Toledo**
Secretario:	**Patricia Fernández del Reguero**

Fecha:	**01 de julio de 2021**
Calificación:	

Resumen del TFG

En este trabajo se presenta el desarrollo de dos soluciones *software* para la automatización de procesos de una empresa. El resultado son 2 sistemas que tienen en común muchas de sus partes fundamentales. La funcionalidad básica de ambos es un *bot* basado en la librería *puppeteer* de *Node.js*. Este *software* simula sesiones en la web para extraer y almacenar datos de interés para la empresa. A través de las herramientas de *Google Sheets* y *Data Studio* estos datos se visualizan en informes que serán consumidos por diferentes departamentos. Las diferencias en los sistemas vienen determinadas por la naturaleza de estos datos. Un sistema representará gráficas de consumos de licencias del *software* de *Acoustic*, mientras que el otro estará orientado a visualizar tiempos de carga de una página web medidos a través de sesiones de *Acoustic*. El contenido de este documento pretende ser un repaso completo de todo el proceso llevado para el desarrollo de estas soluciones.

Palabras clave

Automatización de Procesos Robóticos, *Puppeteer*, web-scraping, Inteligencia de Negocio, visualización, analítica de datos.

Abstract

This project presents the development of two *software* solutions for the automation of a company's processes. This results in 2 systems that have many of their essential pieces in common. The basic feature of both is a bot based on the *puppeteer* library of *Node.js*. This software simulates web sessions to extract and store valuable data for the company. Through *Google Sheets* and *Data Studio* tools, these data are displayed in reports that are then consumed by different departments. The differences between the systems are determined by the nature of this data. One system will represent graphs of licence consumption of the *Acoustic* software, while the other will be focused on visualising web page load times measured through *Acoustic* sessions. The content of this document is intended to be a complete overview of the whole process carried out to develop these solutions.

Keywords

Robotic Process Automation, *Puppeteer*, web-scraping, *Business Intelligence*, visualization, data analytics.

Agradecimientos

A mis padres, por su confianza plena.

A mis abuelos, porque sé que estarían muy orgullosos.

A mis hermanos y círculo cercano, por haberme soportado en esta etapa.

A mis amigos de la carrera, por ser la motivación diaria para continuar.

A Jimena, por apoyarme cada día.

A PJ, por estar siempre en mi equipo.

A Rubén, por guiarme en este proceso.

A Azu y mis compañeros de LUCE, por darme esta oportunidad.

Índice general

Referencias **75**

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

1.1. Motivación

Muchas de las tareas llevadas a cabo por los trabajadores de una empresa resultan poco estimulantes y tediosas por su carácter manual y repetitivo. Esto no sólo tiene un impacto sobre la motivación de los trabajadores, sino también sobre la producitividad de la propia empresa. Los **sistemas de automatización de procesos robóticos**, o **RPA** por sus siglas en inglés (*Robotic Process Automation*), nacen con el objetivo de mitigar estos procesos ineficientes, simples y tediosos [Tau20]. De esta forma, constituyen una nueva fuerza de trabajo que será fuente de valor de la empresa y que está adquiriendo una importancia cada vez mayor. Esta nueva industria está ganándose la atención de empresas, inversores e, incluso, de los propios empleados. Como una primera aproximación, entendemos RPA como la tecnología *software* capaz de emular interacciones de humanos con sistemas digitales para realizar un proceso empresarial de manera automática.

El diseño e implementación de un RPA es el objetivo que nos concierne en este Trabajo de Fin De Grado (TFG). Surge como una oportunidad de negocio en el transcurso de mis prácticas en la empresa *LUCE Innovative Technologies*[1]. *LUCE IT* es una entidad que ofrece soluciones tecnológicas a sus clientes especializada, sobre todo, en *Data* y Automatización. Por ello, entre sus múltiples servicios incluye la automatización de procesos de negocio y de operaciones mediante soluciones RPA. Como empresa tecnológica, además de acompañar a sus clientes en en su transformación digital, incorpora estas nuevas tecnologías en sus propios procesos productivos. En este contexto, yo he colaborado con el departamento de *data* desempeñando tareas de **analítica de datos** enfocadas a la experiencia del cliente. En el desarrollo de mis tareas, surgió la oportunidad de implementar un sistema RPA basado en *web-scrapping*[2] y herramientas de visualización para agilizar procesos del equipo de negocio. Además, suponía una oportunidad de negocio a la hora de ofrecer nuevas soluciones de análisis de datos a los clientes de la empresa.

La puesta en marcha de esta idea vino motivada por la existencia de *Puppeteer*[3]. Se

[1] https://luceit.com/es/
[2] Técnica de extracción de información de un sitio web mediante un programa *software*.
[3] https://pptr.dev/

trata de una librería de *Node.js* que proporciona una API de alto nivel para controlar *Chrome* o *Chromium* a través del *Protocolo DevTools*[4]. Dicho en otras palabras, es un *software* que permite automatizar acciones sobre los navegadores de *Google*. Si además tenemos en cuenta que está desarrollada por *Google*, estamos ante una herramienta tremendamente potente para resolver problemas de *Automatización*, *Big Data* y análisis de rendimiento. Y esto, como hemos visto, son pilares fundamentales de la empresa.

Teniendo todo lo anterior en cuenta, se plantea en el presente Trabajo de Fin de Grado el análisis, diseño e implementación de un sistema RPA en el seno de la empresa *LUCE IT*. Dicho *bot* deberá automatizar las tareas de extracción de datos de la web de *Acoustic Experience Analytics*[5], el procesado de esos datos, su publicación y actualización en un conjunto de datos y, en última instancia, la visualización de esos datos en *dashboards*[6] para poder ser analizados.

1.2. Objetivos y alcance

El objetivo del TFG será implementar un **sistema RPA que, mediante *dashboards*, permita visualizar y analizar datos que previamente haya extraído de la web de *Tealeaf***. Gracias a su flexibilidad, este sistema permitirá agilizar tareas a nivel interno en la empresa. Pese a que el desarrollo de este proyecto podría emplarse para infinidad de aplicaciones, el alcance del trabajo estará delimitado a implementar las siguientes soluciones:

- Extraer, almacenar y visualizar los datos del consumo mensual de licencias de *Tealeaf*. Consistirá en un *dashboard* utilizado a nivel interno por el Equipo de Negocio para controlar que no se exceda el límite mensual contratado.
- Extraer, almacenar y visualizar datos de rendimiento de un sitio web a partir de los datos recogidos en *Tealeaf*. Consistirá en la creación de 3 *dashboards* donde el departamento de Optimización de Rendimiento Web tenga acceso a toda la información relativa a los tiempos de carga percibido por los usuarios en la web.

La consecución de estos objetivos reportará interesantes beneficios a las partes implicadas:

- Se agilizarán procesos internos gracias a la automatización. Esto hará que los empleados no pierdan tiempo y esfuerzos en tareas simples y repetitivas.
- Mejoras del control del consumo licencias del *software* de *Tealeaf* por parte de la empresa. Esto contribuye a una mayor calidad de la información de la que se dispone a la hora de tomar decisiones futuras.

[4] https://chromedevtools.github.io/devtools-protocol/

[5] También referida como *Tealeaf*, es una herramienta para la analítica de experiencia de usuario a través de la captura de sesiones web. Web: https://www.acoustic.com/products/tealeaf.

[6] Traducido habitualmente como *cuadro de mandos*, es una herramienta de visualización de métricas o indicadores que dan información del estado de una empresa, máquina o proceso.

- El desarrollo de este sistema puede utilizarse como punto de partida para automatizar otras muchas tareas y procesos que mejoren la productividad de la entidad.
- Al utilizarlo como solución en la prestación de sus servicios, la empresa adquiere una oportunidad de negocio de la que podrá obtener rentabilidad si es capaz de paquetizarla y comercializarla entre sus clientes.
- Desde la perspectiva del usuario final, recibe un servicio más rápido y que aporta un mayor valor al dato.

Para el desarrollo de todo el sistema se han utilizado diferentes tecnologías. El *bot* consiste en un proyecto de *Node.js* que utiliza la librería de *Puppeteer* para la emulación de navegaciones y extracción de datos y la librería de *Google Sheets API* para poder editar hojas de cálculo de *Google*. Por último, para la visualización de *dashboards* se ha utilizado *Google Data Studio*.

1.3. Metodología

Al tratarse de un Trabajo de Fin de Grado asociado a una empresa, el desarrollo *software* ha formado parte de la metodología *Agile* que utiliza *LUCE IT* en la gestión de sus proyectos. Se trata de una aproximación iterativa e incremental de desarrolo *software* que parte de un plan detallado y requiere una fuerte comunicación entre los equipos [Tau20]. Dentro de este modo *Agile* de trabajo se utilizan los modelos *Scrum* y *Kanban* en la gestión de proyectos. La metodología *Scrum* divide el trabajo en equipos auto-organizados, que realizan pequeñas tareas clasificadas por prioridad y esfuerzo estimado y todo ello en iteraciones de dos semanas denominadas ***sprints***. Gracias a la aproximación *Kanban* se visualiza el estado de las tareas dentro de un proyecto al clasificarlas en columnas [HK10].

Para completar los objetivos del presente TFG, se ha dividido el trabajo en diferentes tareas integradas dentro de distintos proyectos de la empresa y se ha seguido el modelo iterativo incremental descrito en el párrafo anterior. Además, para organizar el contenido del documento se ha utilizado la división en cinco fases del Ciclo de Vida del Desarrollo de Sistemas, **SDLC** por sus siglas en inglés (Systems Development Life Cycle) que se muestran en la Figura 1.1 y que se describen a continuación:

- Planificación: fase inicial en la que se fijan los objetivos que se pretenden conseguir y se propone un primer planteamiendo del problema.
- Análisis: A partir de los objetivos, se definen de manera uniforme los requisitos del sistema que vendrán dados por el cliente o usuario final y que sirven para definir la interacción entre las partes del sistema. También se crean los Diagramas de Flujo de Datos.
- Diseño: que consta de dos partes diferenciadas. Por un lado el diseño del sistema, en el que se analizan y deciden las diferentes partes que lo componen. Por otro lado En esta fase, se definen las tareas que han de completarse, la estimación de esfuerzo de cada una de ellas y su prioridad.

- Implementación: fase de desarrollo del *software*. Consiste en la ejecución de las tareas. Acapara la mayor parte del tiempo.
- Despliegue y Manenimiento: paso a producción del sistema una vez pasados los *tests*. Los cambios a partir de este momento son inusuales.

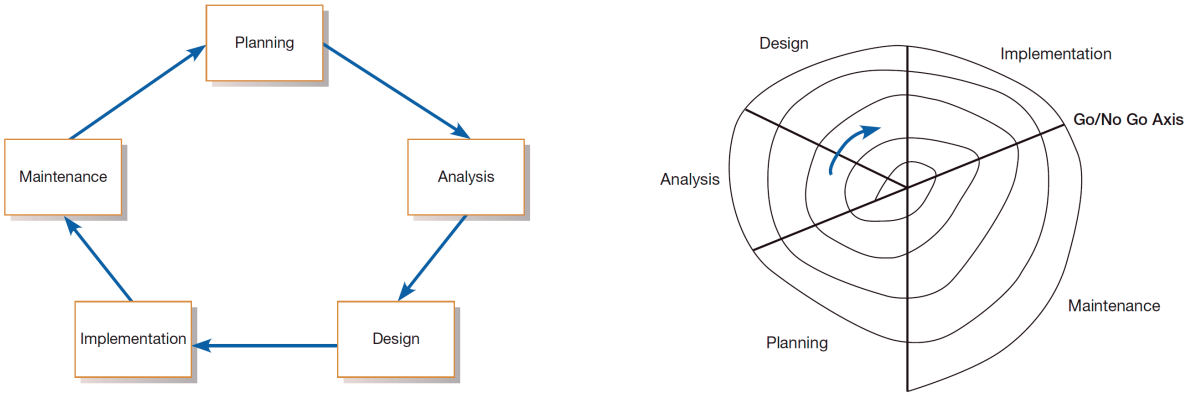

Figura 1.1: Ciclo de Vida del Desarrollo de Sistemas (SDLC) [JSV17].

Todas estas fases se han llevado a cabo durante los mencionados *sprints*. Al incio del sprint se organizan las tareas que se quieren completar en el período de 2 semanas. Durante el *sprint*, tienen lugar a diario unas reuniones cortas de equipo, denominadas *daily Scrum* en las que se revisa el proceso y se analizan posibles obstáculos. Al final de cada *sprint* se evalúa la consecución o no de estas tareas, se ajustan tiempos si es necesario y se crean nuevas tareas. [Sch02]

1.4. Estructura del documento

Este documento está dividido en cinco capítulos teniendo en cuenta este primero que ha sido introductorio. A continuación, en el Capítulo 2, se profundiza en los sistemas RPA y, particularmente, en aquellos basados en *web-scrapping*. Además se pone en valor la importancia del dato aplicado a la Experiencia de Usuario (UX)[7] y a la Inteligencia de Negocio (BI)[8] en las empresas actuales. Después, en el Capítulo 3, se detalla todo lo referido a la Planificación y el Análisis del sistema. Se incluye un primer acercamiento al sistema así como sus requisitos y Diagramas de Flujo de Datos. Más tarde, en el Capítulo 4, se describen todas las tecnologías utilizadas y se detallan los procesos de Diseño, e Implementación del sistema RPA. Finalmente, en el Capítulo 5 encontramos las conclusiones relativas al trabajo realizado y sus potenciales siguientes pasos.

[7]UX por sus siglas en inglés (User Experience), hace referencia a la calidad percibida por los usuarios en su interacción con un sistema.

[8]BI por sus siglas en inglés (Business Intelligence), hace referencia a uso de los datos por parte de los equipos de negocio para transformarlos en conocimiento y mejorar la información en la toma de decisiones de la empresa.

Capítulo 2

Estado del arte

2.1. Introducción

Los sistemas RPA constituyen uno de los puntos clave para el aumento de la productividad de una empresa a través de la automatización de sus procesos. La clave consiste en tranferir la ejecución de aquellas tareas más repetitivas y simples desde los humanos hasta las máquinas. Esta transición hacia la automatización supone una mayor fuente de valor si se aplica sobre procesos de toma de decisiones de la empresa. Pero el abanico de posibilidades que nos brinda esta tecnología es muy variado. Por ejemplo, veremos cómo podemos aplicar la RPA sobre la Experiencia de Usuario, una disciplina que actualmente está tomando más importancia que nunca en las páginas web y aplicaciones.

Este capítulo consta de 4 partes principales: en la Sección 2.2 se explica en profundidad el concepto de RPA, su historia y la proyección futura de la industria. Además en la subsección 2.2.1. se propone una tipología de RPA y en la 2.2.2. se introduce la técnica de *web-scraping*. En la Sección 2.3. se explican los fundamentos de la Inteligencia de Negocio o BI y su relación con la RPA. En la subsección 2.3.1. se pone énfasis en las herramientas de visualización denominadas *dashboards*. Más adelante, la Sección 2.4 está dedicada a la Experiencia de Usuario. A su vez este apartado está dividido en 3 subsecciones: en la 2.4.1. se introduce la técinca de *Real User Monitoring*, en la 2.4.2. nos detenemos en la importante API de *Navigation Timing* y finalizamos con un repaso de las nuevas tendencias en las métricas de la Experiencia de Usuario en la subsección 2.4.3. Finalmente, en la Sección 2.5 se recopila toda la información expuesta referente al estado del arte y se propone una relación entre todos los conceptos tratados a modo de conclusión.

2.2. Sistemas RPA

En el Capítulo 1 se recoge un primer acercamiento a la automatización de procesos robóticos. Para entender mejor el fundamento de estos sistemas, resulta apropiado repasar sus términos. El adjetivo *robóticos* no hace referencia a un artefacto físico, sino más bien a un robot basado en *software* (comúnmente denominado *bot*). El término *procesos* bien

podría ser sustituido por *tareas*, entendidas como un conjunto de acciones para la consecución de un objetivo [Tau20]. Por último tenemos la palabra *automatización*, que se refiere a la técnica aplicada sobre un sistema para que opere de manera automática [SM19]. Recopilando todo esto, describimos un sistema RPA como la **tecnología *software* capaz de emular interacciones de humanos con sistemas digitales para realizar un proceso empresarial de manera automática.**

Pese a que algunas proyecciones actuales predicen que el mercado de RPA estará saturado en 2023[1], estas tecnologías tan solo llevan conviviendo con nosotros desde principios de los años 2000 [Tau20]. A continuación, se hace un repaso de su breve historia.

El concepto de **automatización** dista mucho de ser novedoso, pero su aplicación en ejemplos de la vida real ha sido ampliamente impulsada durante los últimos 70 años gracias a la aparición de los ordenadores. Automatización y ordenadores son conceptos que están estrechamente ligados y han evolucionado conjuntamente sentando las bases de las tecnologías RPA. Las primeras computadoras eran enormes máquinas que sólo las grandes empresas se podían permitir, pero resultaban sumamente últiles en la gestión de sus funciones centrales. Con los avances en los microprocesadores y los sistemas operativos llegó la Revolución del PC que permitió a cualquier empresa automatizar sus procesos. Todo esto sentó los pilares de la RPA, que tuvo su origen a principios de siglo. Inicialmente no atrajo mucha atención, por una parte por ser considerada como baja tecnología y, por otra, porque el foco estaba puesto en el incipiente mercado de la nube. Alrededor del 2012 esta tendencia cambió y el mercado de RPA comenzó a atraer a entidades e inversores erigiéndose a día de hoy como la tecnología de mayor crecimiento dentro de la industria *software*. Esto no fue más que el resultado de la convergencia de muchas tendencias como la búsqueda de reducir costes para hacer frente a las secuelas de la crisis financiera, el interés por la transformación digital o la sofisticación de la tecnología, entre otros [Tau20].

Si miramos al futuro del sector las proyecciones son igualemente prometedoras. Según la consultora global *McKinsey & Company*[2], el impacto financiero de la RPA podría alcanzar los 6,7 billones de dólares en el año 2025. Además, afirmó que, tan sólo en el primer año, las empresas que adopten esta tecnología tendrán un retorno de inverión potencial de entre el 30 y el 200 %.

2.2.1. Tipos de RPA

En función su naturaleza, podemos dividir a la tecnología RPA en 3 clases distintas. Además, estas clases se han diferenciado de manera natural con la constante evolución de la tecnología. Es decir, podemos asociar cada clase a una generación de RPA [Tau20]:

- **RPA asistida**: el *software* porporciona asistencia en ciertas tareas en colaboración con una persona. Es la primera forma de RPA que surgió alrededor del 2003.
- **RPA desatendida**: los procesos se automatizan sin la necesidad de ningún tipo de

[1] https://www2.deloitte.com/ro/en/pages/technology-media-and-telecommunications/articles/deloitte-global-rpa-survey.html

[2] https://globalpayrollassociation.com/blogs/technology/what-the-history-of-rpa-technology-says-about-its-future

intervención humana. Se trata de la segunda generación de RPA.

- **RPA cognitiva** (también conocida como Automatización Inteligente de procesos o IPA): el sistema RPA incluye Inteligencia Artificial que le permite aprender por sí solo. De esta manera, no es necesaria la intervención humana incluso en la toma de algunas decisiones. Se trata de la última generación de RPA.

Una de las preguntas que más se hacen tanto desarrolladores como empleados y gestores es: "¿Qué procesos deben automatizarse y cuáles deberían ser realizados por humanos?" La Figura 2.1 arroja luz a esta pregunta. La línea azul representa la distribución de procesos susceptibles de ser automatizados mediante RPA. Se tiene en cuenta la frecuencia con la que han de realizarse (eje y) los distintos procesos (eje x). Esta función sigue una distribución de Pareto en la que se indica que el objetivo de la automatización se centra en el 20 % de los procesos que son los más frecuentes. La parte final del gráfico representa los procesos menos frecuentes que necesitan ser realizados por humanos. En la parte central se encuentran procesos repetitivos pero no demasiado frecuentes. Esto hace que sean susceptibles de ser automatizados siempre y cuando sea económicamente viable [vdA18]. En la actualidad, con la incorporación de la Inteligencia Artificial y el *Machine Learning* a la tecnología RPA, la posición de estos umbrales se está modificando en favor de una mayor proporción de procesos automatizados.

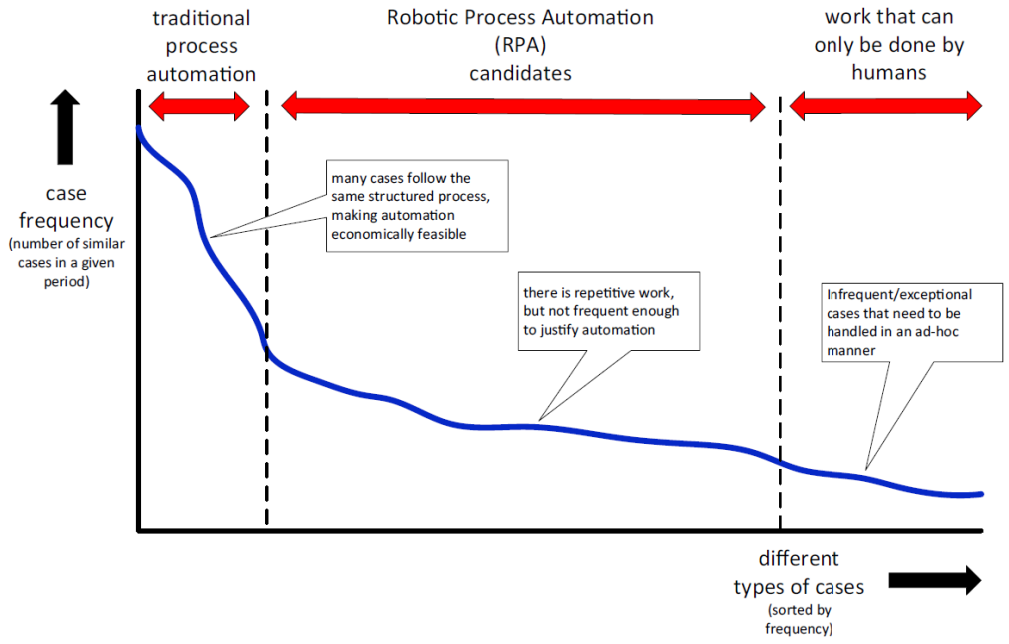

Figura 2.1: Posicionamiento de RPA [vdA18].

Por lo tanto el objetivo de la RPA es sustituir a los humanos en las tareas más repetitivas para que su productividad aumente. Pese a que el abanico se está expandiendo, a continuación se enumeran algunas de las tareas que tradicionalmente han sido automatizadas [Tau20]:

- El traspaso de información de unas plataformas a otras mediante la técnica de copia-pega.
- Abrir una web y hacer el *login*.
- Abrir, o reenviar *e-mails*,
- La lectura y escritura de una base de datos.
- La extracción de contenidos de docmentos, gráficas o formularios.
- El uso de cálculos y flujos de trabajo.

2.2.2. RPA basados en *web-scraping*

En su objetivo de automatizar un proceso, un sistema RPA puede hacer uso de tantas tecnologías *software* como sean necesarias. De entre todas estas, resultan interesantes las herramientas de *web scraping*. ***Web scraping*** es una herramienta que se basa en la construcción de un agente *software* para descargar, analizar y organizar los datos de una web de manera automatizada [SvB18]. Si lo enmarcamos en el contexto de una empresa, la técnica de *web scraping* resulta muy potente para automatización de muchas tareas que precisan de datos que se encuentran expuestos en la web.

En este punto resulta interesante plantear la siguiente pregunta: Si el *web scraping* se utiliza para recuperar datos que se encuentran en un sitio web concreto, ¿no es justo eso lo que nos permiten las *Application Programming Interface* (APIs)? La respuesta es que sí, las APIs proporcionan un medio para que el mundo exterior tenga acceso a un conjunto de datos de un sitio, de una manera estructurada. Sin embargo, el *web scraping* encuentra su fundamento precisamente en las carencias de las APIs. En términos generales, siempre será más adecuado utilizar una API para extraer un conjunto de datos de la web, pero esto no siempre va a ser posible debido a diferentes factores:

- El sitio web no ofrece una API.
- La API tiene un coste que no queremos asumir.
- Existen limitaciones a las peticiones que se pueden realizar.
- Los datos que nos interesan, no están disponibles a través de la API.

El *web scraping* surge como solución a todos los problemas descritos [SvB18]. Si bien es una forma de atacar el problema más compleja y costosa, es también mucho más flexible y completa. Al fin y al cabo, si puedes ver datos en la web, puedes extraerlos.

La recopilación automática de datos presentes en Internet es probablemente tan antigua como la propia Internet. La cantidad de datos que en la red están incluidos es inconmensurable, por ello la técnica de *web scraping* es utilizada ampliamente por distintos agentes con propósitos muy dispares. Por ejemplo, muchos de los productos de *Google* se han beneficiado de esta técnica (el *Google Translate* se entrena a sí mismo con datos que extrae de la propia web). También se puede utilizar con muchos fines sociológicos: una

simple recopilación de mensajes de *Twitter* puede ser suficiente para conocer las tendencias políticas de la población o incluso para prevenir suicidios, como se hizo en un estudio realizado en Canadá[3].

En un contexto más empresarial, algunos ejemplos actuales del uso de esta técnica serían la recopilación de datos de competidores (como sus precios) o la creación de conjuntos de datos a partir de información procedente de distintos canales web.

Para conseguir recopilar estos datos de Internet, el *bot* debe tomar el control del navegador emulando sesiones tal y como lo haría un ser humano, automatizando los *clicks*, la escritura de campos y el resto de interacciones posibles. En la actualidad existen dos herramientas que son las más utilizadas con este propósito. La primera y más conocida es *Selenium*[4] que permite la automatización de navegadores y es comúnmente utilizada para el *testing* de aplicaciones web. La segunda, ya mencionada en el Capítulo 1, es *Puppeteer*, una librería de *Node.js* que proporciona una API de alto nivel para controlar *Chrome* o *Chromium* a través del *Protocolo DevTools*. A pesar de que existen numerosas diferencias entre ellas, ambas pueden ser empleadas para el problema que nos concierne. *Selenium* es mucho más popular entre los usuarios, lo que le hace tener una comunidad más sólida, además de ser compatible con todos los navegadores. Por su parte *Puppeteer* cuenta con menos popularidad, pero su uso está creciendo en los últimos meses. Su ventaja principal es que ha sido desarrollada por el gigante *Google* y es una herramienta muy potente para la automatización de este navegador en concreto. Elijas cualquiera de las dos, u otra herramienta alternativa, lo que se pone de manifiesto es la importancia que tiene este tipo de *software* en la actualidad, contando con el desarrollo de enormes empresas y ampliando las opciones de evolución de los sistemas RPA.

2.3. Inteligencia de Negocio (BI)

En los procesos de decisión de la empresa, el Equipo de Negocio tomará decisiones más acertadas cuanta mayor información tenga a su disposición. Con el objeto de recopilar la mayor cantidad de información posible y con el foco puesto en la calidad de esta información, surgen los **Sistema de Apoyo a la Decisión** (DSS por sus siglas en inglés). Definidos por primera por *Gorry* y *Scott-Morton* en 1971, los DSS son *"sistemas informáticos interactivos, que ayudan a los responsables de la toma de decisiones a utilizar datos y modelos para resolver problemas no estructurados"*. En otra definición clásica de DSS, *Keen* y *Scott-Morton* añadieron que *"los sistemas de apoyo a la toma de decisiones combinan los recursos intelectuales de los individuos con las capacidades del ordenador para mejorar la calidad de las decisiones"* (1978). Estamos por lo tanto ante un término sin una definición aceptada universalmente y que por ello puede tener diferentes significados para distintas personas. Como norma general, se utiliza el concepto DSS para describir cualquier sistema que proporciona apoyo en la toma de decisiones de una organización [RS14].

Con los avances de la tecnología, surgió una nueva generación de gestores mucho más cómodos con el uso de herramientas *software* que encontraron en estas un potente aliado

[3] https://www.sas.com/en_ca/insights/articles/analytics/using-big-data-to-predict-suicide-risk-canada.html

[4] https://www.selenium.dev/

para tomar decisiones de una manera más rápida e inteligente. A mediados de los años 90, nuevas herramientas de almacenamiento de datos, visualización, OLAP[5] o minería de datos, entre otras, surgieron bajo el nombre de ***Business Intelligence*** **(BI)** [RS14]. Podemos definifir la Inteligencia de Negocio como sistemas que combinan recopilación de datos, almacenamiento de los mismo y gestión del conocimiento con técnicas de análisis para evaluar información compleja y competitiva relacionada con la entidad. Su objetivo último es mejorar el tiempo y la calidad de los *inputs* sobre el proceso de decisión [SN08]. La Figura 2.2 muestra los componentes que configuran el sistema único de BI.

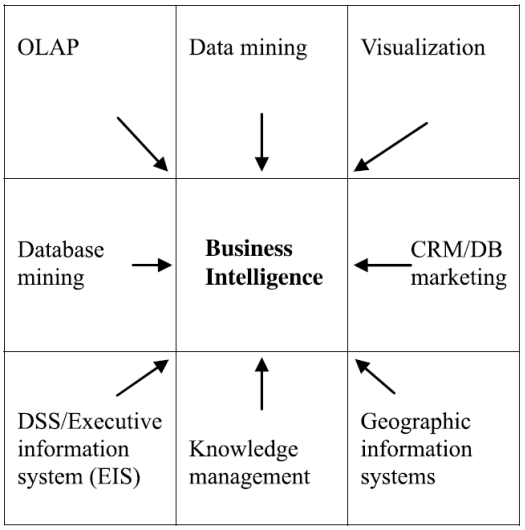

Figura 2.2: Los compomentes de la Inteligencia de Negocio [SN08].

Es posible clasficar la BI en función de sus aplicaciones. En este sentido, *MicroStrategy Corp.*[6] separa sus herramientas de BI en 5 tipos distintos:

- Entrega de informes y Alerta.
- Infomes de Negocio (mediante *dashboards* y tarjetas de puntuación o *scorecards*).
- Análisis de Cubos.
- Consultas Ad hoc.
- Minería de Datos.

Actualmente, muchos los proveedores ofrecen este tipo de herramientas de análisis. Quizá, el *dashboard* sea el más importante debido a sus utilidad [SN08].

[5]OLAP por sus siglas en inglés (OnLine Analytical Processing), es una solución de BI cuyo objetivo es agilizar la consulta de grandes cantidades de datos.

[6]https://www.microstrategy.com/en/business-intelligence

2.3.1. *Dashboards*

Los paneles de control, habitualmente referidos por su término en inglés: ***dashboard***, son herramientas digitales de visualización que concentran la infomación del pasado o presente de las métricas más importantes del rendimiento de una empresa, también conocidas como *Key Performance Indicators* (KPI). Dentro de la clasificación que hemos descrito previamente, se situarían en los Informes de Negocio como herramienta de BI. A través de interfaces web hacen uso de tablas, gráficos y tarjetas para mostrar de una manera ágil los resultados de BI.

Estas herramientas de visualización poseen una serie de características que les proporcionan un gran atractivo: presentan muchas métricas en una única pantalla, tranforman los datos de más bajo nivel en resúmenes agregados y presentan información muy fácil de entender por cualquier persona sin la necesidad de poseer conocimientos previos. Además, los *dashboards* permiten muchísima personalización para lograr informes que se ajusten al máximo posible a las necesidades del usuario final. En la Figura 2.3 se muestra un ejemplo de un *dashboard* construido en la herramienta *UiPath*[7].

Figura 2.3: Ejemplo de dashboard de *UiPath*.

[7] *UiPath* es el proveedor de servicios RPA líder del sector según *Forrester Reseach*https://www.uipath.com/es/resources/automation-analyst-reports/forrester-wave-rpa

Continuando con la propia visión de *UiPath*, debemos entender la técnica de **visualización de datos** como una herramienta disponible dentro de la Inteligencia de Negocio. Desde su posición líder dentro del mercado de RPA, esta corporación pone de manifiesto la importancia de la BI para impulsar un negocio. Entre su beneficios, destaca la identificación de problemas y oportunidades y la aceleración de la productividad. Además remarca que cuando la RPA se combina con la BI para la búsqueda de un objetivo común, los beneficios se multiplican principalmente por dos motivos [Koc18]:

- La automatización que caracteriza a la RPA, será de gran valor para los analistas de BI agilizando principalmente las tareas de recopilación y procesamiento de datos, que suelen suponer un elevado esfuerzo manual por su caracter repetitivo y estandarizado.
- El camino hacia la automatización puede dirigirse de forma más eficiente mediente la integración de herramientas de BI en una plataforma de RPA.

2.4. Experiencia de Usuario

La **Experiencia de Usuario (UX)** es un término que hace referencia a la calidad percibida por los usuarios en su interacción con un producto o sistema. En el campo de la interacción humano-computadora (HCI) es un término que se ha hecho muy popular en los últimos años, pero que reúne diversos significados. Así pues, alude a la experiencia del usuario en términos que van desde la usabilidad a la belleza, pasando por los aspectos más afectivos o funcionales del uso de una tecnología [Has06].

La UX tiene un origen multidisciplinar en el que destacan la sociología, psicología, marketing y diseño. Sin embargo, el paso del tiempo ha contribuido a que sea un concepto que se emplea únicamente en el desarrollo de interfaces digitales como aplicaciones o sitios web. Uno de sus puntos de análisis es la **usabilidad** cuyo objetivo es mejorar el rendimiento en la interacción de los usuarios con el sistema [FDGS18], en nuestro caso será una página web. Para conseguir mejorar este rendimiento debemos ser capaces, en primer lugar, de medir cómo lo están experimentando los usuarios reales; a partir de ahí podemos determinar los puntos de acción. Así surge la necesidad de poder obtener métricas de sesiones de usuarios reales en la web, técnica conocida como *Real User Monitoring*.

2.4.1. *Real User Monitoring* (RUM)

Desde los primeros días de Internet, los desarrolladores de páginas han puesto mucho empeño en medir el rendimiento de su web cuando alguien la visita. Esto va estrechamente ligado a obtener una visión más profunda y significativa de la Experiencia de Usuario que repercutirá directamente sobre los ingresos y el crecimiento de la compañía dueña de la página [Sny15].

La técnica de ***Real User Monitoring*** (RUM), también referida por algunas fuentes como *End-User Experience Monitoring* (EUEM), tiene como objetivo la medición del rendimiento de una página o aplicación directamente de las interacciones de los usuarios reales. A los resultados se les llama métricas de campo o *in the field*. En contraposición, si

los resultados se obtienen de experimentos en entornos controlados se denominan métricas de laboratorio o *in the lab* [Wal19]. Pese que ambos tipos de mediciones son necesarias, sólo seremos capaces de tener una visión completa de la UX de un sitio web si obtenemos métricas *in the field* de usuarios reales.

Han sido muchas las métricas que se han propuesto para cuantificar estos efectos. En sus inicios, las tecnologías de RUM se centraron en un única métrica: *el tiempo de carga de la página*. Esta medida tenía un elevado coste computacional y se centraba únicamente en la perspectiva del navegador y no del usuario. A finales de los 90 y principios de 2000 surgieron proveedores de herramientas de RUM. Grandes compañías como *IBM*, *HP* y *BMC* decidieron invertir en este tipo de herramientas y así surgieron las soluciones de *Tealeaf, BeatBox* y *Coradiant* respectivamente [Sny15].

La definición de la interfaz de *Navigation Timing* en 2012 y su posterior implementación por todos los navegadores modernos, supuso un punto de inflexión para la RUM/EUEM. Esta API habilitó el acceso a tiempos de carga a través de más de 20 atributos permitiendo ilustrar de una manera mucho más precisa cómo contribuyen los diferentes tiempos de carga a la experiencia final de usuario en una web. A pesar de haberse adoptado como el estándar RUM en los últimos años, los resultados que reporta están muy centrados en el navegador. La tendencia actual es buscar métricas orientadas al usuario y que sean realmente valiosas para medir la UX. *Google* está inviertiendo muchos recursos en el desarrollo de este tipo de métricas que se denominan *user-centered performance metrics* [Wal19].

2.4.2. *Navigation Timing API*

Esta API, está basada en la **Recomendación de W3C "*Navigation Timing Specification*"** que define una interfaz para las aplicaciones web para tener acceso a datos relacionados principalmente con los tiempos de navegación. Esta especificación se basa en mecanismos *JavaScript* que recogen los valores adecuados para obtener una completa visión de la latencia percibida desde el lado del cliente [W3C12].

En la Figura 2.4 se representa el desglose de los tiempos que se pueden medir en la carga de una página gracias a los attributos definidos por la interfaz en cuestión.

La valiosa información que permite obtener esta API, junto al extendido soporte que tiene entre todos los navegadores, han contribuido a que, a día de hoy, siga siendo la herramienta más utilizada en la medición del rendimiento de una página. Sin embargo, tiene ciertas limitaciones que provocan la definición de nuevas métricas. Su punto más crítico es que sus resultados están centrados en el navegador, pero en muchos casos no son significativos para el usuario.

2.4.3. Nuevas Tendencias

Con el objetivo de acercarnos a un mundo en el que poder medir el rendimiento de una forma que sea significativa para los usuarios de la web, la industria se está alejando de los tiempos de navegación orientándose hacia nuevas métricas centradas en el usuario que reflejen mejor su UX.

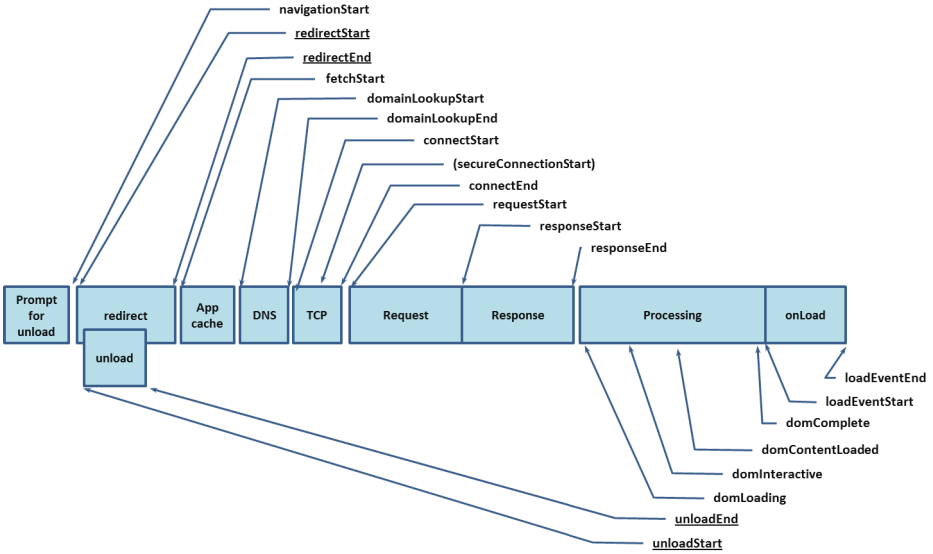

Figura 2.4: Desglose de los tiempos que mide la interfaz durante la carga de una página web [W3C12].

En la búsqueda de encontrar aquellas métricas que mejor representen la Experiencia de Usuario han surgido diferentes participantes. Por un lado tenemos a *WebPageTest*, una herramienta de medición de rendimiento de páginas web que en 2012 añadió la métrica de ***Speed Index*** entre sus resultados. El punto diferencial de esta métrica es que mide el progreso de carga de una página desde una perspectiva visual (mucho más relevante para el usuario final). Cuantifica el resultado a través de una puntuación que será más positiva cuanto menor sea su valor [Web]. Se trata de uno de los índices que más se tienen en cuenta en la actualidad al evaluar una página web.

El último participante en entrar al juego ha sido *Google* con la reciente definición de sus ***Core Web Vitals***. Se trata de un subcojunto de indicadores dentro de su iniciativa *Web Vitals* que tiene como objetivo simplificar el panorama de evaluación de rendimiento de las páginas web creando métricas que midan lo que más importa a los usuarios, las mencionadas *Core Web Vitals* [Wal20].

Como vemos en la Figura 2.5 *Google* define 3 métricas que pueden ir evolucionando con el tiempo. Para cada métrica fija 2 umbrales de rendimiento de la web y, como vemos, el rendimiento será mejor cuanto más bajos sean las puntuaciones de cada métrica. Cada una de ellas se centra en un aspecto de la UX [Wal20]:

- ***Largest Contentful Paint*** **(LCP)**: métrica cetrada en la *velocidad percibida de carga* que mide el tiempo que tarda en renderizar el elemento de mayor tamaño.

- ***First Input Delay*** **(FID)**: indicador de la *interactividad* del usuario con la página a través de la medición del tiempo que tarda en responder el navegador ante el primer intento de interacción.

- ***Cumulative Layout Shift*** **(CLS)**: mide la *estabilidad visual* de la página mediante

una puntuación que será mejor cuantos menos cambios inesperados ocurran en la distribución de los elementos visuales.

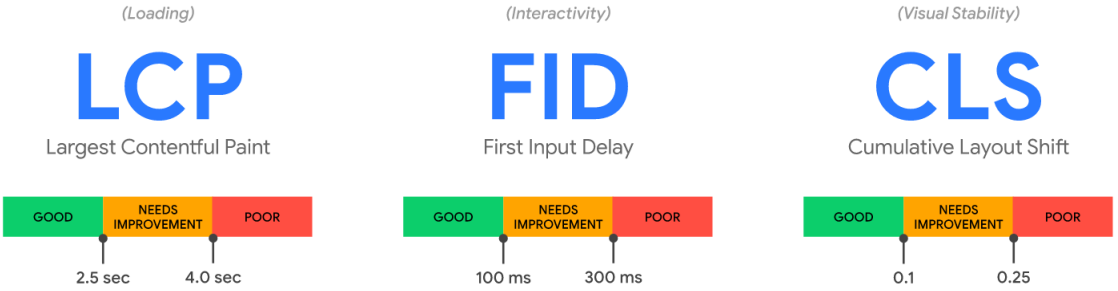

Figura 2.5: Core Web Vitals y los umbrales fijados por *Google* [Wal20].

A pesar de su reciente aparición en escena, estás métricas han captado el interés de muchas empresas. El primer motivo es el atractivo que tiene que estén orientadas plenamente a la Experiencia de Usuario y su afán de simplificación. El segundo, es el indudable impacto que tiene el haber sido desarrolladas por *Google*, el principal motor de búsqueda de todo Internet. Y no es para menos, pues desde la propia entidad han anunciado que ha pasado a ser un factor oficial de *ranking* de *Google* desde de mayo 2021. Esto afecta directamente al posicionamiento y las campañas de las empresas en la web. Sin embargo, la principal limitación es que son métricas que, a día de hoy, sólo pueden medirse en *Chrome*. Esto deja fuera del análisis RUM al resto de usuarios de la red que no utilizan este navegador. Y esto es un problema de sesgo a considerar.

2.5. Discusión y conclusiones

Nos encontramos en un momento en el que la automatización de procesos dentro de una empresa es uno de los puntos en los que más están invirtiendo los empresarios. Prueba de ello es la enorme expasión que experimenta la industria de RPA de unos años a esta parte. Los sistemas RPA necesitan de la combinación de diferentes herramientas para su diseño. Entre todas estas destacan los *software* de *web-scrapping* que son herramientas increíblemente potentes para la extracción de datos de Internet.

En la misma línea que los RPA, los sistemas de Inteligencia de Negocio pretenden agilizar procesos. En este caso, aquellos relacionados con la toma de decisiones de la empresa. Por esto, cuando combinamos las tecnologías de RPA con la de BI con un propósito común, la unión de ambas conduce a un todo que es mayor que la suma de sus partes.

Por otro lado, se ha analizado la importancia de la Experiencia de Usuario (UX) en el rendimiento de páginas web, concretamente en aquel que se puede medir a través de las interacciones de usuarios reales (RUM). Para la recopilación de estos datos se suelen utilizar herramientas de análisis de sesiones como *Tealeaf*. El rendimiento percibido por los consumidores en una página web influye directamente sobre las conversiones que en

ella se producen, repercutiendo en última instancia en las rentabilidades e ingresos de la empresa. Por ello, es posible encontrar una relación entre estos datos de rendimiento y la Inteligencia de Negocio, por ejemplo a través del sistema de Administración de Relaciones con el Cliente (CRM) o mediante la creación de *dashboards*. Además, los procesos de recogida o análisis de estos datos son susceptibles de ser automatizados a través la RPA.

Teniendo en cuenta todo lo anterior, en el presente TFG se propone la creación de una solución RPA que permita consumir a nivel interno *dashboards* que presenten datos que proceden del análisis de sesiones de *Tealeaf*. Estos datos disponibles en la web se recopilarán de manera automática mediante la técnica de *web-scraping*, además se procesarán y publicarán en una base de datos que será la fuente utilizada por los *dashboards*. Estos paneles de control son considerados herramientas de BI que aportarán información referente a: (i) el consumo de licencias del *software* de *Tealeaf* entre los diferentes clientes de la empresa y (ii) datos (RUM) de rendimiento web proporcionados por la API *Navigation Timing* y medido en sesiones reales a través de *Tealeaf*. El cumplimiento de estos requisitos contribuirá, por un lado, a una mejora en la productividad de la empresa por la agilización de los procesos y, por otro lado, a una mejora en la calidad de la información disponible para el Equipo de Negocio a nivel cuantitativo y cualitativo. En los próximos capítulos se explicará en detalle el análisis y diseño de este sistema.

Capítulo 3

Planificación y Análisis del sistema

3.1. Introducción

En este Capítulo se refleja todo el contenido referente a las fases iniciales del Ciclo de Vida del Desarrollo de Sistemas descritas en la metodología del Capítulo 1. Concretamente las fases de Planificación y Análisis.

La **Planificación** del sistema se aborda en la Sección 3.2 de este Capítulo. En este apartado se resumen las actividades de identificación y selección del proyecto. Se propone un primer planteamiento del problema a resolver y se responde a las necesidades principales que debe cubrir el sistema. Al final de la Sección se definirán, por primera vez, las dos soluciones diferentes que se van a desarrollar.

La fase de **Análisis** se recoge en los sucesivos apartados del Capítulo. La Sección 3.3 está dedicada a la definición formal de los requisitos funcionales y no funcionales de los sistemas. Toda esta información se organiza en la Sección 3.4 a través de los Diagramas de Flujo de Datos que representan los sistemas de más alto a más bajo nivel.

3.2. Visión del sistema

La **Planificación** consitituye la fase inicial del SDLC. Principalmente abarca las actividades de **identificación y selección** de potenciales proyectos de desarrollo [JSV17]. En lo que a este proyecto respecta, ya se ha ahondado ampliamente en esta fase en los las secciones 1.1, 1.2 y 2.5. A modo de resumen, se recuerda que el proyecto se ha desarrollado durante las prácticas en una empresa tecnológica. Su puesta en marcha vino motivada originalmente por la existencia de *Puppeteer*, un *software* que proporcionaba la posibilidad de implementar una RPA para agilizar procesos de la empresa. Concretamente aquellos que tienen que ver con la recopilación y visualización de grandes cantidades de datos por parte del sistema de *Business Intelligence*. Además, el proyecto se seleccionó porque el desarrollo y aprendizaje de estas tecnologías generaría una oportunidad de negocio al poder comercializar este tipo de servicios entre los clientes de la empresa.

En esta fase inicial, se realizó un primer acercamiento al problema que se pretende solucionar y que se representa en la Figura 3.1. Mediante el desarrollo del sistema RPA, se recopilan, procesan y almacenan datos de la visibles en la web de *Tealeaf* y se generan informes para la visualización, análisis e interacción de los datos.

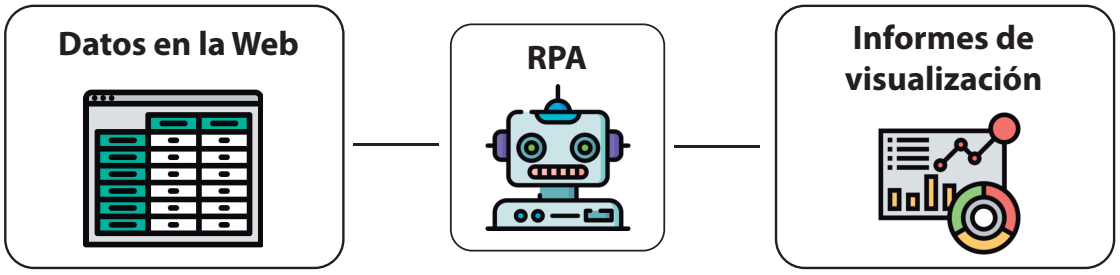

Figura 3.1: Visión del problema. Elaboración propia.

La pregunta más importante que debemos hacernos durante la planificación es: ***¿cuál es el problema?*** O dicho de otra forma: *¿por qué no podemos consumir los datos directamente desde la web de Tealeaf?* Y la respuesta es que si bien es cierto que *Tealeaf* recopila y permite visualizar los datos, presenta ciertas limitaciones:

- Tiene un límite de filas representadas en sus tablas.
- Operaciones de agregado de datos muy limitadas.
- Número muy restringido de tipos de gráficos.
- Escasa personalización de los gráficos.
- Escasa personalización de generación de informes.
- Dificultad para realizar filtrado de datos.

Todas estas limitaciones, no las encontramos en herramientas específicamente dedicadas a la visualización como en nuestro caso es *Data Studio*. Por lo tanto, el objetivo del sistema será poder extraer los datos de la web para visualizarlos en una herramienta de visualización que nos ofrezca todas las funcionalidades de análisis sobre los datos.

Existen otras cuestiones importantes relativas al sistema que deben resolverse en esta fase y que nos ayudarán a definir los requisitos en la Sección 3.3. Estas preguntas son:

1. *¿Quiénes van a ser los **usuarios**?*: los usuarios son las personas que van a consumir los informes de visualización. Habitualmente los informes estarán dirigidos a:
   - Personal del equipo de negocio encargados de tomar decisiones.
   - Analistas de datos.
   - Otros departamentos.
   - Potencialmente (si se comercializa esta solución) los usuarios serán los clientes.

2. *¿Qué tipo de **datos** se van a recopilar?*: los datos se van a extraer de tablas y gráficos que se muestran en la web de *Tealeaf* de la empresa. Concretamente vamos a implementar la solución para la extracción de dos tipos de datos:

   - Consumo mensual de licencias de *Tealeaf* por parte de la empresa.
   - Datos de rendimiento de un sitio web recopilados en *Tealeaf*.

3. *¿Cómo se utiliza el sistema?*: al ser un sistema automatizado, el *bot* se ejecutará como una tarea programada a diario. De esta forma, los datos de los informes de visualización se actualizarán cada día de forma transparente para sus usuarios y sin la necesidad de intervención humana.

4. *¿Cómo **interactúan** los usuarios con el sistema?*: los usuarios, a través de cualquier dispositivo con acceso a Internet, podrán visualizar los informes e interactuar con los datos por ejemplo mediante aplicación de filtros o creando nuevos gráficos.

Como hemos visto, se implementarán dos soluciones diferentes en función del tipo de datos que queremos analizar. Todos los fundamentos descritos hasta este punto del informe son comunes a ambas soluciones. Sin embargo, existen ciertas diferencias entre ambos casos que harán preciso diferenciarlos sobre todo en las próximas fases de Análisis, Diseño e Implementación. Por todo ello, de ahora en adelante y sólo cuando sea necesario, distinguiremos ambos sistemas con la siguiente denominación:

- **Sistema de Licencias**: solución RPA para extracción de datos del **consumo** de licencias de *Tealeaf* de las distintas organizaciones que tiene contratada la empresa. El objetivo es mostrar una gráfica con el consumo acumulado de cada organización frente al máximo de licencias contratadas. De esta forma se llevará un histórico del consumo mensual y, además, servirá de alerta para los administradores en caso de superar el máximo.

- **Sistema de Rendimiento**: solución RPA para extracción de datos de rendimiento de un sitio web asociado a la empresa. El objetivo es elaborar informes que muestren los **tiempos** de carga experimentados por usuarios reales medidos con *Tealeaf* a través de la API de *Navigation Timing*. Estos datos se tienen que poder filtrar por *fecha*, *país* y *plataforma* (ordenador o móvil) en todas sus combinaciones posibles.

3.3. Definición de requisitos

Siguiendo con el Ciclo de Vida del Desarrollo de Sistemas, la siguiente fase a abordar es la del **Análisis**, que tiene como propósito determinar qué información y qué servicios de procesamiento de información son necesarios para apoyar los objetivos y funciones seleccionados [JSV17]. A su vez, podemos definir dos subetapas dentro del Análisis:

1. **Definición de Requisitos**. A la que dedicamos esta Sección. Consiste en la recopilación de las características que debe ofrecer nuestro sistema.

2. **Estructuración de Requisitos**. A la que dedicamos la Sección 3.4. En esta subetapa se producen las actividades de organización de toda la información (en forma de requisitos) generada en la subetapa anterior. Para tal efecto haremos uso de la representación del Diagrama de Flujo de Datos (DFD).

Fruto de distintas reuniones con los usuarios finales en las primeras semanas del proyecto, se recopiló toda la información en torno a sus necesidades y las características que debían tener las diferentes soluciones del sistema. A partir de esa información, se especifican los **Requisitos Funcionales** (FR, *functional requirement*) y los **Requisitos No Funcionales** (NFR, *non-functional requirement*). Ambos tipos de requisitos responden a las necesidades que tiene el sistema en torno a su *funcionalidad*, *fiabilidad*, *rendimiento*, *soporte*, *diseño*, *implementación*, *integración* y *hardware* [Ste15].

3.3.1. Requisitos Funcionales

Los **Requisitos Funcionales** son descripciones detalladas de las capacidades deseadas del proyecto [Ste15]. En la Tabla 3.1 se recopilan los requisitos funcionales indicando, además, si se aplican al *Sistema de Licencias*, al *Sistema de Rendimiento* o a ambos.

Tabla 3.1: Requisitos Funcionales de los dos sistemas.

Requisitos Funcionales			
ID	Descripción	Licencias	Rendimiento
FR01	El sitema deberá superar el proceso de *login* de *Tealeaf* y su barrera de doble autenticación.	Sí	Sí
FR02	El sistema deberá simular una navegación automática hasta el lugar de la web donde se muestran los datos.	Sí	Sí
FR03	El sistema deberá extraer de manera automática los datos de interés la web.	Sí	Sí
FR04	El sistema deberá procesar internamente los datos extraídos. Deberá estructurarlos y eliminar aquellos que no necesita.	Sí	Sí
FR05	El sistema deberá ejecutarse de manera automática según un período deseado.	Diario	Semanal
FR06	El sistema deberá actualizar los datos de manera estructurada mediante campos en un *spreadsheet*.	Sí	Sí
FR07	El sistema deberá almacenar en el *spreadsheet* un histórico de todos los datos.	No	Sí
FR08	El sistema deberá comparar los datos extraídos con los del *spreadsheet* y actualizar aquellos campos que hayan cambiado.	Sí	No

Continuación de la Tabla 3.1.			
ID	Descripción	Licencias	Rendimiento
FR09	El sistema deberá mostrar a los usuarios informes con gráficas con los datos extraídos.	Sí	Sí
FR10	El sistema deberá actualizar los informes de visualización de manera simultánea a las actualizaciones de la fuente de datos (*spreadsheet).*	Sí	Sí
FR11	El sistema deberá tener la opción de exportar los informes de visualización en formato PDF.	Sí	Sí
FR12	El sistema deberá incluir unos parámetros de configuración de la ejecución. Los podrá modificar el desarrollador a través del fichero *config.json*	Sí	Sí
FR13	El sistema deberá permitir a los usuarios que tengan acceso a los informes e interactuar con los datos a través de filtros.	No	Sí
FR14	El sistema deberá permitir editar las gráficas de los informes a aquellas personas que tengan permiso de edición.	Sí	Sí
FR15	Los informes de visualización han de incluir filtros por fecha, país y plataforma (ordenador o móvil). Y todas las posibles combinaciones que existan entre ellos.	No	Sí
FR16	El sistema deberá poder desplegarse en una máquina de los sistemas internos de la empresa.	Sí	Sí
FR17	El sistema deberá poder ejecutarse en un servidor en modo *headless*. Esto es, en un entorno sin interfaz gráfica.	Sí	Sí

3.3.2. Requisitos no Funcionales

Los **Requisitos No Funcionales** son descripciones sobre la calidad de comportamiento del sistema o sobre sus restricciones a la hora de producir un resultado. Con carácter general, se centran en las características de *rendimiento*, *fiabilidad* y *seguridad* [Ste15]. En la Tabla 3.2 se recopilan los requisitos no funcionales del sistema.

Tabla 3.2: Requisitos No Funcionales de los dos sistemas.

Requisitos No Funcionales			
ID	Descripción	Licencias	Rendimiento
NFR01	El sistema deberá tener un control de permisos de propiedad, edición y visualización sobre los informes.	Sí	Sí
NFR02	El sistema deberá ejecutarse de manera que su rendimiento no afecte negativamente a la memoria de la máquina.	Sí	Sí
NFR03	El sistema deberá ser robusto frente a errores.	Sí	Sí
NFR04	El sistema deberá incluir distintos niveles de LOG para poder hacer un seguimiento del proceso de ejecución.	Sí	Sí
NFR05	La interacción con los informes de visualización deberán ser intuitivos.	Sí	Sí
NFR06	La información mostrada en los informes de visualización debe poder ser entendida por cualquier usuario, aunque no tenga conocimientos técnicos.	Sí	Sí
NFR07	Los informes de visualización deberán ser accesibles a través de cualquier dispositivo móvil, tableta u ordenador con acceso a internet.	Sí	Sí
NFR08	La fuente de datos (*spreadsheet*) sólo podrá ser modificada por el sistema y su propietario.	Sí	Sí

3.4. Diagramas de Flujo de Datos

La segunda subetapa del proceso de Análisis es la **Estructuración de los Requisitos** definidos en la Sección 3.3. Para ello haremos uso de los **Diagramas de Flujo de Datos** (DFD, *Data Flow Diagram*) que son una herramienta que permite representar de manera coherente la información recopilada en los requisitos [JSV17]. El análisis a través de DFDs permite:

- Modelar cómo fluyen los datos a través del Sistema de Información.
- Representar las relaciones entre los flujos de datos.
- Describir cómo los datos llegan a almacenarse en diferentes lugares.
- Mostrar los procesos que transforman los datos.

Estos DFDs se llaman **modelos de proceso** porque concentran el movimiento de datos

entre distintos procesos [JSV17]. En nuestro caso, el flujo de datos se lleva a cabo mediante un sistema *software* de RPA.

Antes de representar los DFDs para nuestras dos soluciones, se explica qué representa cada elemento del diagrama y cómo se interpretan. Precisamente, uno de los atractivos que tienen estos diagramas es que son muy sencillos de interpretar, pues sólo se utilizan 4 símbolos distintos. Existen diferentes estándares respecto a esta simbología, pero en este informe se utilizan los símbolos propuestos por Gane y Sarson [CG79] que se representan en la Figura 3.2.

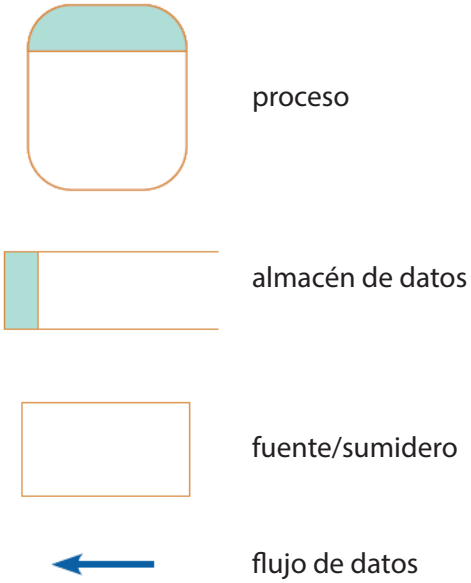

Figura 3.2: Símbolos de Gane y Sarson para representar DFDs [JSV17].

Se repasa a continuación qué representa cada elemento:

- **Flujo de datos**. Representa datos en movimiento desde un lugar del sistema a otro. Tiene forma de flecha y se etiqueta con un nombre significativo del movimiento de datos.
- **Almacén de datos**. Representan datos en reposo dentro del sistema a través de un rectángulo al que le falta el lado derecho. En la caja izquierda se numera y en la parte derecha se utiliza un nombre significativo para el almacén.
- **Proceso**. Representan acciones realizadas sobre los datos. Pueden ser operaciones de *transformación*, *almacenamiento* o *distribución*. Su símbolo es un rectángulo con esquinas redondeadas. En la parte superior se indica el número del proceso y en la parte inferior su nombre.
- **Fuente/Sumidero**. Son el origen y/o destino de los datos. Son entidades externas que están fuera del sistema. Se representan con un rectángulo y tienen el nombre del agente externo al que definen.

Además se ha de tener en cuenta que los DFDs tienen que seguir ciertas reglas en su representación que se recogen en la Figura 3.3.

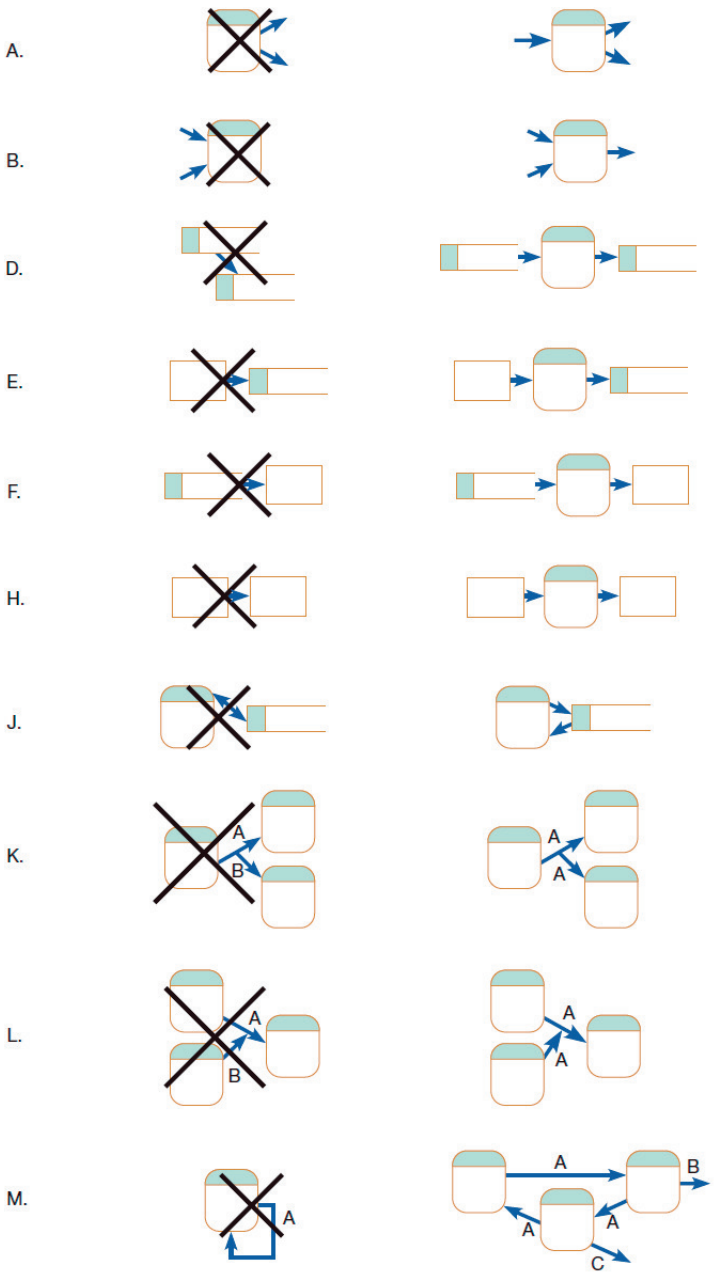

Figura 3.3: Reglas de elaboración de los Diagramas de Flujo de Datos [JSV17].

En las siguientes subsecciones se representan los Diagramas de Flujo de Datos creados durante la etapa de Análisis del proyecto. Serán de gran utilidad para entender los sistemas y los analizaremos de manera iterativa comenzando por diagramas alto nivel y profundizando cada vez más en los detalles. A este proceso se le denomina **descomposición funcional**, que consiste en fragmentar la perspectiva del sistema en partes cada vez más concretas [JSV17].

3.4.1. Diagrama de contexto

El **Diagrama de contexto** representa la vista de todo el sistema en su conjunto al **más alto nivel**. Esta representación del flujo de datos tiene un **un único proceso**, etiquetado con un 0. Las **fuentes/sumideros** representan las fronteras del sistema. **No** se representan **almacenes de datos** porque conceptualmente están dentro del proceso.

En realidad, el resultado del Diagrama de Contexto será similar a la representación del problema que vimos en la Figura 3.1. Tanto en esta representación de más alto nivel como en los niveles sucesivos, vamos a separar los diagramas para los dos soluciones que vamos a desarrollar. Por un lado representaremos los diagramas referentes al *Sistema de Licencias* y por otro los diagramas referentes al *Sistema de Rendimiento*. Esto es lógico, ya que la principal diferencia entre ambos sistemas son los datos con los que trabajan.

A continuación, se muestran los Diagramas de Contexto de los dos sistemas que son objeto de estudio. En la Figura 3.4 se representa el *Sistema de Licencias* al más alto nivel. Lo mismo ocurre con el *Sistema de Rendimiento* en la Figura 3.5. Se han separado porque, ya en este nivel, existen diferencias:

- Respecto a las fronteras de ambos sistemas, **la fuente** es la misma, etiquetada como *Web de Acoustic*, pero vemos que **los sumideros** son distintos. En el caso de la Figura 3.4 los datos se proporcionan al *Equipo de Negocio*, mientas que en la Figura 3.5 se muestra como los datos son consumidos por el equipo de *WPO (Web Performance Optimization)*.

- Como ya hemos notado, los datos con los que trabajan ambos sistemas son diferentes y por eso vemos esas diferencias en sus **flujos de datos**. En *Sistema de Licencias* trabaja con datos de los consumos de licencias medidos en Millones de Interacciones Mensuales (MMI), mientras que el sistema de rendimiento trabaja con tiempos promedios de carga de los usuarios en la web.

- Además, el nombre del **proceso único** de cada diagrama es distinto porque engloba toda la funcionalidad de cada sistema y, como veremos en los siguientes niveles, estos subprocesos son distintos entre sistemas.

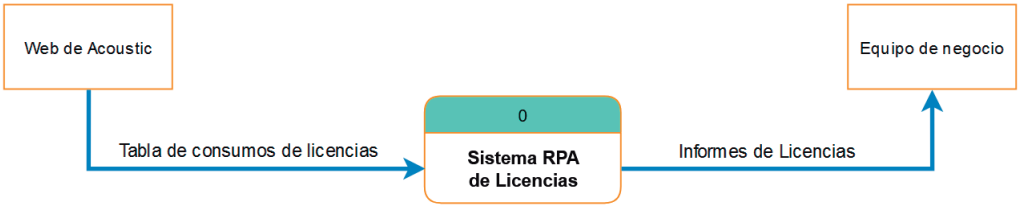

Figura 3.4: Diagrama de contexto del *Sistema de Licencias*.

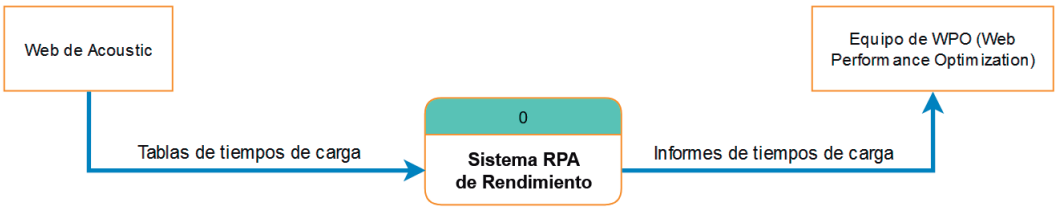

Figura 3.5: Diagrama de contexto del *Sistema de Rendimiento*.

3.4.2. DFD de nivel 0

El siguiente nivel que vamos a analizar son los DFDs de nivel 0. Se llaman así porque representan las funciones principales del sistema al más alto nivel posible. Se construyen separando el único proceso (etiquetado con un 0) del Diagrama de Contexto en ***n* procesos distintos**. Estos procesos representan las funciones principales del sistema y se etiquetan numerándolos como ***i.0*** con *i=1,2,...,n.*.

La buena descripción de estos Diagramas de nivel 0 es una parte fundamental en la fase de Análisis del sistema. Se debe determinar **qué procesos están representados por el proceso único del Diagrama de Contexto**. Como norma general, cada DFD (sea del nivel que sea) no debe contener más de 7 procesos y debe ser mostrado en una página a parte [JSV17].

En la Figura 3.6 se muestra el Diagrama de Flujo de Datos de nivel 0 del *Sistema de Licencias*. Hay que notar que tanto **la fuente** como el **sumidero** coinciden con aquellos que se especificaron en el Diagrama de Contexto. También se mantienen los **flujos de datos** que salen y entran de ellos. En este diagrama se han detectado 5 funciones principales del *Sistema de Licencias* que se representan con los **5 procesos**:

1. **Acceder y extraer datos de consumos de Licencias**.
2. **Formatear** los datos extraídos.
3. **Leer los datos de la tabla del histórico**. Además, esta Tabla de datos se representa como el único **almacén de datos** del sistema (D1).
4. **Comparar y actualizar datos** de distintos orígenes.
5. **Transformar y representar** los datos para elaborar los Informes.

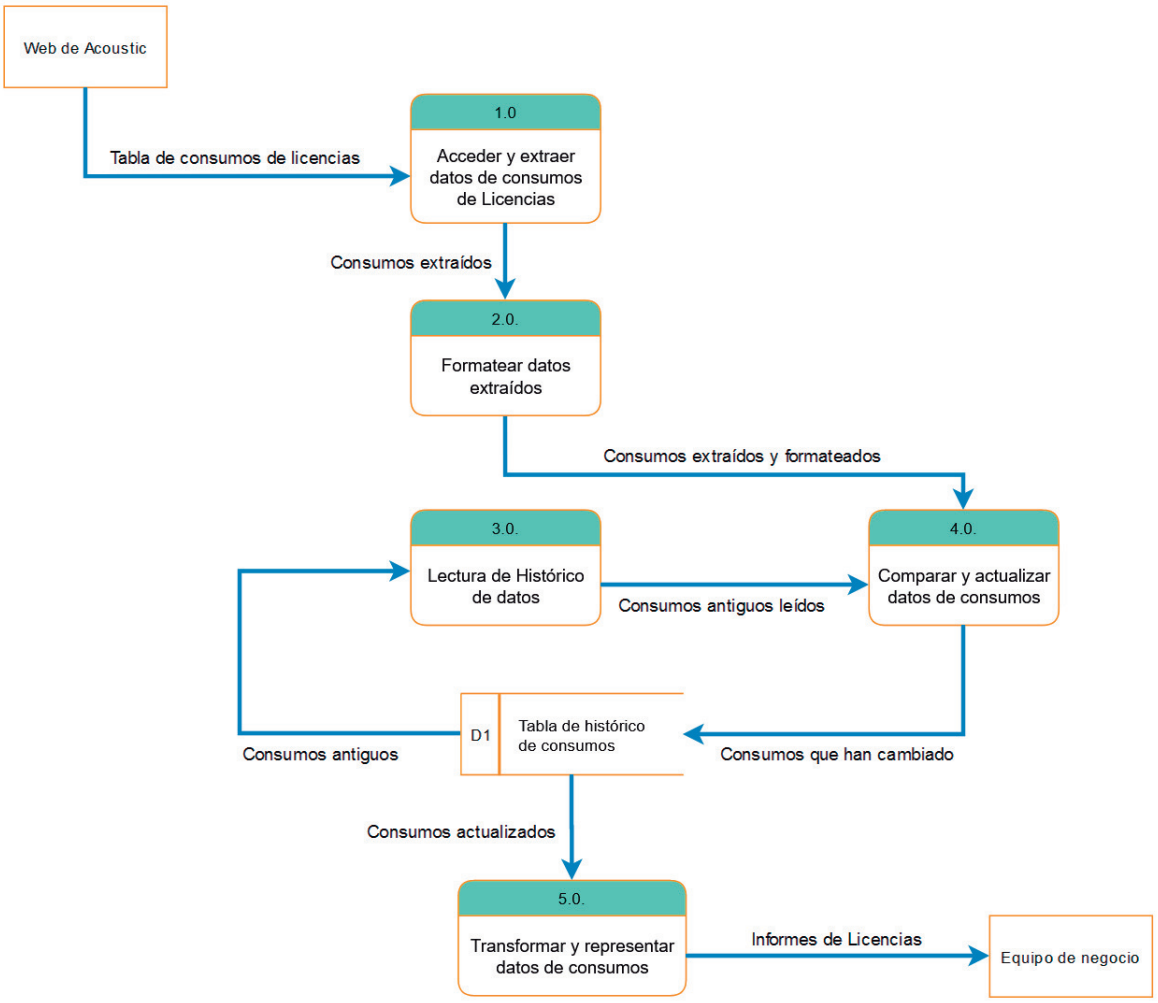

Figura 3.6: DFD de nivel 0 del *Sistema de Licencias*.

El DFD de nivel 0 para el *Sistema de Rendimiento* se representa en la Figura 3.7. En este caso, el proceso único que teníamos en el Diagrama de contexto se divide en **4 procesos** o funciones principales. Algunas de ellas similares a las del *Sistema de Licencias* (como los procesos 1.0, 2.0 y 4.0) y otras distintas como el proceso 3.0, que representa la separación de los datos extraídos en dos flujos según su naturaleza. Por lo tanto, en este sistema se necesitará mantener **dos almacenes de datos** distintos. Uno para el **desglose de *tiempos generales***, que son aquellos que se representaban en la Figura 2.4, y otro para el **desglose de *tiempos de render***[1].

[1]Tiempos que componen el tiempo de *Processing* representado en la Figura 2.4. Se mide la composición de este tiempo porque es el más crítico de todos los que contribuyen al tiempo de carga de una página.

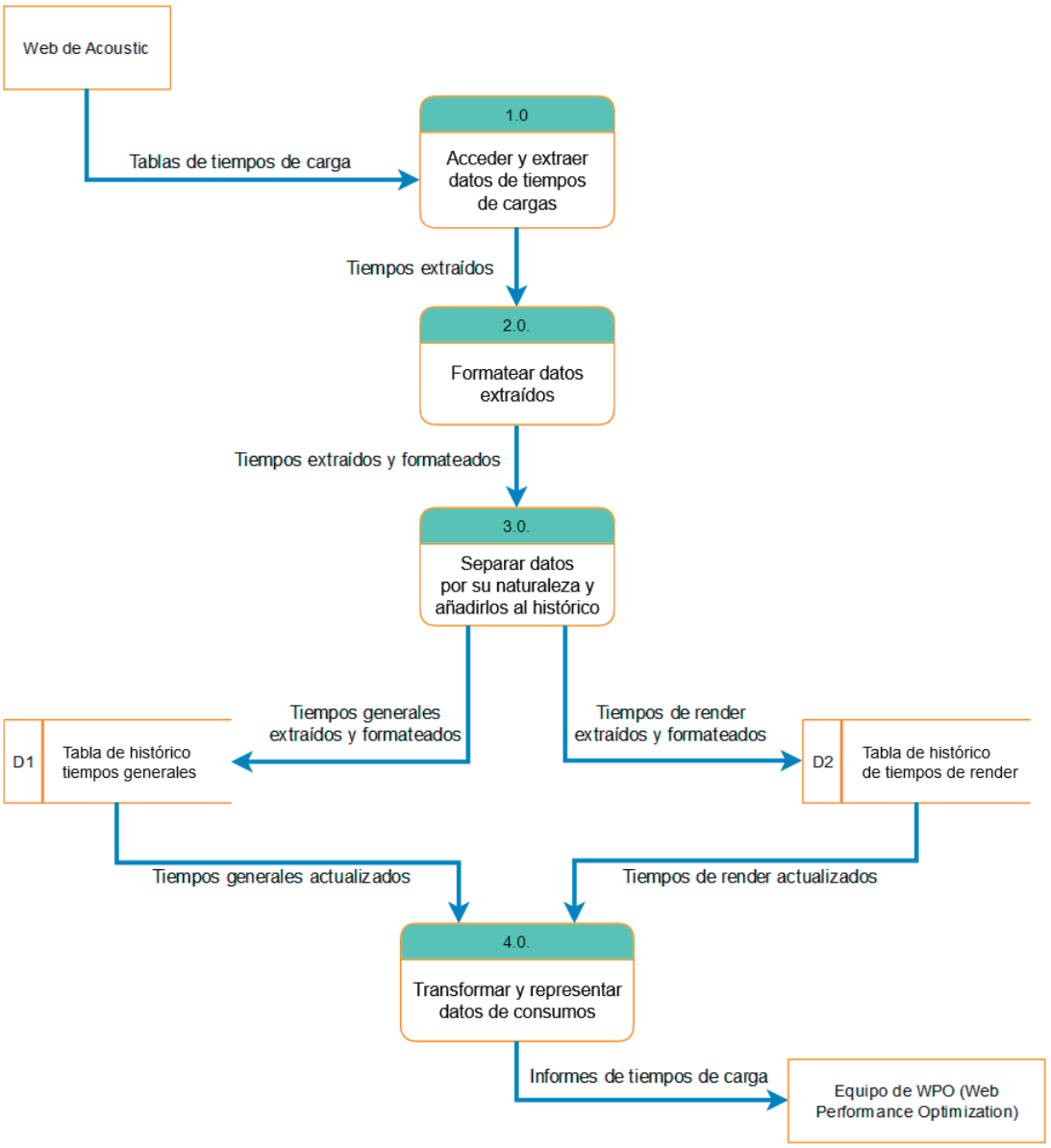

Figura 3.7: DFD de nivel 0 del *Sistema de Rendimiento*.

3.4.3. DFDs de nivel 1

Del mismo modo que el Diagrama de Flujo de Datos de nivel 0 surge como una descomposición del proceso representado en el diagrama de nivel superior (el Diagrama de Contexto), cada uno de los procesos del sistema de nivel 0 son susceptibles desglosarse en nuevos procesos constituyendo nuevos DFDs por cada proceso, que serían los **Diagramas de Flujo de Datos de nivel 1**. A este tipo de Análisis nos hemos referido como descomposición funcional. Así pues, el **diagrama de nivel *n*** es un DFD que representa el resultado de la *n-ésima* descomposición anidada de un proceso del diagrama de nivel 0 [JSV17]. Previo a la elaboración de los diagramas de este nivel, se han de tener en cuenta las siguientes consideraciones:

- No es preciso descomponer a nivel 1 todos los procesos de nivel 0. Es posible que algunos procesos de nivel 0 representen el nivel más bajo de abstracción.
- Al descomponer, se ha de seguir el **principio de Balanceo**. Esto implica que los flujos de entrada y salida de un proceso se han de conservar en el siguiente nivel.
- **No** es necesario representar los **almacenes de datos**, **sumideros** y **fuentes**, pues ya se reflejan en el nivel 0.

A continuación, se van a mostrar los DFDs de nivel 1 que se han considerado necesarios para el análisis del ***Sistema de Licencias***.

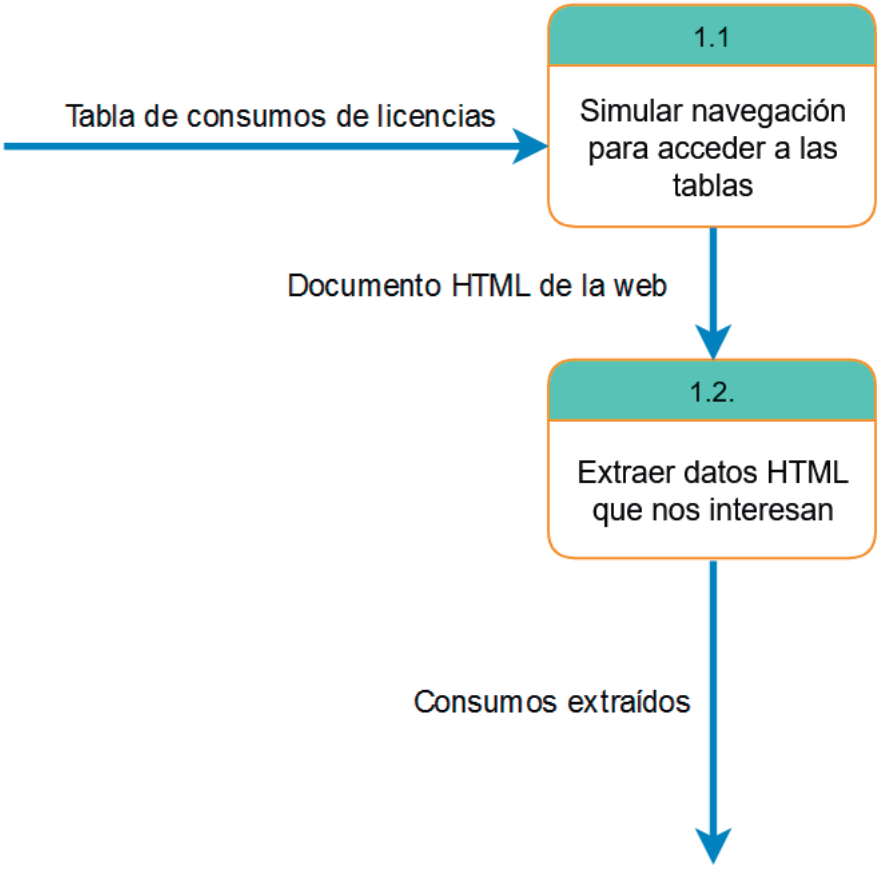

Figura 3.8: DFD de nivel 1 del proceso 1 del *Sistema de Licencias*.

En la Figura 3.8 se representa el DFD de nivel 1 para el proceso 1 del *Sistema de Licencias*. Este proceso, se descompone en dos subprocesos. El primero de ellos (1.1), abstrae la funcionalidad de **simular la navegación** hasta el acceso a la página que contiene los datos de consumo de licencias de la web de *Tealeaf*. El segundo (1.2), representa la acción de extraer del HTML de la página los datos que nos interesan para el análisis.

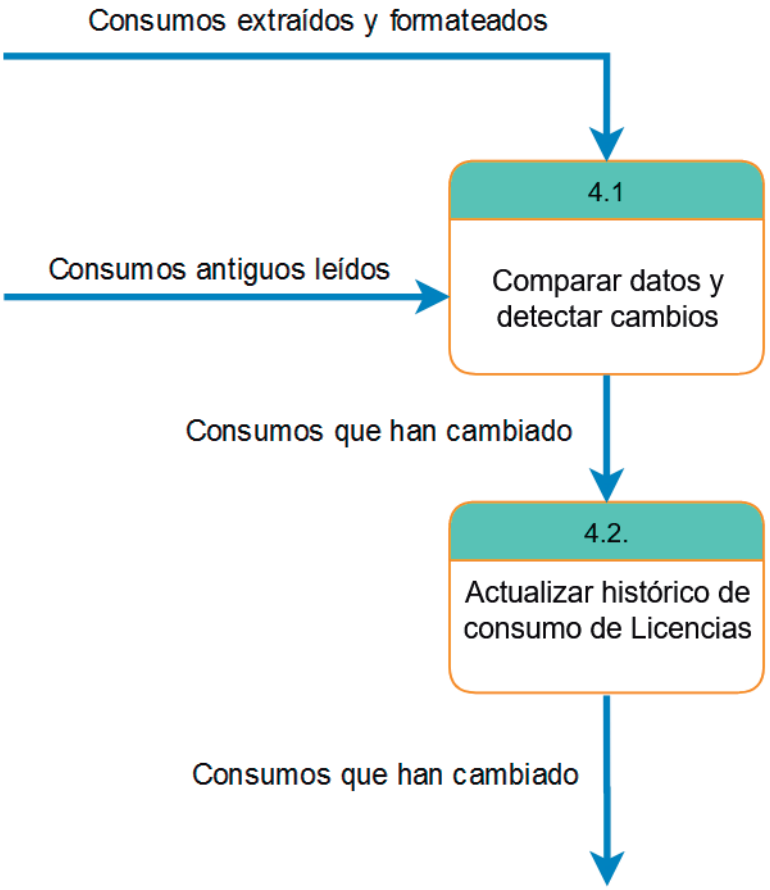

Figura 3.9: DFD de nivel 1 del proceso 4 del *Sistema de Licencias*.

Se considera que los procesos 2 y 3 representados en el DFD de nivel 0 del *Sistema de Licencas* de la Figura 3.6 no necesitan una descomposición mayor al tratarse de funciones simples como son formatear datos o actualizar una tabla. Sin embargo, el proceso 4 sí que precisaba de una descomposición de nivel 1 cuyo DFD se representa en la Figura 3.9.

El proceso original se descompone en dos procesos con funciones bien diferenciadas:

1. **Comparar** datos para detectar cambios. Al proceso 4.1. llegan dos **flujos de datos**. Por un lado, los consumos antiguos que se leen del histórico y, por otro lado, los consumos extraídos de la web y posteriormente formateados. En este proceso, el sistema ha de comparar ambos flujos de entrada y proporcionar como **flujo de salida** los valores de los datos que resultaban distintos tras la comparación.

2. **Actualizar** el histórico de datos con los valores del que han cambiado y que son tanto el flujo de entrada como el flujo de salida del proceso. No se representa, pero esos valores tienen como destino la **Tabla de Histórico de consumos** que se representaba en la Figura 3.6.

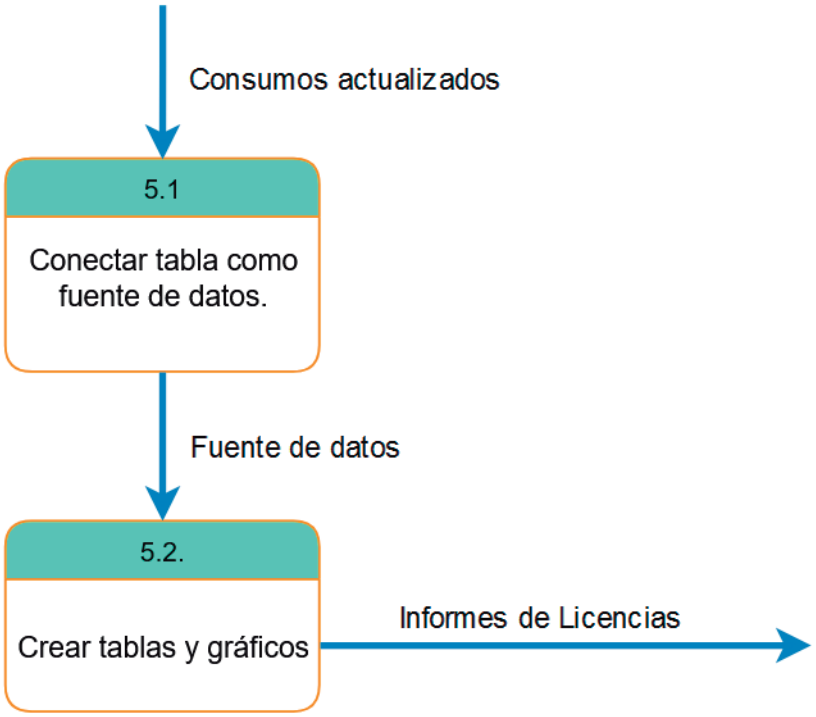

Figura 3.10: DFD de nivel 1 del proceso 5 del *Sistema de Licencias*.

Como vemos en la Figura 3.10, el proceso 5 del DFD de nivel 0 del *Sistema de Licencias* también puede ser desagregado en dos funcionalidades independientes:

1. **Conectar** la tabla como **fuente de datos**. Esta actividad (5.1.), hace referencia al proceso que tiene que llevar a cabo el sistema para utilizar la **Tabla de Histórico de consumos** como fuente de datos para poder construir herramientas de visualización. Por lo tanto se hace evidente el doble papel que juega esta Tabla. Por un lado, representa el destino de los datos extraídos y, por otro lado, representa el origen de los datos representados. Saber gestionar bien esta tabla será de vital importancia para el buen funcionamiento del sistema.

2. **Crear tablas y gráficos**. Las tablas y gráficos son creadas por el desarrollador del sistema, pero el proceso 5.2. abstrae las operaciones que el sistema tiene que realizar internamente (sin intervención del desarrollador) para mantener las tablas y gráficos al día ante las actualizaciones automáticas de la fuente de datos.

Con esta gráfica finalizaríamos el análisis del *Sistema de Licencias* a través de los Diagramas de Flujo de Datos. Se considera que todos los procesos representados hasta este punto no necesitan mayor descomposición funcional. Esto quiere decir que se ha alcanzado el nivel lógico más bajo necesario, al menos para la fase de Análisis. A este nivel se lo conoce como **Diagrama de Flujo de Datos Primitivo** [JSV17].

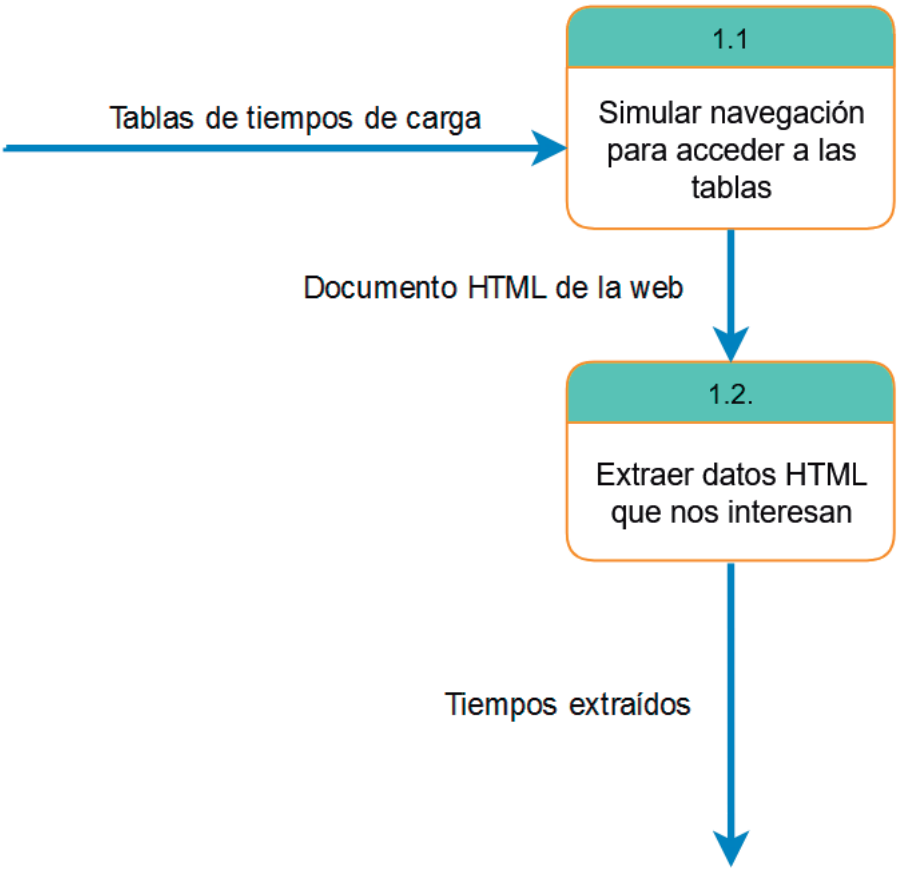

Figura 3.11: DFD de nivel 1 del proceso 1 del *Sistema de Rendimiento*.

En esta página y la siguiente se recogen los DFDs de nivel 1 que se han elaborado para el análisis del ***Sistema de Rendimiento***.

En la Figura 3.11 se representa el DFD de nivel 1 del primer proceso del sistema. El análisis que se ha hecho para la Figura 3.8 es válido, pues ambos diagramas son idénticos a excepción de los flujos de datos, como era de esperar.

Del mismo modo que para el *Sistema de Licencias*, se ha considerado que el proceso 2 del *Sistema de Rendimiento*, representado en la Figura 3.7, no necesita una mayor descomposición al tratarse de una función única de formateo de datos.

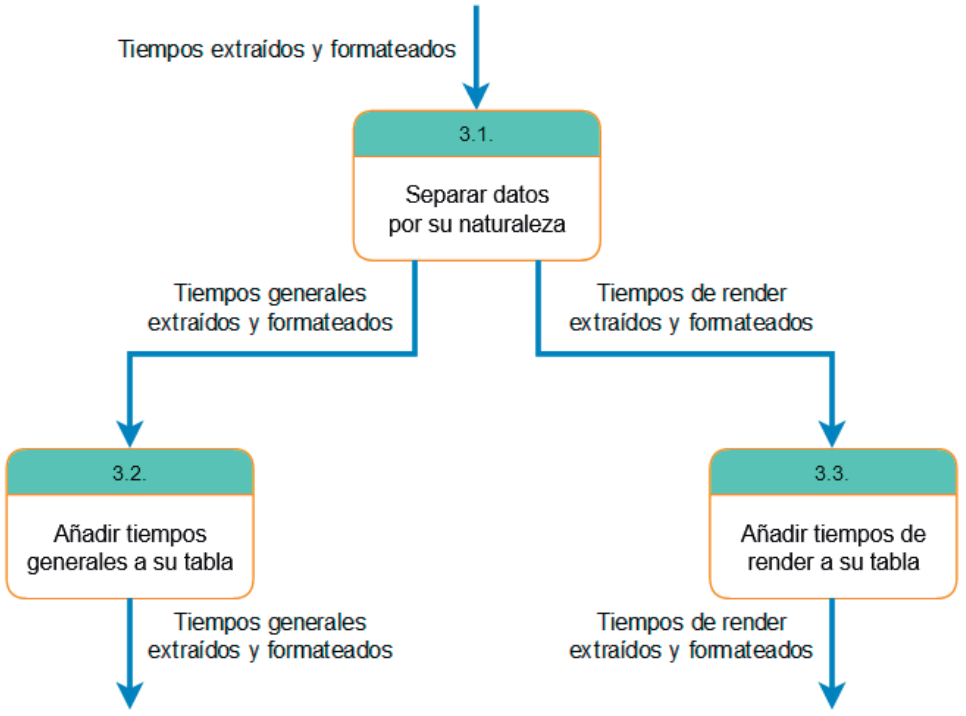

Figura 3.12: DFD de nivel 1 del proceso 3 del *Sistema de Rendimiento*.

El proceso 3 del sistema sí que ha sido objeto de una descomposición en su Diagrama de Flujo de Datos de nivel 1 representado en la Figura 3.12. Concretamente, se descompone en **3 procesos**:

1. **Separación** de datos por su naturaleza. A este objeto se representa el proceso 3.1. del sistema. Como **flujo de entrada** recibe los datos de los tiempos extraídos y formateados del proceso anterior y genera **dos flujos de salida de datos**. Al final de este proceso tendremos perfectamente diferenciados los *tiempos generales* de carga por un lado y, por el otro, el detalle del tiempo de *processing* que hemos comentado anteriormente y que se denomina *tiempos de render* en los diagramas.

2. **Añadir** *tiempos generales* a la Tabla del Histórico correspondiente (proceso 3.2). Notar que se usa el verbo ”añadir” y no el verbo ”actualizar” como se hacía en el *Sistema de Licencias*. Esto es porque en este sistema no hay un proceso de lectura del histórico y comparación previa, sino que simplemente se añaden al final de la tabla los datos extraídos de la web.

3. El proceso 3.3 es idéntico al descrito en el 3.2, pero es necesario separarlos porque son operaciones que se realizan sobre **flujos de datos** diferentes.

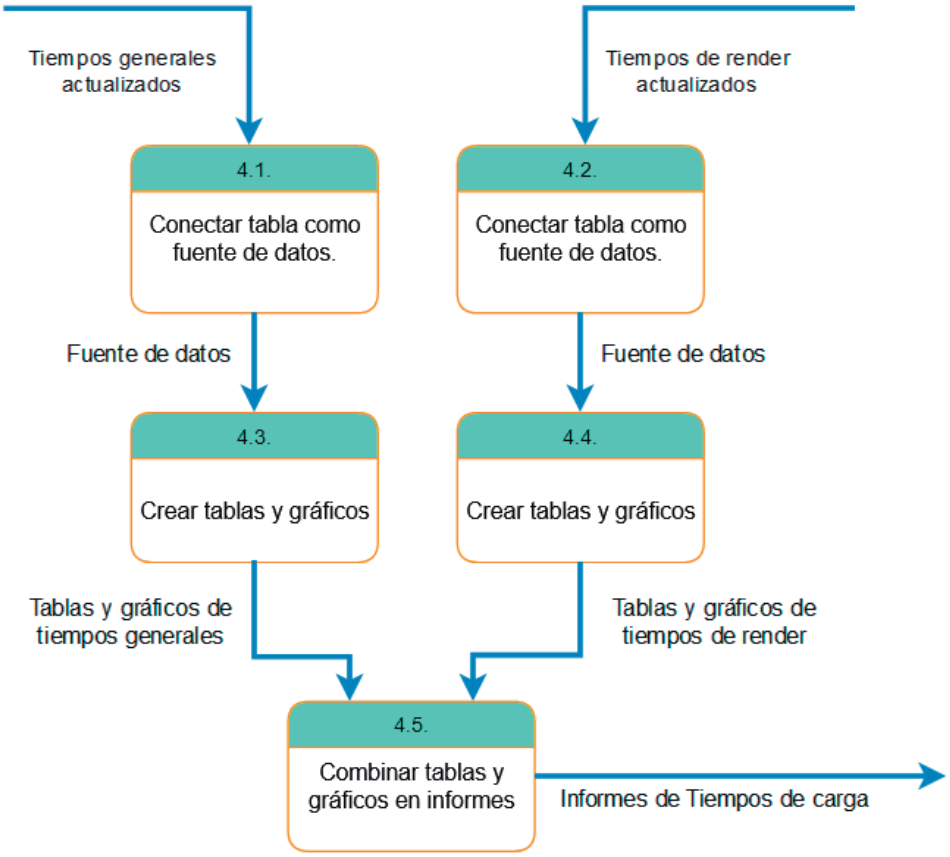

Figura 3.13: DFD de nivel 1 del proceso 4 del *Sistema de Rendimiento*.

En la Figura 3.13 representa la descomposición de nivel 1 del proceso número 4 del *Sistema de Rendimiento*. Vemos que el DFD está compuesto por 5 procesos y es necesario hacer unos comentarios al respecto:

1. Los **procesos 4.1** y **4.2** de nuevo representan la misma funcionalidad del sistema pero para flujos de datos distintos. Esta funcionalidad es la de **Conectar** sus respectivas Tablas Históricas como fuentes de datos de las herramientas de visualización.

2. Los **procesos 4.3** y **4.4** también difieren únicamente en los flujos de datos con los que operan. Abstraen las operaciones del **crear tablas y gráficos** del sistema de una forma muy similar a la descrita en el proceso 5.2 de la Figura 3.10.

3. El **proceso 4.5** representa la funcionalidad del sistema que permite **combinar** las tablas y gráficos procedentes de dos fuentes distintas en los mismos **Informes de Tiempos de carga**.

Con estas gráficas hemos alcanzado el **DFD primitivo** de nuestro proceso de análisis para el *Sistema de Rendimiento*. Esto significa el final del Análisis a través de los Diagramas de Flujos de Datos.

3.5. Conclusiones

Se han presentado las fases iniciales del Ciclo de Vida de Desarrollo del Sistema. A pesar de que puedan parecer fases poco productivas, pues durante ellas no encontramos avances en el desarrollo *software* como tal, es muy importante invertir tiempo en llevar a cabo la Planificación y el Análisis del proyecto de la manera más precisa posible. Muchos de los mayores bloqueos y errores dentro del SDLC se producen por no haber realizado el Análisis previo del sistema de una manera rigurosa. Por tanto, una buena planificación es vital para un desarrollo ágil y eficiente del sistema.

A modo de ejemplo, durante estas fases se ha hecho evidente la necesidad de separar nuestro sistema en dos soluciones distintas. Como consecuencia, durante el análisis se ha realizado un doble esfuerzo por discernir entre las similitudes y diferencias de ambas soluciones. Como ventaja, en las fases sucesivas esta individualización permitirá abordar la implementación de las soluciones de una manera mucho más adecuada.

Además, se ha comprobado la importancia que pueden llegar a tener los diagramas a la hora de atacar un problema y diseñar su solución. No obstante debemos utilizarlos de manera racional y comedida, ya que las figuras y gráficos se crean con el objetivo de ayudar tanto al desarrollador como a los interesados en entender el sistema. Si una representación gráfica no favorece ninguno de estos objetivos, por resultar complejos y/o poco intuitivos, su creación supone un obstáculo en lugar de un apoyo.

Finalizadas estas primeras fases, nos situamos en disposición de atacar las siguientes. En el próximo Capítulo se abordan las fases de Diseño e Implementación de los sistemas que nos atañen.

Capítulo 4

Diseño e Implementación

4.1. Introducción

El contenido de este capítulo se dedica a las fases finales del SDLC de los sistemas bajo estudio. En particular, las fases de Diseño e Implementación.

Antes de eso, en la Sección 4.2 se recopilan las **tecnologías involucradas** en el desarrollo de los sistemas y se introducen brevemente cada una de ellas.

Posterioremente, se describe el proceso de **Diseño** en la Sección 4.3. Se ofrece una visión tanto del *Sistema de Licencias* como del *Sistema de Rendimiento*. Además, se presenta el Diagrama de Flujo del programa que gobierna ambos sistemas y todas las acciones de diseño relativas a los conjuntos de datos y las gráficas de visualización.

Finalmente, en la Sección 4.4 se pretende explicar la etapa de **Desarrollo** de lo sistemas desde una perspectiva comprensible para el lector. Se hace de forma separada para los dos sistemas, aunque siempre destacando los puntos en común. De especial importancia son las secciones en las que se explica le creacción de los proyectos *Node.js* así como las Figuras que muestran los resultados de las visualizaciones.

4.2. Tecnologías utilizadas

4.2.1. Tealeaf

Tealeaf, conocida también en la actualidad como *Acoustic Experience Analytics*, es una herramienta para la analítica de la Experiencia de Usuario a través de captura de sesiones en la web. Es decir, como ellos mismos dicen en su página web[1], sirve para interpretar el comportamiento del cliente y cómo este impacta en el negocio a través del **análisis de sesiones**.

[1] https://www.acoustic.com/products/tealeaf

Esta herramienta constituye una solución completa de analítica ofreciendo versiones tanto en la nube como *on-premise*. En nuestro caso se ha utilizado su versión en la nube accesible a través de la web. No profundizaremos demasiado en su configuración porque en nuestros sistemas simplemente actúa como la fuente de la que extraer los datos de la web. No obstante, veremos que en todo sistema hay que tener en cuenta cómo se disponen estos datos y cómo se accede hasta ellos. Por ello, veremos que la parte inicial del desarrollo se centra en la preparación y configuración de tablas de datos de *Tealeaf*. Para la creación de estas tablas se han utilizado conocimiento previos de la herramienta así como la documentación oficial [Aco].

Además, es importante destacar que esta herramienta es el objetivo de uno de nuestros sistemas, en concreto del *Sistema de Licencias*. Es decir, la empresa como cliente de esta herramienta paga una precio a cambio de su uso. Ese uso está limitado a una cuota, a la que nos referimos como licencia. La utilización en exceso de la herramienta provoca una extralimitación de la cuota que puede acarrear altos costes económicos. El *Sistema de Licencias* precisamente nace con el propósito de monitorizar este consumo a través de los datos disponibles en *Tealeaf* para tener una mayor seguimiento y control. Veremos que la licencia contratada permite un máximo de 12 millones de interacciones mensuales en la herramienta.

Por lo tanto, *Tealeaf* tiene una **doble importancia** en el desarrollo:

- Por un lado, constituye el **origen del que extraer los datos** de ambos sistemas. Esos datos han de disponerse en tablas de forma adecuada para que el *bot* pueda acceder a ellos.
- Por otro lado, monitorizar el consumo mensual que se hace de esta herramienta es el **objetivo principal del *Sistema de Licencias***

4.2.2. Node.js y Puppeteer

Los *scripts* que componen el proyecto están basados en el lenguaje de programación *JavaScript*. Se trata de un lenguaje interpretado cuyo uso está muy extendido entre las páginas web, pero que es también utilizado en entornos fuera del navegador, como ocurre en nuestro caso. Concretamente, los núcleos de nuestros sistemas se han concebido como proyectos que utilizan la tecnolgía de *Node.js*. ***Node.js* es un entorno de ejecución multiplataforma de *JavaScript* para la capa del servidor.** Está basado en el motor V8 de *Google* (el núcleo del navegador) y es de código abierto [Nod].

Su característica de *open-source*, su escalabilidad y la ventaja que ofrece unificando el desarrollo de aplicaciones web en torno a un único lenguaje, han hecho que haya sido una de las tecnologías que más ha crecido en popularidad en los últimos años. A esto, hemos de sumar su gestor de paquetes (el *Node Package Manager*) que simplifica mucho la instalación y actualización de librerías. De entre una infinidad de librerías, destaca ***Pupeeteer***. Como ya se ha introducido previamente en los Capítulos 1 y 2, esta librería de *Node.js* proporciona una API de alto nivel para controlar *Chrome* o *Chromium* a través del *Protocolo DevTools*. El *bot* que da "vida" a nuestros sistemas está basado en esta librería y por ello ha sido preciso realizar un estudio de su documentación [Pup] previo

al desarrollo. Concebida para todas las acciones que tengan que ver con la simulación de sesiones del navegador, utilizaremos esta librería para implementar todos los requisitos funcionales que se refieran a la extracción de datos.

4.2.3. Google APIs y GoogleSheets

Siguiendo el orden lógico, tras introducir las tecnologías utilizadas en el origen de los datos (*Tealeaf*) y en el núcleo del sistema (*Puppeteer*), repasamos a continuación las tecnologías empleadas en el destino de los flujos de datos.

De la definición de Requisitos de la Sección 3.3, así como de los Diagramas de Flujo de Datos de la Sección 3.4, emerge la necesidad de almacenar los datos extraídos en unas tablas. La tecnología utilizada para ello es ***Google SpreadSheets*, una herramienta de hojas de cálculo basada en web**. Sus funcionalidades son muy similares al popular *software* de *Microsoft Excel*. Ello sumado a que únicamente la utilizaremos como almacén de datos, provoca que no sea necesaria una explicación de las funcionalidades más complejas que puede ofrecer esta herramienta. Se mencionan a continuación las características que han hecho que nos decantemos por esta herramienta frente a sus alternativas:

- Está basada en web. Lo que nos permite una mayor accesibilidad de los datos a través de la red.
- *Google* ofrece esta herramienta de manera gratuita.
- Existen conectores tanto con las tecnologías del *core* (*Node.js*) como con las tecnologías de visualización (*Data Studio*). Esto facilita la integración de todos los componentes del sistema.

Precisamente, esta última característica es la que hace que esta herramienta sea tan potente. La conexión de los *sheets* de *Google* con *Data Studio* se hace en cuestión de segundos. Sin embargo, la publicación de los datos extraídos mediante *Puppeteer* en estas hojas de cálculo es una tarea más compleja que precisa de una tecnología intermedia: las ***Google APIs***. Se trata de Interfaces desarrolladas por *Google* para integrar sus servicios con otros servicios. En el caso de nuestros sistemas basados en *Node.js*, nos interesa la API de *Google Sheets* para realizar peticiones simples desde *Node.js* a nuestras hojas de cálculo. Gracias a eso nuestro *bot* podrá leer datos históricos almacenados en las hojas, así como actualizar los datos recogidos en estas tablas. Más adelante se detalla cómo se han implementado estas funcionalidades gracias a la documentación [Gsh].

4.2.4. Data Studio

Sin salir de las herramientas ofrecidas por *Google*, se ha escogido ***Data Studio* como herramienta de visualización de los datos de los sistemas**. Es una herramienta que permite crear informes y *dashboards* a partir de distintas fuentes de datos. Constituirá la parte final del desarrollo y será la única parte visible de los sistemas para los usuarios finales. Son muchas las bondades que tiene esta herramienta, las principales por las que se ha elegido son:

- Posee conexión directa con *Spreadsheets de Google*.
- Es gratuita.
- Permite generar informes a partir de distintas fuentes. Lo que será útil para combinar las tablas del *Sistema de Rendimiento*.
- Fácil de compartir información y colaborar.
- Gran interactividad con los gráficos.

Para resumir, es una herramienta de bajo coste que cumple con todos los Requisitos Funcionales y No Funcionales referidos a la visualización.

4.3. Diseño

El **Diseño** constituye la siguiente fase del SDLC en la que el desarrollador y el usuario interactúan para **concretar cómo va a operar el sistema** [JSV17]. Gracias a la información recopilada y generada en las fases previas de Análisis y Planificación, en la fase del **Diseño se plantea la solución final que se quiere desarrollar**, utilizando las tecnologías seleccionadas y teniendo en cuenta esos **requisitos funcionales, no funcionales y los flujos de datos** de las fases previas. Además, al Diseño le sucede la fase de Desarrollo del sistema propiamente dicho, por ello se deben especificar claramente todas las necesidades previas a la configuración de cada tecnología.

Como se viene haciendo en este TFG, también separaremos la fase de Diseño para las 2 soluciones: el *Sistema de Licencias* y el *Sistema de Rendimiento*.

4.3.1. Sistema de Licencias

A continuación, se detallan todas las actividades llevadas a cabo durante la fase de **Diseño** del *Sistema de Licencias*.

Visión del Sistema de Licencias

Para diseñar un sistema, es fundamental definir los elementos que lo conforman, las tecnologías que utilizan y cómo interactúan e intercambian datos. Por ello, es muy importante tener en cuenta los Diagramas de Flujo de Datos generados durante la fase del Análisis en la Sección 3.4. Partiendo del DFD de nivel 0 de la Figura 3.6 se diseña el *Sistema de Licencias* que vemos en la Figura 4.1. Se puede comprobar que está compuesto de **4 elementos** principales (en cajas redondeadas) que interactúan entre ellos para el correcto funcionamiento. Esta relación entre los elementos se representa mediante flechas. Observamos dos tipos de flechas. Las **flechas continuas** representan **acciones** que un elemento realiza sobre otro. Las **flechas discontinuas** representan el **intercambio de datos** entre elementos.

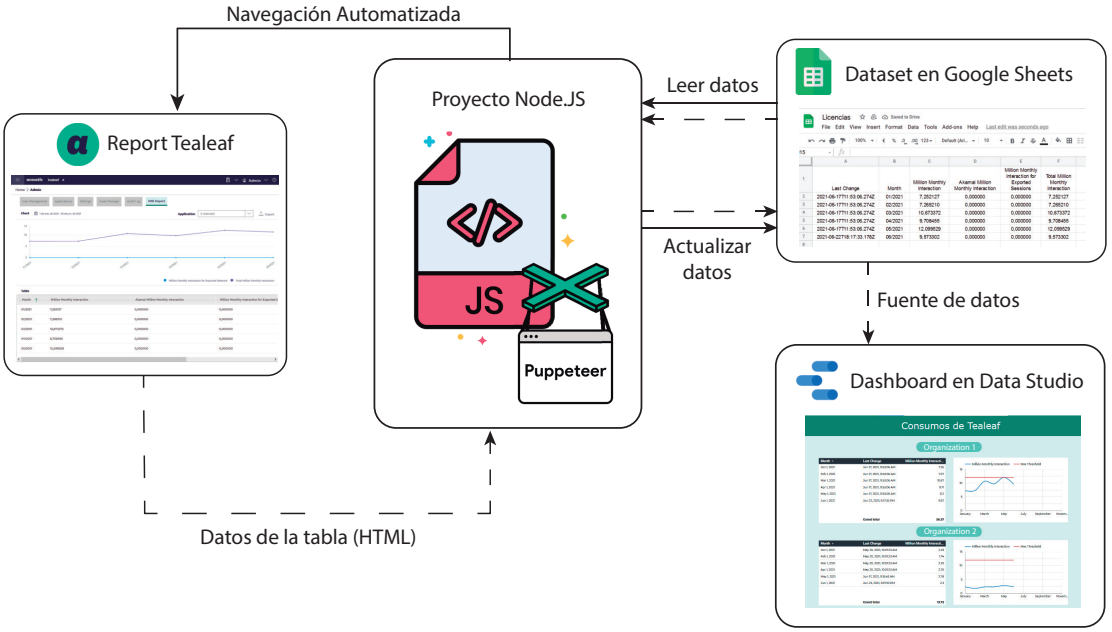

Figura 4.1: Visión del *Sistema de Licencias*. Elaboración propia.

Cada uno de los elementos constituye una parte del sistema y utiliza diferentes tecnologías. El elemento principal del sistema es el *Proyecto Node.js* que engloba todos los ficheros *JavaScript* necesarios para la creación del *bot* de *Puppeteer*. En la sección de Desarrollo veremos cómo está configurado este proyecto. Lo que se extrae de la Figura 4.1 es que este elemento, a través de una navegación automatizada, interactúa con el elemento *Report de Tealeaf* y obtiene los datos de la tabla HTML que está en la web. Por otro lado, el elemento se interrelaciona con el *Dataset de Google Sheets* leyendo los datos históricos almacenados y actualizando la tabla de datos con los nuevos datos. Por último, el elemento *Dashboard en Data Studio* utiliza las tablas del *Dataset* como fuente de datos.

En los siguientes apartados se profundiza en el diseño de cada uno de los elementos que conforman el sistema.

Diagrama de Flujo del programa

El proyecto de *Node.js* finalmente se materializa en un programa que deberá llevar a cabo diferentes acciones de la lógica del sistema. Estas acciones se producirán de manera secuencial y antes de programarlas es preciso diseñar el flujo que deberá seguir el programa. Esto es lo que se representa en la figura 4.2

La Figura 4.2 representa la secuencia del programa que constituye el núcleo del *Sistema de Licencias*. Se ha elaborado teniendo en cuenta que será implementado en *Node.js* a través de programación asíncrona y que debe de ir en concordancia con los Diagramas de Flujo de Datos de nivel 0 y de nivel 1 elaborados en la Sección 3.4. En azul se representan las **acciones** o **estados de actividad** y en amarillo los **nodos de decisión**.

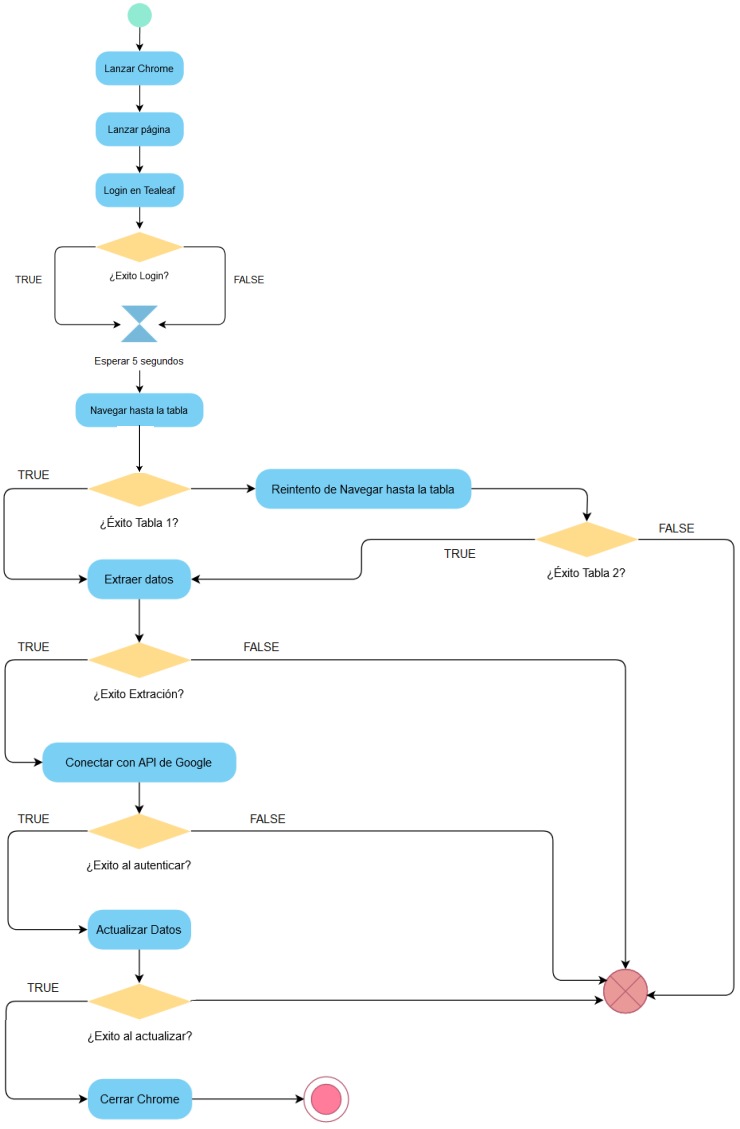

Figura 4.2: Diagrama de flujo del programa. Elaboración propia.

Desde arriba hacia abajo, encontramos un círculo verde que representa el **estado inicial**. A continuación se suceden 3 acciones: *Lanzar Crhrome*, *Lanzar la página* en el navegador y realizar el *Login en Tealeaf*. A través del primer nodo de decisión, se evalúa si se ha producido el *login*. Las dos ramas que surgen de este nodo convergen en el mismo **evento de tiempo** y el programa espera a que cargue la página de inicio durante 5 segundos. Esta convergencia se da porque la rama TRUE representa la situación en la que se han introducido usuario y contraseña completándose el proceso de *login* y, por otro lado, la rama FALSE representa el caso en el que no se necesita hacer el *login* porque las credenciales se han guardado de manera automática en el navegador y no ha sido necesario introducir el usuario y contraseña para acceder hasta la pantalla de inicio. Es decir, en ambos casos conseguimos acceder hasta la pantalla de inicio y no se contempla el escenario en el que se produzcan errores en el *login*.

Tras la espera de 5 segundos, el programa ha de simular la *navegación hasta la tabla*. Si esta acción falla, se realizará un segundo intento de acceso a la tabla y, si volvemos a errar en el intento, el programa finalizará sin éxito. Este estado de **final con error** se representa con el círculo rojo tachado. Por otro lado, si conseguimos acceder hasta la tabla ya sea en el primer o segundo intento, la siguiente acción es la de *extraer los datos*. Una vez extraídos los datos del HTML, se probará a *Conectar con la API de Google Sheets*. Si se consigue con éxito, el siguiente paso es *Actualizar los Datos* del *spreadsheet*. El programa ha de evaluar si cualquiera de estas 3 últimas acciones no se puede completar con éxito, finalizando la secuencia en el estado de final con error. En caso contrario, se *cierra Chrome* llegando así al **estado final sin errores** del programa, representado como un círculo rojo y una circunferencia concéntrica.

Esto sería el diseño de la lógica que ha de seguir el programa. En la Sección 4.4, veremos cómo se ha llevado a cabo el desarrollo de todas estas acciones del programa.

Diseño de tablas de datos

El diseño de la base de datos que utiliza un sistema suele ser una de las actividades que más esfuerzo requieren. Sin embargo, en las soluciones que se desarrollan en este TFG, esta tarea se ha realizado de una manera muy sencilla por la poca cantidad de datos con los que trabajamos y las ventajas que tiene el trabajar con hojas de cálculo *on cloud*. Para el *Sistema de Licencias*, la decisión de los datos que vamos a guardar viene dada directamente por los datos que están disponibles en la web, porque son los que vamos a extraer. Estos campos son los que vemos en la captura de la Figura 4.3.

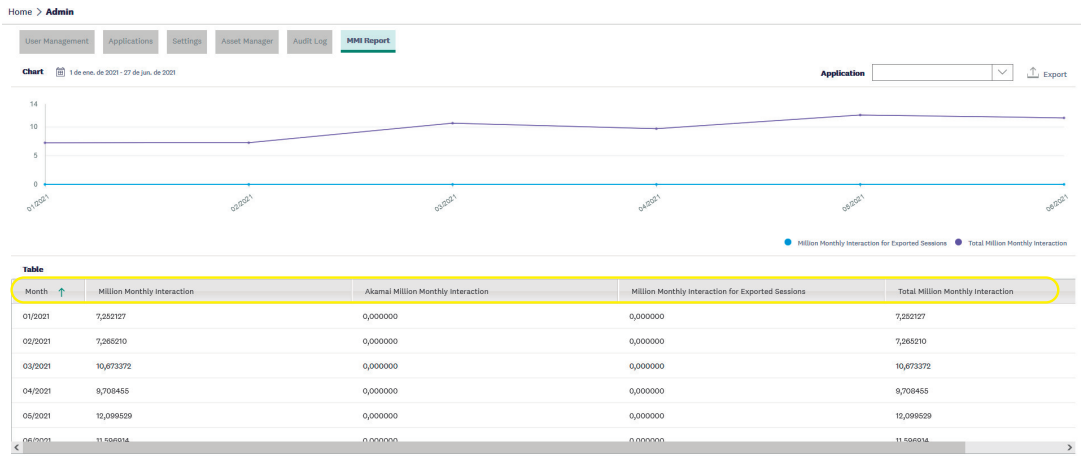

Figura 4.3: Información disponible en *Tealeaf* y que el sistema debe capturar.

Todos esos campos son los que necesitaremos. A mayores, se incluye el campo *Last Change* para llevar un registro de la última fecha en la que se ha modificado la tabla. En la la Tabla 4.1 se recogen los campos que almacenaremos en el *spreadsheet*.

Durante el diseño, para cada campo hemos de decidir el *Tipo* de dato que guardamos y una pequeña *Descripción*. Los campos más importantes para el objetivo de realizar el seguimiento del consumo de licencias son los señalados en negrita.

Tabla 4.1: Campos de Datos del *Sistema de Licencias*.

Campo	Tipo	Descripción
Last Change	Fecha y Hora	Fecha y Hora de la última actualización de datos realizada sobre una fila concreta de la tabla
Month	Fecha (mes y año)	Mes y Año al que se refiere el consumo de Licencias
Million Monthly Interaction	Número	Interacciones Mensuales registradas en *Teleaf* y medidas en millones.
Akamai Million Monthly Interaction	Número	Interacciones Mensuales registradas en por el uso del conector de *Akamai* y medidas en millones.
Million Monthly Interaction for Exported Sessions	Número	Interacciones Mensuales registradas en por el uso de la funcionalidad de exportar sesiones y medidas en millones.
Total Million Monthly Interaction	Número	Consumo de Licencias medido en millones de Interacciones Mensuales. Es la suma de los 3 campos anteriores.

Diseño de gráficos de visualización

El resultado final del sistema es el Informe de Licencias creados en *Data Studio*. Como será consumidopor el Equipo de Negocio, el diseño de estos ha de ser fruto de un entendimiento entre este equipo y el desarrollador. Por parte del Equipo de Negocio, lo que demandaba eran dos gráficos sencillos por cada organización con licencia en *Tealeaf*. Uno primero debía ser una **tabla con 3 columnas**: el mes, la última actualización de ese mes en la tabla y el dato de millones de interacciones mensuales. Un segundo gráfico consistirá en representar los datos de **millones de interacciones** como una sencilla **función respecto del tiempo** (medido en meses). Además se ha de representar también el **umbral contratado** de consumo que no se ha de sobrepasar. Para nuestro caso son 12 millones de interacciones mensuales.

Esto ha de repetirse para las 5 organizaciones para las que la empresa tiene licencia de este *software* y mostrarse todo ello en un único informe.

4.3.2. Sistema de Rendimiento

En los siguientes apartados se recogen las distintas actividades realizadas durante la fase de **Diseño** del *Sistema de Rendimiento*. El contenido será similar al del *Sistema de Licencias*, aunque el diseño será algo más complejo. Por lo tanto, nos detendremos sobre todo en los puntos que no son comunes respecto al otro sistema.

Visión del sistema

Como vemos en la Figura 4.4, el *Sistema de Rendimiento* es muy similar al de *Licencias*.

Está compuesto por los mismos 4 tipos de elementos. Entre las diferencias respecto al de la Figura 4.1 podemos destacar que:

- Ahora encontramos 2 elementos de tipo *Report Tealeaf*. Esto es porque la información se encuentra en dos tablas distintas de la web y no en una como ocurría en la otra solución.
- De cada uno de esos dos elementos obtenemos un tipo de datos distinto. De uno de los *reports* obtenemos los datos de la **tabla de tiempos general** y, del otro, los datos de la **tabla de tiempos *render***. Por este motivo, hay dos flujos distintos de datos hacia el elemento principal del sistema representados con las flechas discontinuas.
- Ahora el *proyecto Node.js* únicamente realiza la acción de *Actualizar datos* sobre el *dataset*. Es decir, ya no es necesaria la lectura de datos. Además se actualizan los dos tipos de datos en dos *datasets* distintos.
- Esos dos *datasets distintos* serán las dos fuentes de datos que se utilizarán para crear los *dashboards en Data Studio*. Además, a diferencia del sistema anterior, en este caso se producirán tres informes distintos en lugar de uno solo.

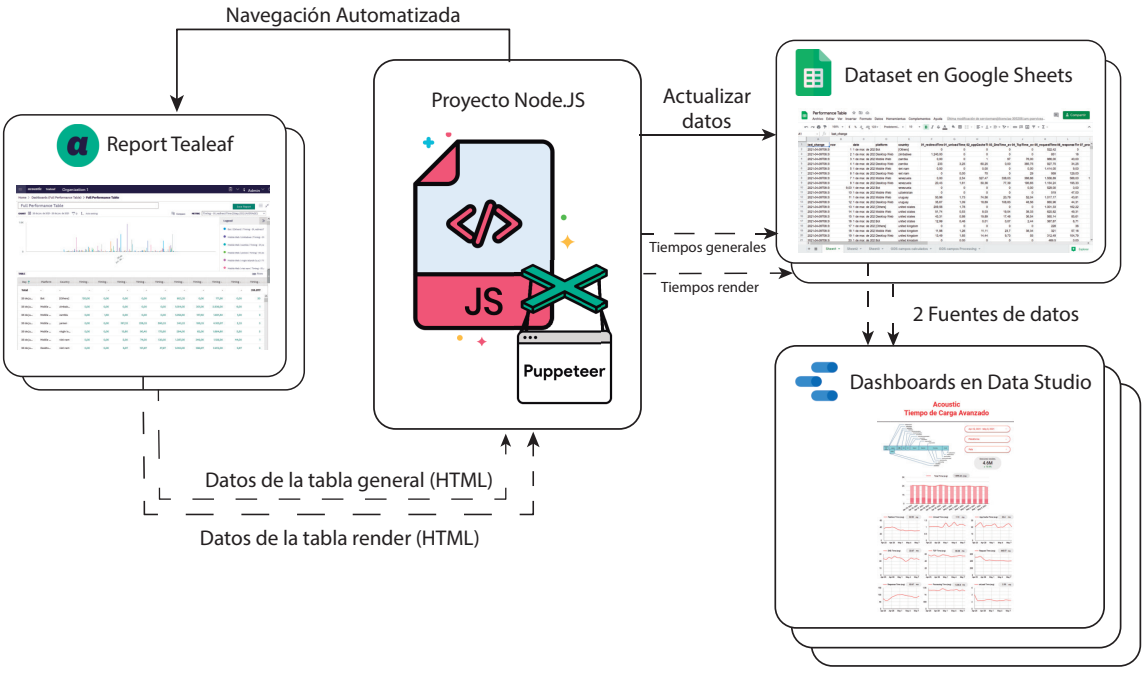

Figura 4.4: Visión del *Sistema de Rendimiento*. Elaboración propia.

Diseño de las tablas de Tealeaf

Esta es una de las diferencias más grandes respecto al otro sistema implementado. En este caso, los datos que extraemos de la web no provienen de un *Report* automático que *Tealeaf* ofrece en su página, sino que nosotros tenemos que crear los *Reports* con los datos necesarios para realizar los informes. Por lo tanto, habrá que configurar unas tablas dentro del *workspace* de *Tealeaf* que nos muestren los datos que nos interesan. Estos datos se capturan mediante configuración de eventos de la herramienta, pero este trabajo ya estaba realizado y no entra dentro del alcance del TFG. Las tablas a realizar se diseñan a partir de las necesidades que tienen los informes que quiere generar y consumir el equipo de *Web Performance Optimization*. Estos requisitos, entre otros, se recogieron en las Tablas 3.1 y 3.2. Como resultado del diseño de estas tablas, obtenemos las siguientes requisitos:

- Se han de configurar dos tablas distintas en la web de *Tealeaf* en función de los datos que se quieran mostrar.
  - Tabla de *tiempos* de rendimiento *generales*. Son los valores para los diferentes tiempos descritos en la Figura 2.4 medidos a través de *Tealeaf*.
  - Tabla de *tiempos de render*. Como ya se introdujo, se trata de un desglose del tiempo de *render*. Esta descomposición en otros 5 tiempos se representa en la Figura 4.5
- Además, para poder cumplir con el *FR15*, los tiempos de las tablas deben mostrarse segmentados por *fecha*, *país* y *plataforma*.

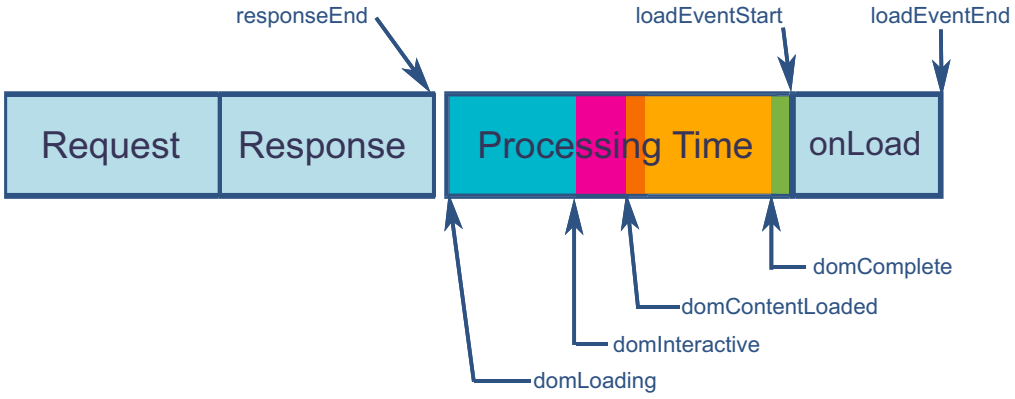

Figura 4.5: Desglose de tiempos del *Render Time*. Elaboración propia.

Diagrama de Flujo del programa

A pesar de de las muchas diferencias que ya se han ido haciendo presentes entre el *Sistema de Licencias* y el *Sistema de Rendimiento*, son muchos más los puntos que los unen. Prueba de ello es que el Diagrama de Flujo del programa que gobierna cada sistema es válido para ambos y es el que vimos en la Figura 4.2. Esto es porque la secuencia de actividades que sigue el *script* es la misma para el nivel de abstracción representado

en el diagrama. Ya hemos visto que hay diferencias entre los procesos que componen los sistemas, pero estos no están visibles para el alcance representado. En la implementación se verá que la acción *Extraer datos* de la Figura 4.2 diferirá mucho entre las dos soluciones. Existirán muchas diferencias también en la implementación de la acción *Actualizar datos*, por ejemplo.

No obstante, la validez del mismo diagrama para los dos sistemas manifiesta y refuerza la idea de desarrollarlos ambos entre los contenidos del mismo TFG.

Diseño de tablas de datos

Para este sistema vamos a trabajar con una mayor cantidad de datos que para el anterior. Estos datos vienen definidos por las necesidades de visualización. Vamos a incluir un nuevo concepto que son las **métricas calculadas**, que son campos que se calculan al aplicar operaciones de **agregación** sobre los **campos originales**.

Antes de diseñar las tablas con los campos necesarios del sistema, vamos a aclarar todo el "viaje" que experimenta un dato a través del *Sistema de Rendimiento*.

1. El **dato original** se captura en *Tealeaf* y se muestra en la web a través de una tabla.
2. El sistema simula una navegación en *Chrome*, accede hasta la tabla y recoge el dato disponible en el **HTML** de la página.
3. El sistema actualiza el *spreadsheet* de *Google*. El dato ya se encuentra en una **celda** de la hoja de cálculo que le corresponda.
4. *Data Studio* utiliza la hoja de cálculo como fuente de datos. Aquí tenemos dos posibilidades:
   - Si nos interesa representar el **dato original** en alguna gráfica, se hará directamente.
   - Si nos interesa representar una **métrica calculada** en *Data Studio*, deberemos añadir una nueva métrica agregada añadiendo una fórmula que combine otros campos (ya sean originales del *dataset* o calculados).
5. Finalmente, habrá que incluir el campo que queremos representar en el informe a la hora de crear las gráficas.

Como ya hemos visto, tendremos 2 hojas de cálculo en función de la naturaleza de los tiempos que extraemos. Es muy importante tener claro que, en cada una de las hojas de cálculo, únicamente se almacenaran los **campos originales**. Sin embargo, en las tablas que se presentan a continuación se añaden también las **métricas calculadas** porque, desde el punto de vista de *Data Studio*, los conjuntos de datos para crear las gráficas incluyen también estas métricas.

Es decir, cuando hablamos de *dataset* podemos referirnos a dos conceptos diferentes:

- *Dataset desde el punto de vista de Google Sheets.* Son los datos que extrae y almacena el sistema. Está compuesto únicamente por **campos originales**.
- *Dataset desde el punto de vista de Data Studio.* Incluye todos los valores que son susceptibles de utilizarse en las gráficas de visualización. Esto son tanto los **campos originales** como las **métricas calculadas**.

Los campos asociados a los *tiempos generales* constituyen el conjunto de datos denominado **general_performance**. El desglose de los *tiempos de render* y sus campos asociados se recogen en el conjunto de datos denominado **processing_detallado**. A continuación representamos los tablas para cada *dataset* desde el punto de vista de *Data Studio*.

Tabla 4.2: Dataset: general_performance

Campos Originales		
Campo	Tipo	Descripción
01_redirectTime_avg	Número	Tiempo promedio transcurrido desde que comienza una redirección hasta que se recibe el último *byte* de la respuesta de la última redirección.
01_unloadTime_avg	Número	Tiempo promedio transcurrido desde que el navegador comienza la descarga (*unload*) del documento previo hasta que la finaliza.
02_appCacheTime_avg	Número	Tiempo promedio que emplea en recuperar recursos de cualquier caché de aplicación relevante.
03_DnsTime_avg	Número	Tiempo prmedio empleado para la resolución de nombres de dominio. Coincidirá con el 02_appCacheTime si la resolución de nombres se recupera a través de la caché.
04_TcpTime_avg	Número	Tiempo promedio que se tarda en establecer la conexión con el servidor para obtener el documento.
05_requestTime_avg	Número	Tiempo promedio que transcurre desde que el navegador comienza su petición HTTP del documento hasta que recibe el primer byte de la respuesta del servidor.
06_responseTime_avg	Número	Tiempo promedio que transcurre en el navegador entre la recepción del primer byte y el último byte de la respuesta del servidor.

Continuación de la Tabla 4.2		
Campo	Tipo	Descripción
07_processingTime_avg	Número	Tiempo promedio desde que el navegador comienza el parseo de documento hasta que termina de cargar todos los recursos asociados al mismo.
08_onLoadTime_avg	Número	Tiempo promedio de duración del evento onLOAD. Durante este tiempo se ejecutan los scripts asociados a este evento.
country	País	País asociado a los tiempos.
date	Texto	Fecha asociada a los tiempos.
day_hour	Fecha	Día y Hora asociados a los tiempos.
last_change	Fecha y Hora	Fecha y Hora de la última actualización de datos realizada sobre una fila concreta de la tabla
ocurrences	Número	Tamaño de la muestra sobre el que se realiza el promedio de cada tiempo
platform	Texto	Plataforma asociada a los tiempos.
row	Número	Fila de la tabla del *Report* de Tealeaf asociada a los tiempos
Métricas Calculadas		
01_redirectTime_total_avg	Número	Métrica final que se representa en las gráficas. Consiste en calcular el tiempo según el peso que tenga a la contribución total de la media. Se calcula como: SUM(01_redirectTime_avg * ocurrences) / SUM(ocurrences)
01_unloadTime_total_avg	Número	SUM(01_unloadTime_avg * ocurrences) / SUM(ocurrences)
02_appCacheTime_total_avg	Número	SUM(02_appCacheTime_avg * ocurrences) / SUM(ocurrences)
03_DnsTime_total_avg	Número	SUM(03_DnsTime_avg * ocurrences) / SUM(ocurrences)
04_TcpTime_total_avg	Número	SUM(04_TcpTime_avg * ocurrences) / SUM(ocurrences)
05_requestTime_total_avg	Número	SUM(05_requestTime_avg * ocurrences) / SUM(ocurrences)
06_responseTime_total_avg	Número	SUM(06_responseTime_avg * ocurrences) / SUM(ocurrences)

Continuación de la Tabla 4.2		
Campo	Tipo	Descripción
07_processingTime_total_avg	Número	SUM(07_processingTime_avg * ocurrences) / SUM(ocurrences)
08_onLoadTime_total_avg	Número	SUM(08_onLoadTime_avg * ocurrences) / SUM(ocurrences)
totalTime_avg	Número	(01_redirectTime_total_avg + 02_appCacheTime_total_avg + 03_DnsTime_total_avg + 04_TcpTime_total_avg + 05_requestTime_total_avg + 06_responseTime_total_avg + 07_processingTime_total_avg + 08_onLoadTime_total_avg)

Tabla 4.3: Dataset: processing_detallado

Campos Originales		
Campo	Tipo	Descripción
01_domLoadingToDomInteractive_avg	Número	Tiempo promedio desde que el navegador comienza el parseo de documento (domLoading) hasta que termina de parsear y todos los recursos bloqueantes.
02_domInteractiveToDomContentLoadedEventStart_avg	Número	Tiempo promedio desde que la página es interactiva (domInteractive) hasta que se completa la ejecución de los scripts que fueron cargados con defer (domContentLoadedEventStart).
03_domContentLoadeEventStartToDomContentLoadedEventEnd_avg	Número	Tiempo promedio de ejecución de los scripts asociados al evento domContentLoaded.
04_domContentLoadedEventEndToDomComplete_avg	Número	Tiempo prmedio que tardan en cargarse el resto de recursos no mencionados de la página (imágenes, ficheros, css no bloqueante, etc.).
05_domCompleteToLoadEventStart_avg	Número	Tiempo promedio que transcurre entre domComplete y onLoadEventStart que marca el inicio del evento onLOAD.

Continuación de la Tabla 4.3		
Campo	Tipo	Descripción
07_processingTime_avg	Número	Tiempo promedio desde que el navegador comienza el parseo de documento hasta que termina de cargar todos los recursos asociados al mismo.
country	País	País asociado a los tiempos.
date	Texto	Fecha asociada a los tiempos.
day_hour	Fecha	Día y Hora asociados a los tiempos.
last_change	Fecha y Hora	Fecha y Hora de la última actualización de datos realizada sobre una fila concreta de la tabla
ocurrences	Número	Tamaño de la muestra sobre el que se realiza el promedio de cada tiempo
platform	Texto	Plataforma asociada a los tiempos.
row	Número	Fila de la tabla del *Report* de *Acoustic* asociada a los tiempos
Métricas Calculadas		
01_domLoadingToDomInteractive_total_avg	Número	Métrica final que se representa en las gráficas. Consiste en calcular el tiempo según el peso que tengan a la contribución total de la media. Se calcula como: SUM(01_domLoadingToDomInteractive_avg * ocurrences) / SUM(ocurrences)
02_domInteractiveToDomContentLoadedEventStart_total_avg	Número	SUM(02_domInteractiveToDomContentLoadedEventStart_avg * ocurrences) / SUM(ocurrences)
03_domContentLoadeEventStartToDomContentLoadedEventEnd_total_avg	Número	SUM(03_domContentLoadeEventStartToDomContentLoadedEventEnd_avg * ocurrences) / SUM(ocurrences)
04_domContentLoadedEventEndToDomComplete_total_avg	Número	SUM(04_domContentLoadedEventEndToDomComplete_avg * ocurrences) / SUM(ocurrences)
05_domCompleteToLoadEventStart_total_avg	Número	SUM(05_domCompleteToLoadEventStart_avg * ocurrences) / SUM(ocurrences)
07_processingTime_total_avg	Número	SUM(07_processingTime_avg * ocurrences) / SUM(ocurrences)

Diseño de gráficos de visualización

Desde el departamento de *Web Performance Optimization* se describieron los puntos que debían seguir los informes de visualización. Tras un intercambio de correos la propuesta inicial era la siguiente:

- Poder sacar informes (exportar en PDF desde *Google Data Studio*) y enviarlos cada 2 semanas.
- Poder dar acceso a otros departamentos (ya sea de Web, Medición o Informática) para su consumo propio.
- (A medio plazo) Poder relacionarlo con métrica *Core Web Vitals* de *Google*.

El objetivo impuesto era *"Por vuestro lado habíamos pensado en mostrar el desglose de los tiempos de carga (y poder segmentarlo por dispositivo/país). Y en una página control inicial que ofrezca las peores/mejores combinaciones"*.

En base a esto, el equipo de WPO propuso un primer diseño de los informes basándose en las gráficas disponibles en *Acoustic*. A continuación, en las Figuras 4.6 y 4.7 se muestran estas primeras propuestas.

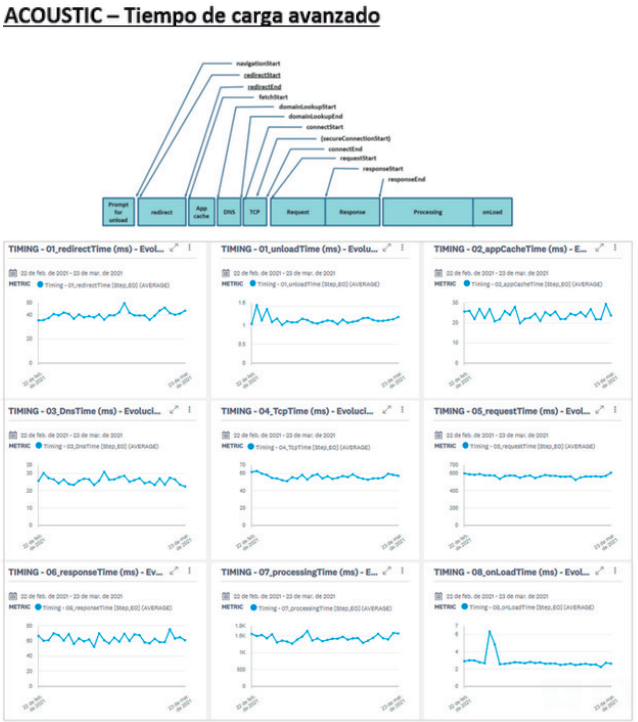

Figura 4.6: Diseño del informe de tiempos generales propuesto por el equipo de WPO.

En cuanto al informe inicial que se menciona, el proceso de diseño era más abierto y recaía sobre mi lado. Se decidió que constaría de 3 partes:

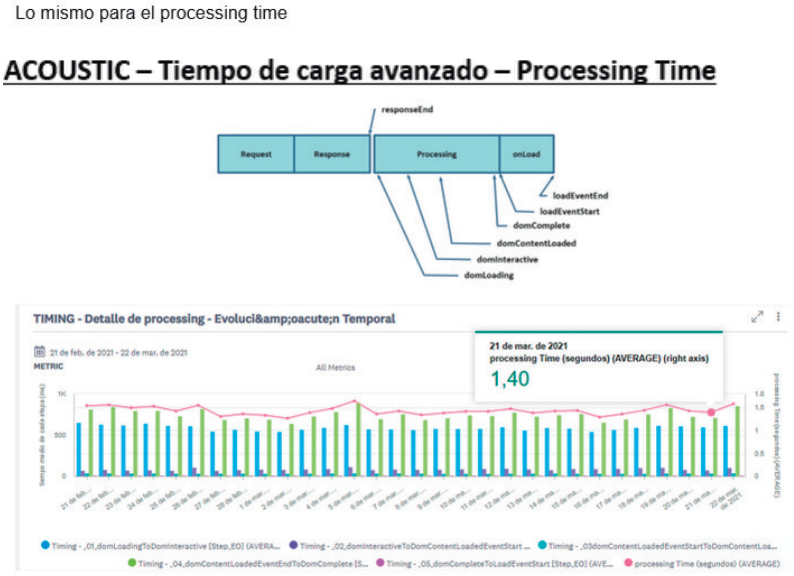

Figura 4.7: Diseño del informe de tiempos de *render* propuesto por el equipo de WPO.

- En la primera parte, unas gráficas que mostrasen los datos de las **plataformas** con mejores tiempos.
- En la segunda parte, unas gráficas que mostrasen los datos de los **países** con mejores tiempos.
- En una tercera parte, se crearía una tabla de doble entrada (o matriz) que mostrase las mejores y peores combinaciones de **país/plataforma**

4.4. Implementación

La **Implementación** es la fase más costosa y la que más tiempo consume de todo el SDLC. En esta fase, las especificaciones físicas del diseño deben convertirse en código, que debe ser testado hasta que la mayoría de los errores hayan sido detectados y corregidos. En los dos siguientes apartados se pretende documentar esta etapa del desarrollo *software* para las dos soluciones que nos conciernen. Se trata de un contenido con mayor carácter técnico que el visto hasta ahora. Sin embargo, el objetivo es dar una visión global de las principales configuraciones que se han tenido que llevar a cabo para la creación de los sistemas.

4.4.1. Sistema de Licencias

Durante esta etapa del desarrollo, se ha hecho un uso constante de todos los diagramas, tablas y requisitos producidos durante las fases previas. Así pues, para la implementación del *Sistema de Licencias* será muy importante tener siempre en mente la visión del sistema

de la Figura 4.1 y del diagrama del flujo del programa de la Figura 4.2. El contenido se ha dividido en diferentes apartados que siguen el orden lógico de la secuencia de las acciones del sistema.

Conexión con la API de Google Sheets

Como paso previo al desarrollo del *script*, será necesario habilitar el cliente de la API de *Google Sheets* a través de nuestra cuenta de *Google*. Este paso tan sólo se realizó una vez y es válido para poder usar esta API en nuestras dos soluciones. Los pasos son los siguientes:

1. Vamos a *https://console.developers.google.com* y creamos un proyecto con nuestra cuenta de *Google*.
2. Clicamos en el proyecto y habilitamos la API de *Google Sheets*.
3. Creamos las credenciales asociadas al proyecto. Vemos un ejemplo en la Figura 4.8.

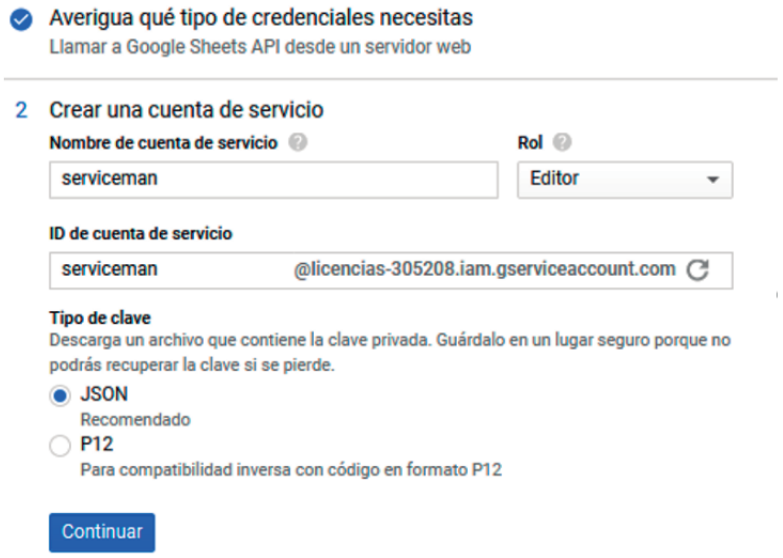

Figura 4.8: Credenciales necesarias para habilitar la API de *Google Sheets*.

4. Descargamos el fichero *XXX.json* que se genera. Lo movemos a nuestra carpeta de trabajo y lo renombramos como **keys.json**.
5. Con nuestra cuenta de *Google* debemos crear un *spreadsheet* y una Hoja que será nuestro conjunto de datos. Importante:
   - De la URL obtenemos el spreadsheetId que es un identificador único.
   - Debemos tener en cuenta el nombre de la hoja que vamos a editar. Por ejemplo, *Hoja 1*.
   - Hay que dar permisos de edición del *sheet* al *client mail* creado en el paso anterior. En nuestro caso: *servicemanlicencias-305208.iam.gserviceacount.com*

Creación del proyecto de Node.js

El primer paso para poder desarrollar un proyecto de *Puppeteer*, como es lógico, es tener instalado el programa *Node.js* e instalar la librería de *Puppeteer*. Hay que tener en cuenta que la programación es asíncrona, por lo que se ha sido muy cuidadoso a la hora de gestionar los tiempos de espera y se ha puesto mucho énfasis en la robustez del código. El proyecto consta de varios ficheros:

- **licencias.js**: es el fichero principal que implementa toda la lógica del programa.
- **keys.json**: ficheros con las credenciales necesarias para utilizar el cliente de la API de *Google Sheets* y que el programa pueda editar las hojas de cálculo.
- **config.json**: fichero con los principales parámetros de configuración del sistema. Más adelante se le dedica un apartado.
- Otros ficheros de configuración del proyecto: *package.json*, *package-lock.json* y el directorio *node_modules*. Sobre estos ficheros no entraremos porque se generan de manera automática.

El desarrollo de **licencias.js** implementa la secuencia descrita en la Figura 4.2. Se ha utilizado tanto la documentación de *puppeteer* [Pup] como la de *Google APIs* [Gsh]. Sus partes más importantes son:

1. **Simular la navegación** en *Chrome* hasta la URL que contiene los datos. Para ello hay que:

   a) *Instanciar un nuevo navegador y lanzar una nueva página.* Lo vemos en la Figura 4.9. Al lanzar el navegador se pueden configurar parámetros como las dimensiones de la pestaña, si estará o no visible, el tipo de navegador y un *path* para cargar datos de usuario del navegador.

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

```
const browser = await puppeteer.launch({
    userDataDir: "C:\Users\UserName\AppData\Local\Google\Chrome\User Data",
    headless: false,
    executablePath: 'C:\\Program Files\\Google\\Chrome\\Application\\chrome.exe',
    defaultViewport: { width: 1920, height: 1080 }
});
const page = await browser.newPage();
```

Figura 4.9: Código *JavaScript* para instanciar un navegador.

   b) *Simular navegación hasta los datos.* Para acceder a la tabla del consumo de licencias de, por ejemplo, la 5ª organización de *Acoustic* hay que seguir los siguientes pasos: Login >Click en la Organización nº 5 >Click en Usuario >Click en Admin >Click en MMI Report. Esto se consigue mediante un análisis exhaustivo del código HTML de las páginas por las que queremos navegar y haciendo

un uso adecuado de los métodos *page.click()*, *page.type()*, *page.goto()* o *page.waitForSelector()*, entre otros. Además, como buena práctica de desarrollo *software*, se ha creado una función por cada tarea del programa. Esta parte del proceso se ejecuta a través de las funciones:

1) *async function* ***login****(page,* ***usr****,* ***pswd****, loglevel)*. Simula el proceso de *login* en la web de *Acoustic* haciendo uso de los parámetros de usuario y contraseña que recibe.
2) *async function* ***tabledata****(page,* ***org****, loglevel)*. Comprende toda la simulación de la navegación descrita arriba hasta la tabla de datos. El parámetro *org* sirve para seleccionar la organización de la que queremos extraer los datos. Además, como se especificó en el diagrama de flujo, si se falla en el primer intento de acceder hasta la tabla se vuelve a llamar una segunda vez a esta función. Si todo ha ido bien, hemos conseguido llegar hasta la tabla que vimos en la Figura 4.3

2. **Lectura de datos**. En este caso, como vemos en la Figura 4.10, la tabla tiene un identificador único: **#mmiGrid**. Esta tarea del programa se ha separado también en una función:*async function* ***dataexport****(page,loglevel)*. En esta función esperamos a que aparezca el selector #mmiGrid y mediante el método *page.evaluate()* pasamos un *callback* donde añadimos el código necesario para obtener los datos que guardamos en la variable **data**. Lo hacemos a través de un bucle *for* que extrae todos los datos de las celdas de la tabla. Lo vemos en la Figura 4.11.

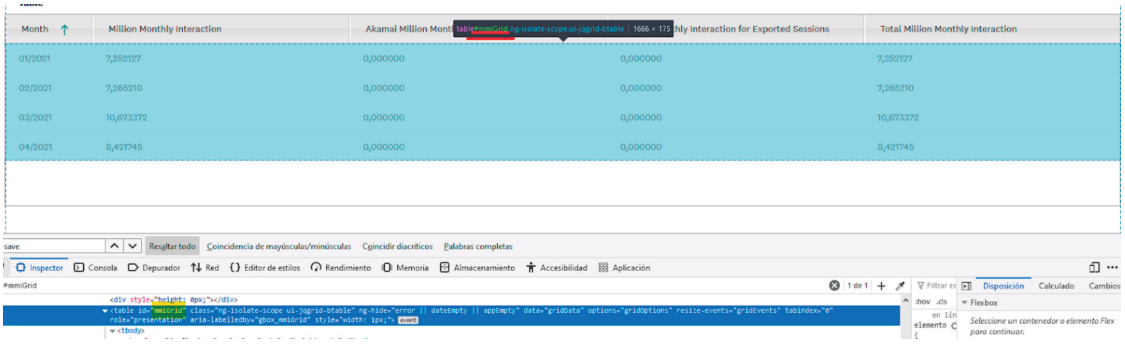

Figura 4.10: Tabla web que contiene los datos de licencias en la página de *Acoustic*.

```
await page.waitForSelector('#mmiGrid');
let tableinfo = await page.evaluate(() => {
    const tds = document.querySelectorAll('#mmiGrid td');

    const data = [];
    for (let td of tds) {
        data.push(td.innerText);
    }
    return data;
});
```

Figura 4.11: Porción de código dentro de la función que extrae datos del HTML.

3. **Actualización de los datos en el *sheet* de *Google***. Antes de programar esta acción, es importante tener habilitada la API de *Google Sheets* tal y como se ha descrito en el apartado anterior. Superado eso, en el *script* se programan las siguientes acciones.

   a) *Importar la API de GSheets y las credenciales del cliente* como se muestra en la Figura 4.12.

```
const { google } = require('googleapis');
const keys = require('./keys.json');
```

Figura 4.12: Importar la API de *GSheets* y las credenciales del cliente.

   b) *Crear el cliente y comprobar que autenticamos sin errores.* Se detalla en la Figura 4.13.

```
//Conexion del cliente con la API de google
const client = new google.auth.JWT( //JSONWebTok ens
    keys.client_email,
    null,
    keys.private_key,
    ['https://www.googleapis.com/auth/spreadsheets']
);
//Comprobamos que hemos autenticado sin errores
client.authorize(function (err, tokens) {
    if (err) {
        if (config.log_level >= 0){
            console.log('⛔ Fallo al conectar con la API de Google Sheets: '+err);
        }

        return;
    } else {
        if (config.log_level >= 2){
            console.log('✅ Conexión correcta con la API de Google Sheets');
        }

        //Actualizamos los datos
        gsupdate(client, tableinfo, config.organization, config.log_level);
    }
});
await browser.close();
```

Figura 4.13: Instanciación del cliente y comprobación de autenticación.

   c) *En caso de autenticación correcta, podemos realizar acciones sobre el Sheet de Google.* Esto lo hacemos a través de la función *async function* ***gsupdate(****cl, input, organization, loglevel****)***. Primero, hay que leer los datos que hay en la la hoja *Meses* como vemos en la Figura 4.14. Luego hay que adecuar los datos que queremos añadir o modificar (a través de la función *function* ***compare(****oldvalues, res****)***. Esta función compara los datos antiguos de la tabla tras leerlos (*oldvalues*) con los datos nuevos extraídos de la web (*res*). En caso de que haya diferencias entre ambos, habrá que actualizar las filas de la tabla que tengan

```
const gsapi = google.sheets({ version: 'v4', auth: cl });
//Leemos los datos que ya contiene la tabla
const getOptions = {
    spreadsheetId: '1hrKrfBeVJeSfeyWrOUrm-9PEn2Qnp2W3YUYVoGLFSxg',
    range: sheetName+'!A2:F7' //Este rango aumenta con los meses
}

let olddata = await gsapi.spreadsheets.values.get(getOptions);
const oldvalues = olddata.data.values; //obtenemos un array bidimensional
```

Figura 4.14: Lectura de datos de la hoja de cálculo.

nuevos datos así como la fecha de actualización. Por último, actualizamos estos valores en la tabla con el comando que vemos en la Figura 4.15.

```
const updateOptions = { //Necesitamos configurar las opciones
    spreadsheetId: '1hrKrfBeVJeSfeyWrOUrm-9PEn2Qnp2W3YUYVoGLFSxg', //elId que se va a modificar
    range: sheetName+'!A2',
    valueInputOption: 'USER_ENTERED',
    resource: { values: updt.values }
}
let resp = await gsapi.spreadsheets.values.update(updateOptions);
```

Figura 4.15: Actualización de datos de la hoja de cálculo.

4. **Finalmente se cierra el navegador y finaliza la ejecución del programa.**

Teniendo todo lo anterior en cuenta, se ha repasado toda la estructura del código que implementa el diagrama de flujo del programa del *Sistema de Licencias*. En este punto del desarrollo, cada vez que ejecutemos el fichero **licencias.js**, el sistema automáticamente actualizará los datos del *sheet* correspondiente. Podemos elegir la organización de la que queremos actualizar los datos o actualizar los datos de las 5 organizaciones que tienen licencia de *Acoustic*. Alcanzado este nivel de funcionalidad, estamos en disposición de configurar los *dashboards* para visualizar estos datos.

Creación de los dashboards en Data Studio

Se puede afirmar que la creación de las gráficas de visualización tiene un componente menos técnico que la del desarrollo de los *scripts*, pero es la pieza más importante del sistema, ya que es la parte **visible**. El *Sistema de Licencias* consta de 1 hoja de datos por cada organización. En nuestro caso, el informe mostrará los datos de 5 organizaciones (habrá que obtener los datos de 5 hojas). Por simplicidad, explicaremos la configuración de un único *dataset* que se replicaría tantas veces como organizaciones queramos representar. Esta configuración consta de 2 tareas fundamentales:

1. **Añadir las fuentes de datos en *Data Studio***. Como los dos servicios son propiedad de *Google*, existe un conector directo con las *Hojas de cálculo de Google* y su configuración es muy sencilla, como vemos en la Figura 4.16. Tras esta conexión, la herrmienta configura un *dataset* formado por tantos **campos originales** como columnas tiene la hoja de cálculo. Para el *Sistema de Licencias*, estos campos originales son suficientes para crear las gráficas.

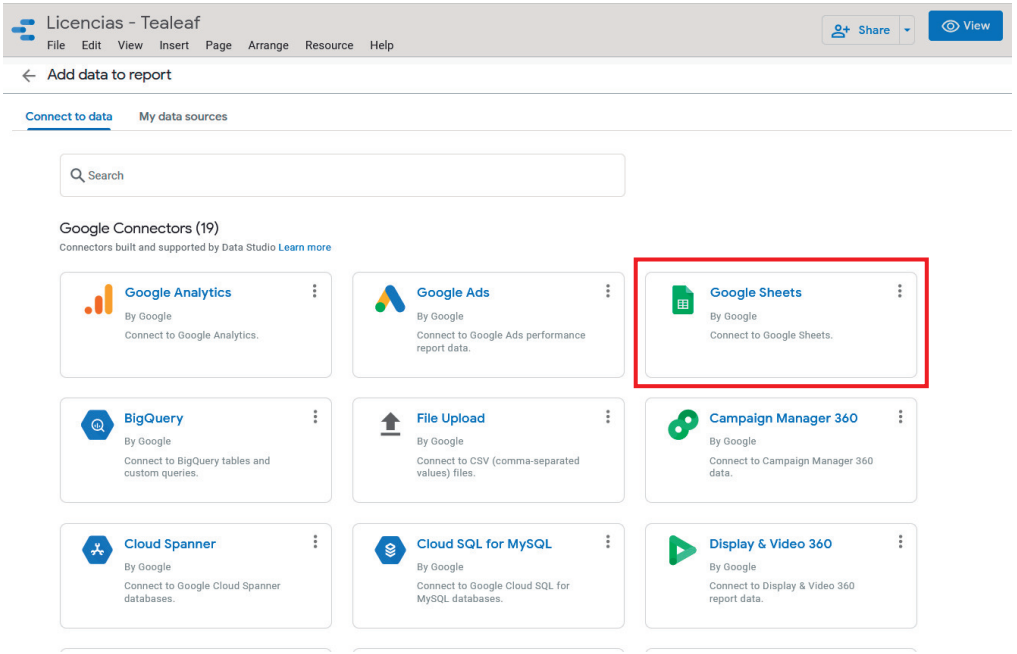

Figura 4.16: Algunos de los conectores de datos de *Data Studio*.

2. **Creación de las gráficas**. Es el proceso con mayor carga creativa de todo el TFG, pero se ha de ser muy cuidadoso. En todo momento debemos tener en cuenta los requisitos impuestos sobre estas gráficas así como el diseño de los gráficos propuesto previamente. La configuración de las gráficas se realiza de una manera muy intuitiva, basta con seleccionar el tipo de gráfica que vamos a representar, escoger el *dataset* asociado a los datos y elegir los campos que se van a mostrar a través de la técnica *drag-and-drop*. En este diseño se fijó que cada organización constaría de 2 gráficas:

   a) Tabla de 3 columnas con los datos más relevantes del consumo.

   b) Evolución mensual del consumo de licencias. Además, en rojo se representa el límite máximo contratado.

El resultado del informe del *Sistema de Licencias* se muestra en la Figura 4.17. Hay que tener en cuenta que en la figura sólo se muestran 3 organizaciones. Por ejemplo, vemos que la *Organización 1* ha excedido el límite de licencias durante los dos últimos meses. Esto debe ser una alerta para que el Equipo de Negocio tome medidas con el objetivo de solucionar este problema.

Figura 4.17: Informe de visualización para 3 organizaciones del *Sistema de Licencias*.

4.4.2. Sistema de Rendimiento

Tras el desarrollo del *Sistema de Licencias*, se implementó el *Sistema de Rendimiento*. Se trata de un sistema que tiene muchas similitudes con el anterior, pero aumenta un poco en complejidad. Por ello, en esta subsección vamos a señalar las principales diferencias que se han tenido en cuenta durante el desarrollo.

Configuración de Tablas de Tealeaf

A diferencia que en el *Sistema de Licencias*, ahora las tablas que contienen **el dato** no están montadas por la herramienta, sino que se han tenido que crear dos *reports* con los datos de los tiempos de rendimiento. En esta parte del desarrollo lo que se consigue es hacer

que los datos que nos interesan estén disponibles en la web para que luego el sistema pueda llegar hasta ellos y extraerlos. En la Figura 4.18 se muestra la tabla creada en *Acoustic* con los *tiempos generales* y sus campos asociados. De manera análoga se construye la tabla de los *tiempos de render*. Como ya vimos, estos datos son los **campos originales** que definimos en las Tablas 4.2 y 4.3 durante el proceso de diseño. Esto tiene sentido, porque son los datos que vamos a almacenar en los *sheets*.

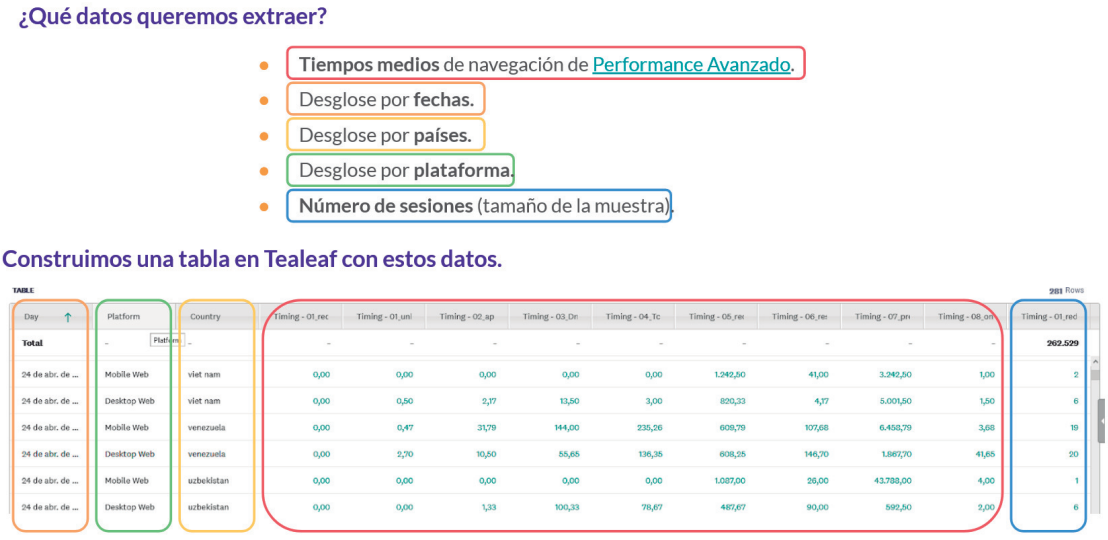

Figura 4.18: Tabla creada en *Tealeaf* que contiene los *tiempos generales*.

Tras esta primera configuración, tenemos disponibles los datos que nos interesan a través de la web.

Creación del proyecto de Node.js

Como ya se adelantó en la fase de diseño, el flujo del programa de ambos sistemas es muy similar. La estructura de ficheros es la misma con la única salvedad de que ahora el *script* principal pasa a llamarse **rendimiento.js**. Las acciones que implement son:

1. **Simular la navegación.** Este *script*, de manera análoga al que vimos para el *Sistema de Licencias*, realiza el *login* en la herramienta y accede hasta los datos que, en este caso, se encuentran en un lugar diferente de la web. Esto se hace con la función *async function* ***performancedata****(page, org,* ***tr****,* ***td)*** que, a través de los parámetros *tr* y *td*, permite escoger el día del que se quieren extraer los datos.

2. **Lectura de datos.** Se hace a través de una función muy similar a la desarrollada en el sistema anterior. La única diferencia es que podemos escoger el día del que queremos extraer los datos.

3. **Actualización de los datos en el *sheet* de *Google*.** En el caso del *Sistema de Rendimiento*, este proceso es mucho más sencillo que para las licencias, pues no hay que leer los datos del histórico almacenados en el *sheet* ni compararlos con los

actuales, sino que simplemente hay que añadir nuevas filas a la tabla con los nuevos datos leídos. Esto es porque el *Sistema de Licencias* va actualizando los datos de las filas de su tabla a medida que pasa el mes, mientras que el *Sistema de Rendimiento* actualiza sus tablas con datos diarios consolidados que no van a cambiar en siguientes lecturas.

Estos 3 pasos han de ejecutarse 2 veces para una sola ejecución del *script*: primero para los datos de *tiempos generales* y luego para los *tiempos de render*. Cada una de estas ejecuciones añadirá datos a un *sheet* diferente configurando así los dos *datasets* que forman parte del *Sistema de Rendimiento*.

Creación de los dashboards en Data Studio

Antes de generar los informes, hay que utilizar el conector de las hojas de cálculo de Google para añadir las 2 fuentes de datos a *Data Studio*. Una vez hecho esto, se llevaron a cabo las siguientes tareas.

1. **Transformación de las fuentes de datos.** Los datos que albergan las hojas de cálculo son los **campos orginales**, pero, como definimos en la fase de diseño, los datos que queremos mostrar en las gráficas son a nivel agregado a través de las **métricas calculadas.** Entonces, debemos añadir estas métricas para configurar nuestros *datasets* de *Data Studio*. Para los datos de tiempos de *render* usamos lo descrito en la Tabla 4.3 hasta obtener lo que se muestra en la Figura 4.19.

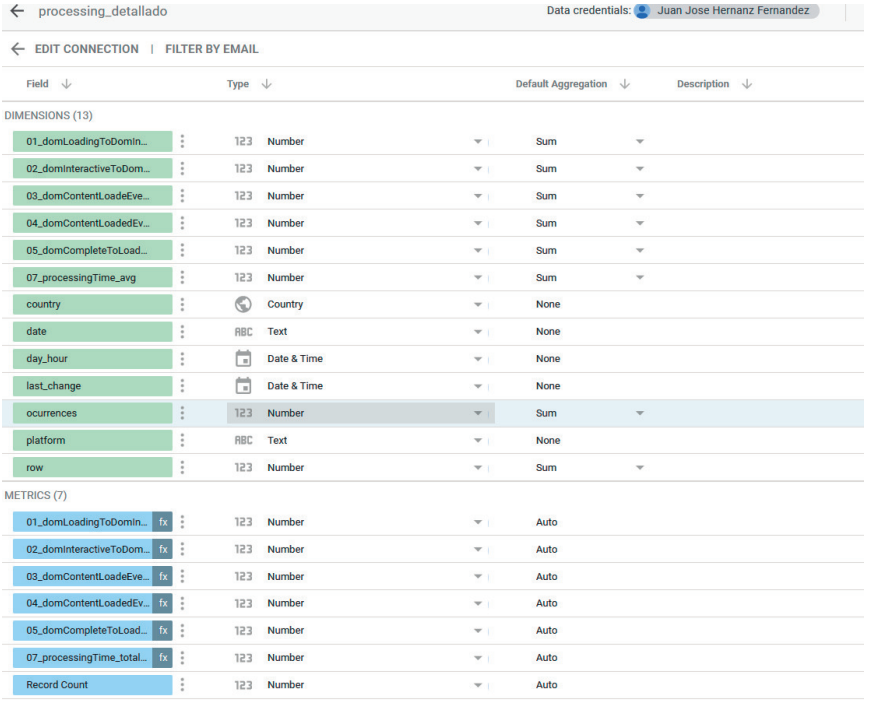

Figura 4.19: Fuente de datos: *processing_detallado* configurada en *DataStudio*.

Hacemos lo propio con los datos generales a partir de la Tabla 4.2 y llegamos a lo que vemos en la Figura 4.20

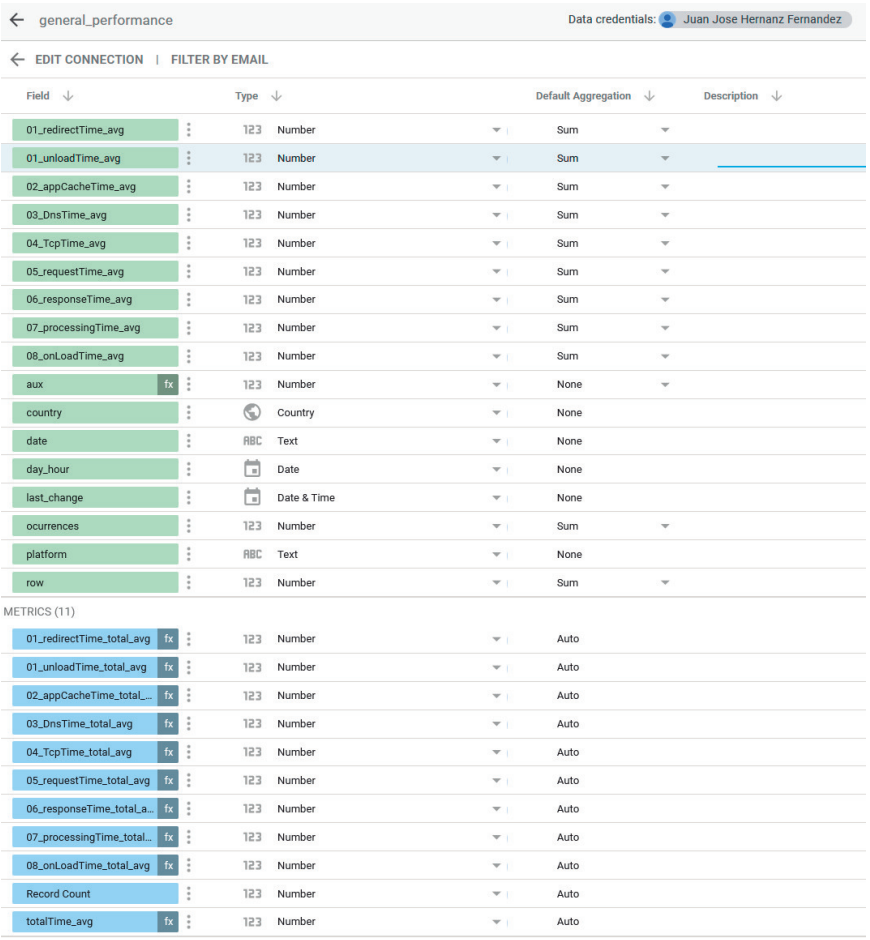

Figura 4.20: Fuente de datos: *general_performance* configurada en *DataStudio*.

2. **Creación de los informes**. Se generan 3 informes para llevar un seguimiento de las métricas calculadas de los tiempos generales y los tiempos de render. El objetivo principal es ver la evolución diaria de estos tiempos, y poder analizarlos por país y plataforma. En bases a las propuestas de la fase de diseño, mi misión fue crear los siguientes 3 informes:

   - **Informe de tiempos generales**. Lo vemos en la Figura 4.21. Titulado como *Acoustic Tiempo de Carga Avanzado*. En la parte superior muestra el desglose de tiempos que define la *API Navigation Timing* que ya hemos comentado otras veces. Además, se muestran unos controles para filtrar los datos del informe por fecha, plataforma y país. Justo debajo de esos controles, se representa el tamaño de la muestra que estamos utilizando en el informe. Para este ejemplo, la muestra es de 4,6 milliones de sesiones y supone un 10,4 % de incremento repecto al período anterior. En la parte central se muestra una gráfica de la evolución temporal de todos los tiempos representados como barras apiladas. La acumulación de estos tiempos es la variable *Total Time* que se dibuja como

una línea roja. Finalmente, en la parte inferior del informe se sitúan 9 gráficas de evolución temporal de cada uno de los tiempos que contribuyen al tiempo total. Se indica también el tiempo medio de cada uno en un recuadro gris. Como podemos ver, la mayor contribución al tiempo de carga es el **Processing Time**, por ello la necesidad de desglosarlo en el segundo informe.

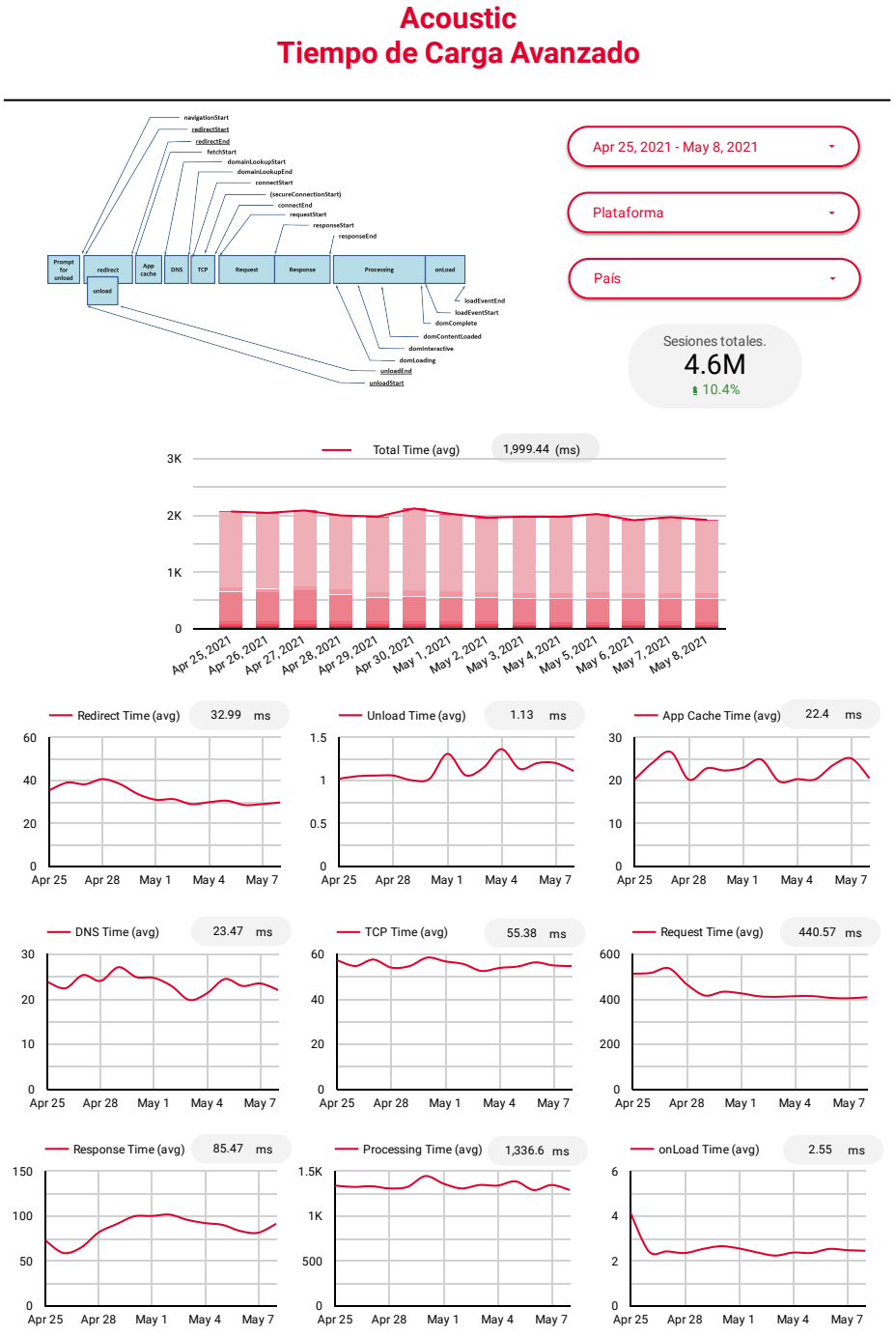

Figura 4.21: Informe de tiempos generales del *Sistema de Rendimiento*.

- **Informe del *processing time* detallado**. Este informe sigue exactamente la misma estructura que el anterior, pero únicamente para los tiempos que contribuyen al *processing time*. Lo vemos en la Figura 4.22. La necesidad de este informe surge de profundizar en este tiempo de procesamiento, que es el más crítico entre los tiempos de carga de una página web.

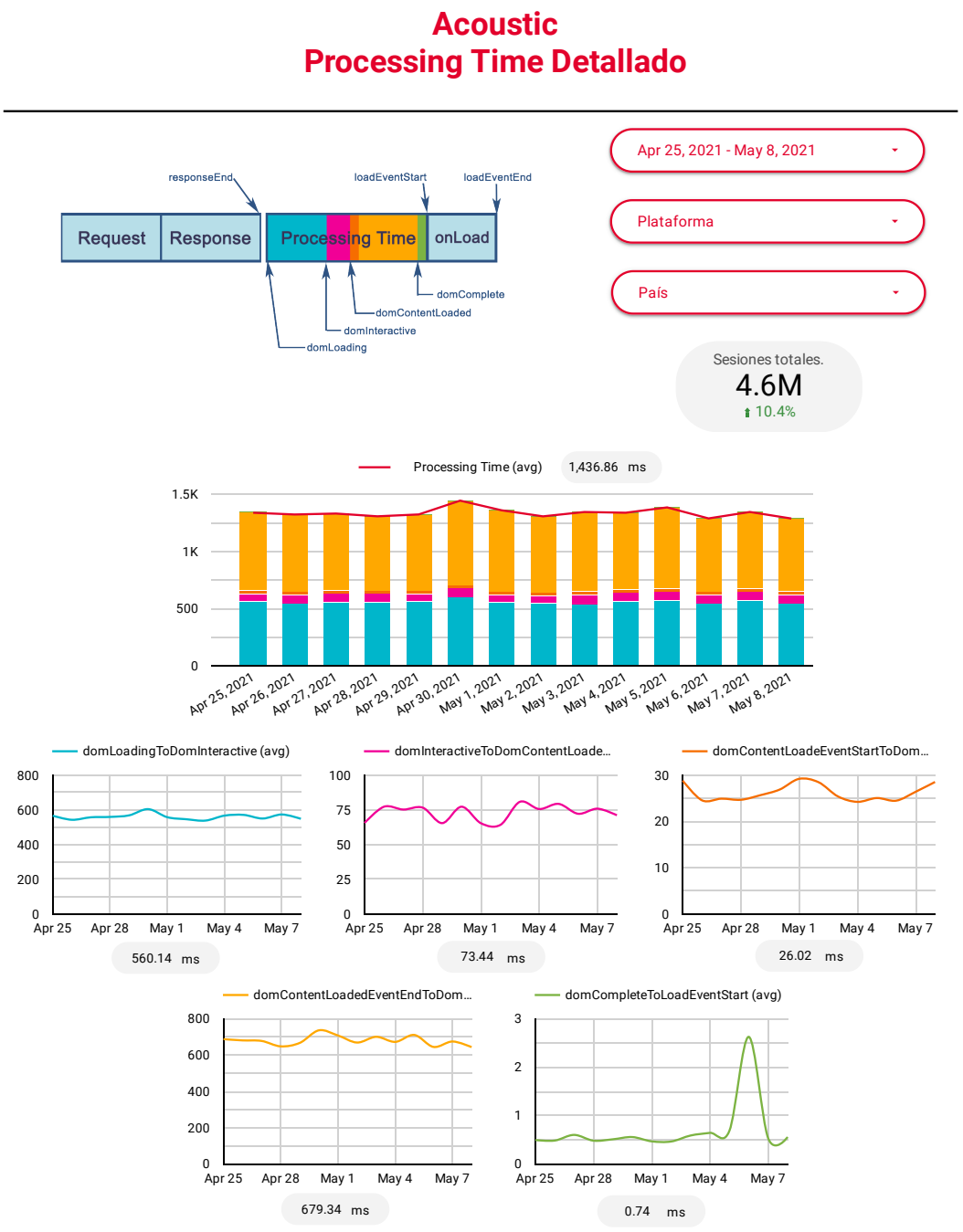

Figura 4.22: Informe del *processing time* detallado.

- **Informe de Control**. El diseño de este informe no estaba tan definido, aunque sí los datos que se querían mostrar. En la Figura 4.23 se muestra el resultado final.

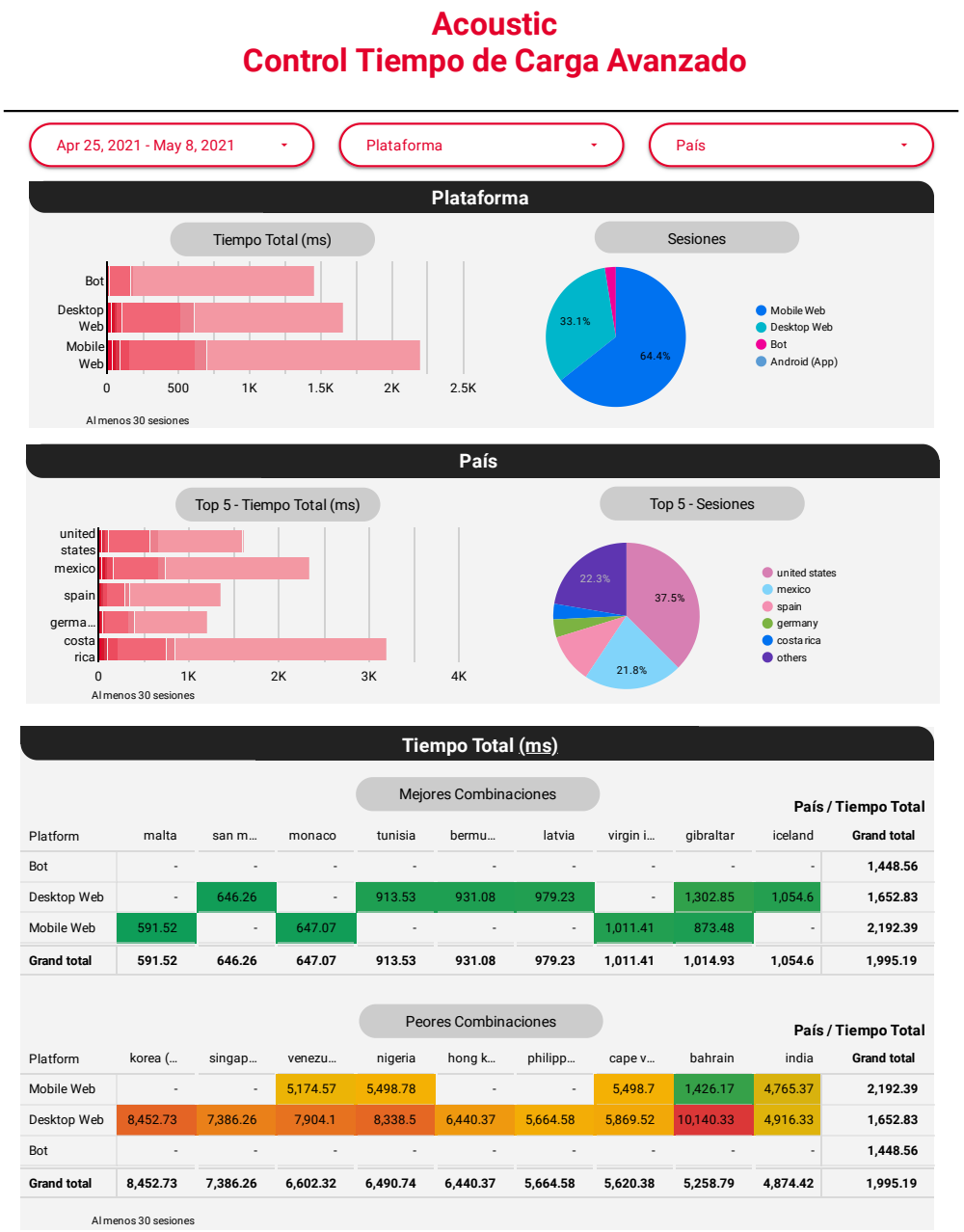

Figura 4.23: Informe de Control de Tiempo de Carga del *Sistema de Rendimiento*.

Como vemos, en la parte superior se vuelven a mostrar los 3 controles de filtrado de datos: rango de fechas, plataforma y país. En la parte superior se muestra el tiempo de carga total desglosado para las 3 plataformas principales. Además, en un diagrama de sectores, se completa la información representando

el volumen de sesiones que acumula cada plataforma sobre el total. De estas gráficas podemos interpretar que: si bien las sesiones realizadas con *bot* sobre la web bajo estudio son las que menores tiempos de carga experimentan, estas representan un volumen muy bajo de sesiones respecto del total. De forma análoga representamos los tiempos y el volumen de sesiones desglosados por países. Por último, en la parte inferior del informe, se muestran dos tablas de doble entrada (país/platforma) que mapean el tiempo total de carga. Las mejores combinaciones serán aquellas que tienen menores tiempos de carga (en verde) y las peores, las que tienen mayores tiempos de carga (en rojo). Por ejemplo, podemos descubrir que las sesiones móviles desde Malta tardan, en media, menos de 0,6 segundos en cargar la página, mientras que las sesiones realizadas desde ordenador en Baréin acumulan tiempos de carga promedio superiores a 10 segundos.

4.4.3. Ficheros config.json

Como fase final del desarrollo, se decidió separar las variables más relevantes de cada proyecto en un fichero denominado **config.json**. Con esto se cumple el FR12 que decía que *"El sistema deberá incluir unos parámetros de configuración de la ejecución. Los podrá modificar eldesarrollador a través del ficheroconfig.json"*. El uso de este tipo de ficheros es muy habitual en el desarrollo *software* de sistemas modernos, ya que permite centralizar todos los paramétros de configuración en un único lugar y hace los sistemas más flexibles desde el punto de vista del desarrollador. A modo de ejemplo, se proporciona la Figura 4.24 en la que se ven los parámetros que podemos configurar para la solución del *Sistema de Licencias*.

Figura 4.24: Fichero de configuración *config.json* para el *Sistema de Licencias*.

En la Figura 4.24, vemos como podemos configurar las **credenciales** que se usan en el *login* de *Acoustic*, la **organización** sobre la que queremos obtener los datos, diferentes opciones del **navegador** o el ***log_level***, al cual dedicamos la siguiente Sección.

4.4.4. Sistema de LOG

En un esfuerzo por mejorar la calidad de la información que emite el *script* y como práctica para mejorar la depuración de la secuencia del programa, he configurado un sencillo sistema de LOG. Gracias a ello, cumplimos también con el NFR04. A través de la función: *function* ***setlog****(page,* ***level)***, se define la cantidad de información que va a emitir el programa al ejecutarse, a través del argumento *level*. Este nivel puede ser configurado por el usuario a través del valor ***log_level*** del fichero *config.json*. En la Figura 4.25, se representa la tabla que se creó para definir los diferentes niveles de LOG que se iban a implementar asociados a los eventos sobre los que se emitiría información.

log_level	Mensaje	Descripción
Eventos		
0	✖ ${error}	Emitted when the page emits an error event (for example, the page crashes)
0	✖ ${error}	Emitted when a script within the page has uncaught exception
3	☑ Page is loaded	Emitted when the page is fully loaded
3	☑ Frame is attached	Emitted when the page attaches a frame
3	⧖ Timestamp added at ${data.metrics.Timestamp}	Emitted when a script within the page uses `console.timeStamp`
3	☑ Frame is detached	Emitted when the page detaches a frame
3	☑ Page is closed	Emitted after the page is closed
4	ℹ ${message.text()}	Emitted when a script within the page uses `console`
4	ℹ ${dialog.message()}	Emitted when a script within the page uses `alert`, `prompt`, `confirm` or `beforeunload`
4	ℹ New page is opened'	Emitted when a new page, that belongs to the browser context, is opened
4	ℹ Request: ${request.url()}	Emitted when the page produces a request
4	⚠ Failed request: ${request.url()}	Emitted when a request, which is produced by the page, fails
4	ℹ Finished request: ${request.url()}	Emitted when a request, which is produced by the page, finishes successfully
4	ℹ Response: ${response.url()}	Emitted when a response is received
4	ℹ Worker: ${worker.url()}	Emitted when the page creates a dedicated WebWorker
4	ℹ Destroyed worker: ${worker.url()}	Emitted when the page destroys a dedicated WebWorker
Lógica del programa		
0 ó 1 dependiendo de dónde este (será 0 en los catch y en los segundos intentos y será 1 en los primeros intentos).	⛔	Emitted when an error occurs in the logic of the *script*
2	◆	Emitted when a click is made.
2	💬	Emitted when field y filled.
2	✅	Emitted when a function succeed.
3	🚧	Emitted when the program is wating.

Figura 4.25: Eventos asociados a los diferntes niveles de LOG. Elaboración propia.

Se diseñó un sistema de 5 niveles de información que son acumulativos. Es decir, si escogemos el nivel 2, se mostrarán los mensajes asociados al nivel 0, al nivel 1 y al nivel 2. Los niveles que recogen en la Figura 4.25 son los siguientes:

- **log_level = 0 (rojo):** errores fatales producidos en el navegador o en la lógica del programa que hacen imposible seguir con la secuencia. El sistema no se terminará de ejecutar en ningún caso.

- **log_level = 1 (naranja):** errores en alguna función de la lógica del programa que podrían no terminar con su ejecución si está concebido dentro de la secuencia.

- **log_level = 2 (verde):** avances fluidos de la lógica del programa. Avisan de los clicks, las escrituras de campos, o los éxitos de ejecución de las funciones.

- **log_level = 3 (azul):** se muestran además los tiempos de espera de la secuencia del programa.

- **log_level = 4 (morado):** asociados con la información relativa a eventos relacionados con la simulación de la página del navegador.

Para más información sobre los eventos del sistema de LOG, se recomienda leer la columna de *description* redactada en inglés.

A modo ilustrativo, se muestran tres capturas de la información que proporciona la terminal al ejecutar el *script* en diferentes escenarios. En la Figura 4.26, se representa la salida por la terminal de la ejecución con errores de *licencias.js*, para un nivel 2 de LOG. En la Figura 4.27, se representa la salida de la terminal del mismo caso anterior, pero en este caso ejecutado sin errores. Podemos ver en la Figura 4.28 que la cantidad información que muestra por terminal, amplía para una ejecución con nivel 3 de LOG.

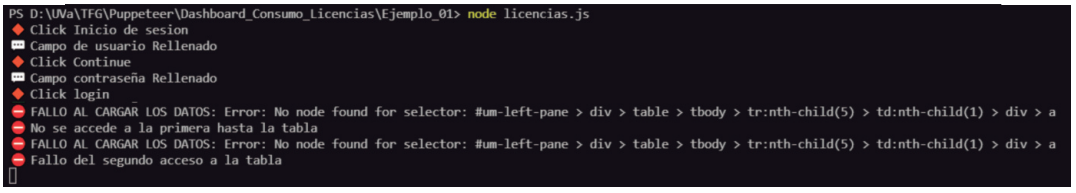

Figura 4.26: Captura de la terminal al ejecutar *licencias.js* con un nivel 2 de LOG y para unas credenciales incorrectas.

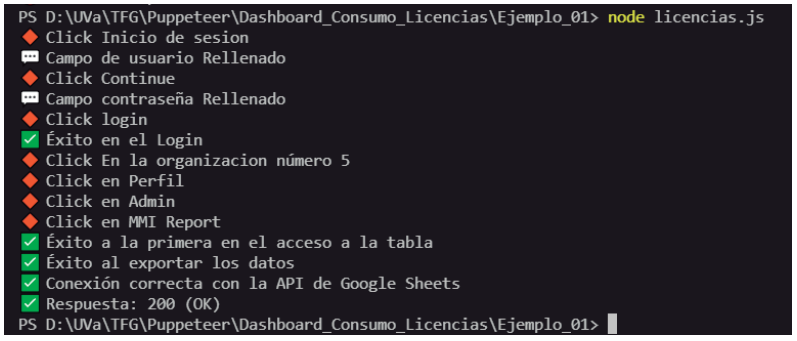

Figura 4.27: Captura de la terminal al ejecutar sin errores *licencias.js* con un nivel 2 de LOG

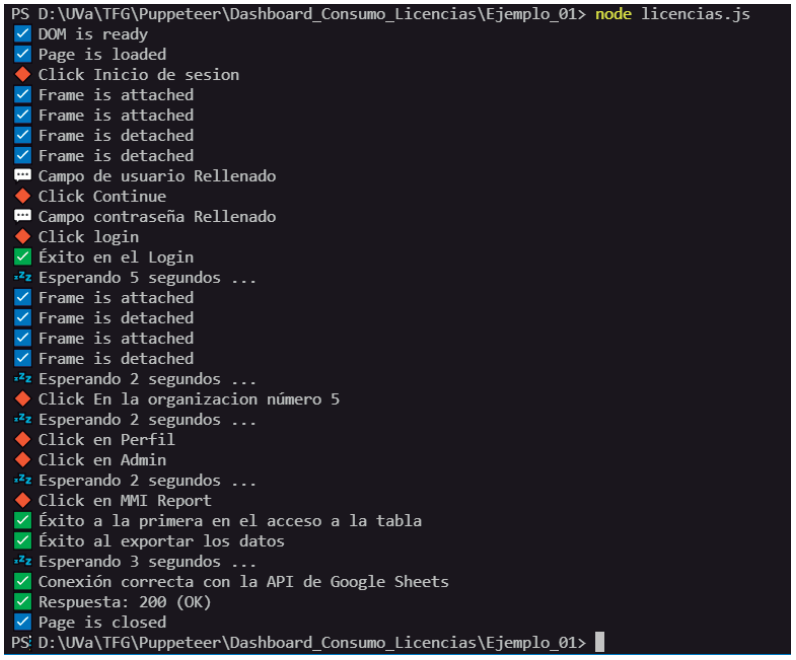

Figura 4.28: Captura de la terminal al ejecutar sin errores *licencias.js* con un nivel 3 de LOG

4.5. Despliegue

A través del proceso de **Despliegue** se consigue que los sistemas de información desarrollados estén en funcionamiento y disponibles para su uso. Por la metodología de desarrollo de la empresa, esta fase es llevada a cabo por el **Equipo de Sistemas** y no por los **Desarrolladores**. Es por ello que esta fase no es contenido de este TFG. El despliegue de los sistemas tiene pensado llevarse a cabo en una de las máquina de la infraestructura interna de la empresa. La intención es ubicar los proyectos en una de esas máquinas y lanzarlos con cierta periodicidad a través de un *software* de **programación de tareas**. El despliegue de los sistemas está planteado realizarse en el corto plazo por no suponer a penas un incremento del uso de recursos.

4.6. Conclusiones

Con este Capítulo se pone fin a las fases de desarrollo de los sistemas implementados durante el Trabajo de Fin de Grado. Sin duda las fases de Diseño y Desarrollo han sido las que han concentrado la mayor parte de los esfuerzos. Además, es importante mencionar que, en muchos de los casos, estas fases han sucedido de forma paralela e iterativa. Por ejemplo, nos dimos cuenta de que necesitábamos representar algún dato a mayores en los informes una vez estaban desarrollados los *scripts*, lo que provocó modificaciones sobre todos los elementos del sistema (incremento de los datos mostrados en la web, incremento de las columnas de la tabla, incremento de filas de datos, etc.).

Aunque no se ha mencionado, se han encontrado algunos **obstáculos** durante la etapa de Desarrollo. El mayor de ellos fue el fallo que se producía al simular el *login* en *Acoustic*, ya que utiliza procesos de autenticación en dos pasos. Esto obligó a cargar las cookies del usuario en el navegador para conservar datos del usuario y conseguir sobrepasar este proceso. Otro de los problemas que más se ha manifestado durante el desarrollo del código es la dificultad de trabajar con funciones asíncronas, así como con simuladores de navegación. La red es dinámica y cambios en la conexión pueden provocar que la ejecución del mismo programa, en momentos distintos, provoque resultados distintos a causa de una conexión más lenta, por ejemplo. Esto ha hecho necesario una programación muy robusta a fallos y con un gran número de comprobaciones a lo largo delo flujo del programa.

Por último, considero de gran relevancia mencionar el resultado de los informes finales. Todo el sistema que se ha construido por detrás para obtener los datos carece de sentido si no se consigue obtener **valor** de ellos. Los gráficos son el producto final y deben recibir la atención que merecen. Para su desarrollo hay que tener siempre en mente los requisitos impuestos por el usuario final. Además, en nuestro caso, teníamos reuniones periódicas para obtener *feedback* y reajustar los *dashboards* hasta obtener los que se han presentado en este Capítulo. Sin duda, los usuarios finales han quedado muy satisfechos y se ha conseguido obtener valor de los datos. Por ejemplo, gracias al informe del *Sistema de Licencias* de la Figura 4.17, el Equipo de Negocio ha detectado que, durante los dos últimos meses con datos, se ha sobrepasado la cuota de licencias, lo que ha desembocado en una serie de acciones encaminadas a revertir esta situación.

Capítulo 5

Conclusiones y líneas de trabajo futuro

5.1. Conclusiones del trabajo realizado

En los últimos años, las empresas están dirigiendo una parte importante de sus activos a la transformación digital y la automatización de procesos empresariales. Gracias a ello, las soluciones RPA se han ido haciendo un hueco de unos años a esta parte adquiriendo una importancia cada vez mayor. En este Trabajo de Fin de Grado, se ha mostrado todo el proceso llevado a cabo para la creación de dos soluciones de esta naturaleza como resultado de una combinación de diferentes tecnologías actuales. En primer lugar, queda demostrado que, gracias a la inmensa cantidad de tecnologías disponibles en la actualidad, es posible desarrollar este tipo de soluciones a coste casi nulo. Además, tras obtener los informes de visualización, uno se da cuenta de la relevancia que tiene presentar la información de una forma clara y fácil de interpretar. Los mismos datos, presentados de dos formas diferentes, aportan un valor muy distinto. Sin duda, el momento de interactuar con los informes completamente montados, es una de las cosas más emocionantes que he experimentado durante todo el desarrollo del sistema.

Sin embargo, durante el camino también han aparecido obstáculos que ponían en entredicho el consecución del objetivo. El mayor de ellos, sin duda, ha sido el problema derivado de la doble autenticación que aplica *Tealaeaf* al realizar el *login* en su web. Su solución supuso un gran esfuerzo en entender qué ocurría y decidir qué alternativa era la menos mala. Aún en el estado actual del proyecto, este obstáculo impone ciertas limitaciones, sobre todo a la hora de automatizar la ejecución de los sistemas.

El hecho de haber trabajado en *JavaScript* y, además, utilizando una programación asíncrona, ha supuesto un importante reto para mí, pues no sólo he sido capaz de producir todo el código, sino que lo he hecho a la vez que aprendía este lenguaje. Los errores más comunes al trabajar con este tipo de código que simula acciones del navegador son los que están relacionados con las situaciones cambiantes de la web. A veces la conexión es más lenta, a veces el HTML de una página es modificado, a veces un mismo proceso lleva más tiempo... Por todo ello, como se ha comentado previamente, he trabajado en producir un

código lo más robusto a fallos posible.

Finalmente, considero que ha sido muy enriquecedor enfretarme a la tarea de montar un sistema desde cero que, además, incorpora tecnologías distintas y actuales. Probablemente existan maneras más eficientes de implementar estos sistemas, pero haberlo hecho de esta forma ha sido muy enriquecedor para mi formación. Tengo claro además que, haber contado con el apoyo de mis compañeros de empresa, ha hecho que este trabajo sea mucho más sencillo de llevar, sobre todo en los momentos de mayores bloqueos.

5.2. Líneas de trabajo futuro

El nivel de los sistemas alcanzados hasta el momento cumple con casi la totalidad de Requisitos recopilados durante las primeras fases. No obstante, existen sobre todo dos puntos sobre los que se debería trabajar en un corto plazo.

Por un lado, y el más importante, es **llevar a cabo la fase de Despliegue** tanto del *Sistema de Rendimiento* como del *Sistema de Licencias*. Esto, como ya se explica en la Sección 4.5, se ha de hacer en colaboración con el **Equipo de Sistemas**, ya que se quiere poner en funcionamiento en alguna máquina de la infraestructura de la propia empresa. Además, este proceso no se ha llegado a completar todavía por las limitaciones (ya comentadas) relativas a la doble autenticación durante el proceso de *login* de los sistemas implementados. Superado este paso, los sistemas alcanzarían un estado tal en el que funcionasen de manera totalemente automática sin la necesidad de la intervención humana.

Por otro lado, uno de los objetivos con los que se originó el *Sistema de Rendimiento* era poder combinar los tiempos que se visualizan con los ***Core Web Vitals*** de *Google* mencionados en el Capítulo 1. Esto enrequicería mucho los datos que ya se miden y mejoraría la calidad de la información de los tiempos de carga de la página web. Se ha realizado una pequeña investigación acerca de la viabilidad de está combinación, pero actualmente no existen herramientas que permitan medir los *Core Web Vitals* con la precisión necesaria.

A un nivel más general y desde el punto de vista de **explotación** de la empresa, los resultados del desarrollo de estos sistemas resultan muy interesantes y suponen una vía real para ofrecer **servicios** a los clientes de la entidad.

Referencias

[Aco] *Tealeaf Acoustic Developers Documentation.* [Website]. Último Acceso: junio 2021. Disponible en: https://developer.goacoustic.com/acoustic-exp-analytics/docs/.

[CG79] T.Sarson C. Gane. *Structured Systems Analysis.* Prentice Hall, 1979.

[FDGS18] Ticiana Agustina Alvarado Wall Federico Del Giorgio Solfa, Guido Amendolaggine. New paradigms for product design. design thinking, service design and user experience. *Arte e Investigación*, 2018.

[Gsh] Google api sheets documentation. [Website]. Último Acceso: junio 2021. Disponible en: https://developers.google.com/sheets/api/quickstart/nodejs.

[Has06] Tractinsky Noam Hassenzahl, Marc. User experience - a research agenda. *Behaviour Information Technology*, 2006.

[HK10] Mattias Skarin Henrik Kniberg. *Kanban and Scrum - Making the Most of Both.* InfoQ, 2010.

[JSV17] Joey F. George Joseph S. Valacich. *Modern Systemas Analysis and Design.* Pearson, octava edición, 2017.

[Koc18] Katharina Koch. What is business intelligence? and why do i need to know?, Septiembre 2018. [Website]. Último Acceso: mayo 2021. Disponible en: https://www.uipath.com/blog/what-is-business-intelligence-why-do-i-need-to-know.

[Nod] Node.js documentation. [Website]. Último Acceso: junio 2021. Disponible en: https://nodejs.org/en/docs/.

[Pup] Puppeteer developers documentation. [Website]. Último Acceso: junio 2021. Disponible en: https://pptr.dev/.

[RS14] Efraim Turban Ramesh Sharda, Dursun Delen. *Business Intelligence and Analytics: Systems for Decision Support.* Pearson, décima edición, 2014.

[Sch02] Ken Schwaber y Mike Beedle. *Agile Software Development with Scrum.* Prentice Hall, 2002.

[SM19] Durgesh Kumar Jaiswal Somayya Madakam, Rajesh M. Holmukhe. The future digital work force: Robotic process automation (rpa). *Journal of Information Systems and Technology Management – Jistem USP*, 2019.

[SN08] Paul Gray Solomon Negash. *Handbook on Decision Support Systems 2.*, chapter 45 Business Intelligence., páginas 175–193. Springer, Berlin, Heidelberg, 2008.

[Sny15] Chase Snyder. Real-user monitoring's next frontier: Context, context, context!, Mayo 2015. [Website]. Último Acceso: mayo 2021. Disponible en: https://www.extrahop.com/company/blog/2015/real-user-monitorings-next-frontier-context/.

[Ste15] Rod Stephens. *Beginning Software Engineering.* John Wiley Sons, Inc., primera edición, 2015.

[SvB18] Bart Baesens Seppe vanden Broucke. *Practical Web Scraping for Data Science*, chapter 1: Introduction (Web Scraping Basics), páginas 1–77. Apress, 2018.

[Tau20] Tom Taulli. *The Robotic Process Automation Handbook.* Apress, 2020.

[vdA18] Bichler Martin Heinzl Armin van der Aalst, Wil M. P. Robotic process automation. *Business Information Systems Engineering*, May, 2018.

[W3C12] Navigation timing. W3c recommendation, World Wide Web Consortium, 2012.

[Wal19] Philip Walton. User-centric performance metrics, Noviembre 2019. [Website]. Último Acceso: mayo 2021. Disponible en: https://web.dev/user-centric-performance-metrics/.

[Wal20] Philip Walton. Web vitals, Abril 2020. [Website]. Último Acceso: mayo 2021. Disponible en: https://web.dev/vitals/.

[Web] Speed index. [Website]. Último Acceso: mayo 2021. Disponible en: https://docs.webpagetest.org/metrics/speedindex/.