TRABAJO FIN DE GRADO

INGENIERÍA EN INFORMÁTICA

Gestor de Trending Topics y Noticias

Estudio y Visualización de Trends de Twitter

Autor

Nikita Stetskiy Stetskiy (alumno)

Directores

Juan Francisco Huete Guadix (tutor)

Carlos Alberto Cruz Corona (cotutor)

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

—

Granada, 9 de noviembre de 2022

NIKITA STETSKIY STETSKIY (ALUMNO):
Gestor de Trending Topics y Noticias
Estudio y Visualización de Trends de Twitter.

Supervisado por:
JUAN FRANCISCO HUETE GUADIX (TUTOR)
CARLOS ALBERTO CRUZ CORONA (COTUTOR)

Granada, 9 de noviembre de 2022

DECLARACIÓN

D. **Juan Francisco Huete Guadix (tutor)**, Profesor del Área de XXXX del Departamento de Ciencias de la Computación e Inteligencia Artificial de la Universidad de Granada.

D. **Carlos Alberto Cruz Corona (cotutor)**, Profesor del Área de XXXX del Departamento de Ciencias de la Computación e Inteligencia Artificial de la Universidad de Granada.

Informan:

Que el presente trabajo, titulado ***Gestor de Trending Topics y Noticias, Estudio y Visualización de Trends de Twitter***, ha sido realizado bajo su supervisión por **Nikita Stetskiy Stetskiy (alumno)**, y autorizamos la defensa de dicho trabajo ante el tribunal que corresponda.

Y para que conste, expiden y firman el presente informe en Granada a 9 de noviembre de 2022.

Los directores:

Juan Francisco Huete Guadix (tutor)
Carlos Alberto Cruz Corona (cotutor)

RESUMEN

PALABRAS CLAVE: Aplicación Web, Arquitectura de capas, Modelado API REST, Trending Topics, Noticias, Diseño Responsive.

. . .

El principal enfoque de este Trabajo de Fin de Grado se basa en la aportación de información útil, resumida y de fácil alcance para el usuario final, sin que esta sea abrumante y difícil de leer.

A día de hoy, las aplicaciones web sociales más dominantes del mercado intentan captar y mantener al usuario el mayor tiempo posible. Lo consiguen aportando cantidad ingente o casi infinita de información. Esta información se actualiza en cada momento, dando la posibilidad de mantenernos partícipes del mundo social en cada instante.

Dada la atosigante abundancia de información he decidido hacer una aplicación web que proporcione información de una manera mucho más simple sobre tendencias actuales. De esta manera, los usuarios no tendrán que estar tan pendientes en mantenerse actualizados, sino que al final del día tendrán simplificada o resumida la información más destacable y otros datos útiles en base a la tendencia.

La idea que se intenta realizar con este proyecto es una gestión de tendencias a partir de la API de Twitter y la extracción de contenido relevante en base a la tendencia, como puede ser el volumen de publicaciones, palabras claves, sentimientos generales o noticias relevantes.

El proyecto involucra todos los conocimientos, tanto en la ingeniería informática como fuera de ella, que he podido adquirir a lo largo de mi ciclo de aprendizaje del ámbito informático. Además de estudiar la parte programable y funcional de la aplicación, se realiza un estudio riguroso sobre la planificación, estructura y arquitectura ideal al proyecto.

Entrando más en detalle en los conocimientos aplicados, el proyecto se planteará con una arquitectura por capas y un modelo API REST. Los lenguajes de programación usados en su mayoría serán JavaScript, Python y el tipo de bases de datos será NoSQL.

No obstante, a lo largo del proyecto se estudiará la razón de estas decisiones y se demostrará cómo combinando estas tecnologías, se llega a realizar una única aplicación en cara al usuario final.

ABSTRACT

KEYWORDS: Web Application, Layered Architecture, REST API Modelling, Trending Topics, News, Responsive Design.

. . .

The main focus of this Final Degree Project is based on providing useful information, summarized and easily accessible to the end user, without it being overwhelming and difficult to read.

To this day, the most dominant social web applications on the market try to engage and keep the user as long as possible. They achieve this by providing a huge or almost infinite amount of information. This information is updated everytime, giving the possibility of staying involved in the social world almost forever.

Given the overwhelming abundance of information, I have decided to make a web application that provides you with information in a much simpler way about current trends. In this sense, users will not have to be so focused on keeping up to date, but at the end of the day they will have summarized the most valuable information and useful data based on the trend.

The idea that this project tries to carry out is the management of trends from the Twitter API and the extraction of relevant content based on the trend, such as the volume of publications, keywords, general sentiments or relevant news.

The project involves all the knowledge, both in computer engineering and outside of it, that I have been able to acquire throughout my learning cycle in the computer field. In addition to studying the programmable and functional part of the application, a rigorous study is carried out on the most ideal planning, structure and architecture to the project.

Going into more detail in the applied knowledge, the project will be approached with a layered architecture and an API REST model. The mostly used programming languages will be JavaScript, Python and the type of databases will be NoSQL.

However, throughout the project the reason for these decisions will be studied and it will be demonstrated how by combining these technologies, a single application is achieved for the end user.

AGRADECIMIENTOS

Me gustaría poder agradecer a todas las personas que han tenido que escuchar hablarme sobre este proyecto y sobre todo a las personas que me han ayudado a realizarlo.

Quisiera expresar un especial agradecimiento a mis supervisores, Juan y Carlos, por la oportunidad de permitirme trabajar en un proyecto con ambición personal. Sin olvidar el trabajo y esfuerzo de aquellos profesores cuya dedicación ha conseguido que llegase hasta aquí.

Finalmente, quiero agradecer de corazón a mi familia y mi círculo de amigos. Aun viviendo en distintas situaciones, han podido confiar en mí.

ÍNDICE GENERAL

I INTRODUCCIÓN

1 CONTEXTO 3
- 1.1 Motivación o justificación del proyecto 3
- 1.2 Objetivos 4
  - 1.2.1 Objetivo general 4
  - 1.2.2 Objetivos específicos 5

2 ESTADO DEL ARTE 7
- 2.1 Dominio del problema presentado 7
  - 2.1.1 Causas del problema presentado 8
  - 2.1.2 Proceso gradual del problema 12
  - 2.1.3 Consecuencias producidas en base al problema 15
- 2.2 Conclusión elaborada en base al problema 16

3 PLANIFICACIÓN 19
- 3.1 Planificación inicial 19
- 3.2 Planificación detallada 20
- 3.3 Presupuesto 22

II PROPUESTA

4 ANÁLISIS 25
- 4.1 Metodología de desarrollo 25
  - 4.1.1 Metodología de trabajo escogida (SCRUM) . . . 26
  - 4.1.2 Adaptación de la metodología de trabajo escogida 28
- 4.2 Análisis del entorno 29
  - 4.2.1 Estudio competitivo 31
  - 4.2.2 Estudio competitivo técnico 32
  - 4.2.3 Conclusiones de los estudios realizados 34
- 4.3 Requisitos del sistema 36
  - 4.3.1 Requisitos Funcionales 36
  - 4.3.2 Requisitos No Funcionales 38
- 4.4 Product Backlog 40
- 4.5 Sprint Backlog 42
  - 4.5.1 Sprint 1 42
  - 4.5.2 Sprint 2 44
  - 4.5.3 Sprint 3 45
  - 4.5.4 Sprint 4 47
  - 4.5.5 Sprint 5 48
  - 4.5.6 Sprint 6 49
  - 4.5.7 Sprint 7 51
  - 4.5.8 Sprint 8 54
  - 4.5.9 Sprint 9 56

5 DISEÑO 57

5.1 Arquitectura planteada 57
5.2 Capa de Base de Datos 58
5.2.1 Estudio técnico 59
5.2.2 Modelado de datos 60
5.3 Capa de Dominio o Negocio 63
5.3.1 Estudio técnico 63
5.3.2 Diagramas de interacción de distintas rutas . . 66
5.4 Capa de Presentación 69
5.4.1 Estudio técnico 69
5.4.2 Transcurso de la Interfaz de Usuario 71
5.4.3 Vista Principal 73
5.4.4 Subvista Popularidad 74
5.4.5 Subvista Palabras más Comunes y Keywords . 75
5.4.6 Subvista Sentimiento General 76
5.4.7 Subvista Noticas Relacionadas 76

III REALIZACIÓN

6 TERMINOLOGÍA DEVOPS 81
6.1 Control de Versiones 82
6.2 Implementación Back-end 83
6.3 Implementación Front-end 85
7 IMPLEMENTACIÓN 87
7.1 Implementación Back-end 87
7.1.1 Implementación Principal 87
7.1.2 Implementación de la Generación del Diccionario 92
7.1.3 Implementación de la Popularidad 96
7.1.4 Implementación de las *Keywords* 97
7.1.5 Implementación de los Sentimientos 104
7.1.6 Implementación de las Noticias 109
7.1.7 Implementación del Servidor 113
7.2 Implementación Front-end 114
7.2.1 Implementación Principal 114
7.2.2 Implementación de la Popularidad 118
7.2.3 Implementación de las *Keywords* 120
7.2.4 Implementación de los Sentimientos 122
7.2.5 Implementación de las Noticias 126
8 DEDUCCIONES Y RESULTADOS 131
8.1 Conclusiones y pruebas de prestaciones 131
8.2 Trabajos futuros 133

BIBLIOGRAFÍA 135

LISTADO DE FIGURAS

Figura 3.1 Planificación inicial del proyecto 19
Figura 3.2 Planificación detallada del proyecto 21
Figura 5.1 Diagrama conceptual de la arquitectura 58
Figura 5.2 Diagrama conceptual con más detalle (1) . . . 59
Figura 5.3 Diagrama de relación de la base de datos . . . 60
Figura 5.4 Diagrama de relación de Tendencia 61
Figura 5.5 Diagrama de relación de Popularidad 61
Figura 5.6 Diagrama de relación de Palabras más Comunes y Keywords 62
Figura 5.7 Diagrama de relación de Sentimiento General 62
Figura 5.8 Diagrama de relación de Noticas Relacionadas 63
Figura 5.9 Documentación generada por FastAPI 65
Figura 5.10 Diagrama conceptual con más detalle (2) . . . 66
Figura 5.11 Diagrama interacción de rutas (1) 66
Figura 5.12 Diagrama interacción de rutas (2) 66
Figura 5.13 Diagrama interacción de rutas (3) 67
Figura 5.14 Diagrama interacción de rutas (4) 67
Figura 5.15 Diagrama interacción de rutas (5) 67
Figura 5.16 Diagrama interacción de rutas (6) 68
Figura 5.17 Diagrama interacción de rutas (7) 68
Figura 5.18 Diagrama interacción de rutas (8) 68
Figura 5.19 Diagrama conceptual con más detalle (3) . . . 69
Figura 5.20 Diagrama de actividad 70
Figura 5.21 Formularios de selección y búsqueda 71
Figura 5.22 Primer boceto de la aplicación 71
Figura 5.23 Segundo boceto más detallado 72
Figura 5.24 Concepto final de la aplicación 72
Figura 5.25 Concepto del error en la visualización de parámetros 72
Figura 5.26 Diagrama de clase de las Vistas 73
Figura 5.27 Estructura y Diseño de la primera subvista . . 74
Figura 5.28 Estructura y Diseño de la segunda subvista . . 75
Figura 5.29 Diseño descartado de la segunda subvista . . . 75
Figura 5.30 Estructura y Diseño de la tercera subvista . . . 76
Figura 5.31 Estructura y Diseño de la última subvista . . . 77
Figura 6.1 Imagen explicativa del concepto DevOps . . . 81
Figura 6.2 Esquema *DevOps* en el *Back-end* 84
Figura 6.3 Esquema *DevOps* en el *Front-end* 86
Figura 7.1 Comparación de la ejecución sin multiprocesamiento 91

Figura 7.2 Comparación de la ejecución con multiprocesamiento 92
Figura 7.3 Normalización utilizada con alfa como 15 . . . 106
Figura 7.4 Ciclo de vida de una aplicación Vue 116
Figura 7.5 Ejemplo de tendencia, vista de popularidad . . 120
Figura 7.6 Ejemplo de tendencia, vista de *keywords* 122
Figura 7.7 Ejemplo de tendencia, vista de sentimientos . 124
Figura 7.8 Ejemplo de tweet (1) 125
Figura 7.9 Ejemplo de tweet (2) 125
Figura 7.10 Ejemplo de tweet (3) 126
Figura 7.11 Ejemplo de tendencia, vista de noticias 128
Figura 7.12 Ejemplo de tendencia, vista de noticias por defecto 129
Figura 8.1 Prueba de prestaciones con Jmeter 132
Figura 8.2 Prueba de prestaciones con *PageSpeed Insights* . 132

LISTADO DE TABLAS

Tabla 3.1 Presupuesto de la mano de obra 22
Tabla 3.2 Presupuesto de servicios adicionales 22
Tabla 4.1 Comparación de metodologías 25
Tabla 4.2 Primer caso de estudio de entorno 29
Tabla 4.3 Segundo caso de estudio de entorno 30
Tabla 4.4 Títulos de Historias de Usuario 40
Tabla 4.5 Subniveles de Títulos de Historias de Usuario 41
Tabla 4.6 Títulos de Sprint 1 42
Tabla 4.7 Descomposición de la Historia de Usuario 1 . 42
Tabla 4.8 Descomposición de la Historia de Usuario 6 . 43
Tabla 4.9 Títulos de Sprint 2 44
Tabla 4.10 Descomposición de la Historia de Usuario 2 . 44
Tabla 4.11 Títulos de Sprint 3 45
Tabla 4.12 Descomposición de la Historia de Usuario 3 . 45
Tabla 4.13 Descomposición de la Historia de Usuario 4 . 46
Tabla 4.14 Títulos de Sprint 4 47
Tabla 4.15 Descomposición de la Historia de Usuario 5 . 47
Tabla 4.16 Títulos de Sprint 5 48
Tabla 4.17 Descomposición de la Historia de Usuario 7 . 48
Tabla 4.18 Títulos de Sprint 6 49
Tabla 4.19 Descomposición de la Historia de Usuario 8 . 49
Tabla 4.20 Descomposición de la Historia de Usuario 8.1 50
Tabla 4.21 Títulos de Sprint 7 51
Tabla 4.22 Descomposición de la Historia de Usuario 9 . 51
Tabla 4.23 Descomposición de la Historia de Usuario 9.1 52
Tabla 4.24 Descomposición de la Historia de Usuario 10 . 53
Tabla 4.25 Títulos de Sprint 8 54
Tabla 4.26 Descomposición de la Historia de Usuario 11 . 54
Tabla 4.27 Descomposición de la Historia de Usuario 11.1 55
Tabla 4.28 Títulos de Sprint 9 56
Tabla 4.29 Descomposición de la Historia de Usuario 12 . 56
Tabla 6.1 Instalación del proyecto 83
Tabla 6.2 Ejecución de *script* 83
Tabla 6.3 Ejecución de servidor 83
Tabla 6.4 Ejecución de *mypy* 83
Tabla 6.5 Ejecución de *flake8* 83
Tabla 6.6 Ejecución de *pytest* 84
Tabla 6.7 Ejecución de *tox* 84
Tabla 6.8 Instalación del proyecto 85
Tabla 6.9 Ejecución del proyecto 85
Tabla 6.10 Construcción del proyecto 85
Tabla 6.11 Ejecución del linter 85

LISTADO DE CÓDIGOS

Código 7.1 Operación principal 88
Código 7.2 Declaraciones de paises 90
Código 7.3 Declaración de *endpoint* 90
Código 7.4 Autentificación de la API de Twitter 90
Código 7.5 Ejecución con multiprocesamiento 91
Código 7.6 Creación de la estructura de tendencias 93
Código 7.7 Creación de la estructura constituyente 94
Código 7.8 Estructuración de la tendencia 95
Código 7.9 Comprobación de los nombres de tendencia . 96
Código 7.10 Estructuración de la popularidad 97
Código 7.11 Actualización de la popularidad 97
Código 7.12 Estructuración de las keywords 97
Código 7.13 Limpieza de las publicaciones 98
Código 7.14 Limpieza de las publicaciones implementada . 99
Código 7.15 Extracción de recuento de palabras y *keywords* 101
Código 7.16 Filtrado de las *keywords* 103
Código 7.17 Estructuración de los sentimientos 104
Código 7.18 Extracción de los sentimientos 107
Código 7.19 Extracción de las coordenadas de los sentimientos 108
Código 7.20 Estructuración de las noticias 109
Código 7.21 Extracción de las noticias 110
Código 7.22 Extracción de la información de las noticias . . 112
Código 7.23 Modelo *Country* y ejemplo de ruta 113
Código 7.24 Secciones de Vue 114
Código 7.25 Funciones de carga de contenido en Vue 117
Código 7.26 Detalles de la implementación del diseño . . . 118
Código 7.27 Diferencias destacables de la implementación de la popularidad 118
Código 7.28 Diferencias destacables de la implementación de las keywords 121
Código 7.29 Diferencias destacables de la implementación de los sentimientos 122
Código 7.30 Diferencias destacables de la implementación de las noticas 127

ACRÓNIMOS

RRSS redes sociales

FOMO *fear of missing out*, «temor a dejar pasar» o «temor a perderse algo»

TT *trending topic*, «tendencia», «tema de tendencia» o «tema del momento»

PaaS *plataforma as a service*, «plataforma como servicio»

YAKE *Yet Another Keyword Extractor*

RAKE *Rapid Automatic Keyword Extraction*

NLTK *Natural Language Toolkit*

VADER *Valence Aware Dictionary and sEntiment Reasoner*

NLP *Natural Language Processing*, «procesamiento de lenguajes naturales»

DOM *Document Object Model*

API *application programming interface*, «interfaz de programación de aplicaciones»

REST *representational state transfer*, «transferencia de estado representacional»

HTTP *Hypertext Transfer Protocol*, «Protocolo de Transferencia de Hipertexto»

JSON *JavaScript Object Notation*, «notación de objeto de JavaScript»

YAML *YAML Ain't Markup Language*, «YAML no es un lenguaje de marcado»

CRUD *Create, Read, Update and Delete*, «Crear, Leer, Actualizar y Borrar»

ASGI *Asynchronous Server Gateway Interface*

WSGI *Web Server Gateway Interface*

RSS *Really Simple Syndication*, «sindicación realmente simple»

Parte I

INTRODUCCIÓN

1 CONTEXTO

1.1 MOTIVACIÓN O JUSTIFICACIÓN DEL PROYECTO

El año 1989 fue clave en distintos aspectos, conocido por la disolución del Telón de Acero en Europa y la caída del Muro de Berlín, aunque sobre todo destacó por la apertura del Internet al público. Una herramienta, en un principio, al alcance de pocos que inicio su meteórico ascenso a ser uno de los mayores inventos. Hoy en día, capaz de conectar al mundo como muy pocos y que situaciones como la reciente pandemia COVID-19 no han hecho más que impulsar el intercambio digital de información entre personas.

Para podernos situarnos con más claridad, el estudio anual de IAB Spain nos indica que en el año 2022 el 93 % de las personas que residen en España son usuarios del Internet. Además de ese porcentaje, el 85 % (28,3 millones de españoles) utilizan las redes sociales. [1]

Estos datos nos dejan en evidencia el hecho de que el uso de estas herramientas, tales como el Internet y las redes sociales, van en aumento. Aunque no nos debería de asombrar, ya que la generalización y el uso del Internet y las redes sociales viene de mucho antes. En el año 2012 se realizó un estudio en el que se mostraba que las redes sociales pueden llegar a crear una dependencia aún más grande que el tabaco. [2]

Incluso muchos programadores de redes sociales han admitido que el diseño de tales aplicaciones es adictivo de manera deliberada. Un ejemplo de esto puede ser la creación del scroll infinito, donde se brinda gran cantidad de información de una manera continua. [3]

La solución más sencilla, pero drástica, puede llegar a ser abandonar las redes sociales, aunque nos surge el problema de no estar tan *actualizados* como otras personas a nuestro alrededor. Otra solución es usar en una menor medida las redes sociales, aunque esto nos lleva de nuevo al problema de cuándo parar si la aplicación que usamos tiene scroll o contenido infinito.

Ante tal necesidad de querer estar actualizados y a la vez querer saber cuándo parar de consumir contenido, la solución que se propone en este texto es un consumo finito de contenido diario y lo más objetivo posible. Una aplicación web que aporte la información necesaria de las tendencias más destacables del día.

1.2 OBJETIVOS

1.2.1 *Objetivo general*

El interés principal de este proyecto reside en la construcción de una aplicación que pueda explicar lo relevante en un momento dado, actualizando la información sin intervención humana y de manera completamente automática. A la hora de cumplir esta meta, se pretende llevar a cabo un estudio completo y óptimo sobre la planificación, el diseño, la estructura y arquitectura de la propia aplicación.

El propósito de la aplicación es que el usuario final pueda consumir contenido relevante de una manera eficaz y eficiente. Para poder proveer el contenido más relevante, podemos partir de la idea de que lo más destacable en internet es lo que más se repite o se comparte con otras personas. El concepto de lo más destacable en la red, suele recibir el nombre de *trending topic*, «tendencia», «tema de tendencia» o «tema del momento» (TT).

Por tanto, las tendencias o los TT nos permiten llegar a conocer y saber en un momento dado lo que genera mayor conversación y debate entre la sociedad, pudiendo conocer los temas más actuales, ideas generales o multitud de voces que se hayan unido por una causa.

En consecuencia, se ha decido como óptimo que la información que se le brinda al usuario se basará en las tendencias, pudiendo dividirlas por el país y la fecha seleccionada por él. Al querer que el contenido no sea abrumante, las tendencias se presentarán en un *top* o una clasificación de diez tendencias más populares ocurridas a lo largo del día. A partir de cada tendencia se deducirán datos, información útil, estadísticas, noticias y enlaces de interés que le puedan ser útiles al usuario.

La visualización de contenido que pretende tener esta página web, es un contenido presentado en forma de cartas o módulos, con un carácter informativo, finito y objetivo. Los datos deducidos o extraídos de la tendencia serán datos relevantes como la popularidad o el volumen de publicaciones que está teniendo la tendencia. La información útil que se pretende mostrar serán las palabras claves o más usadas relacionadas a la tendencia. También se pretende mostrar la opinión general de manera objetiva, esto se puede conseguir analizando los sentimientos de las publicaciones mediante analizadores léxicos. Por último, las noticias extraídas también albergarán contenido actual y relevante sobre dichas tendencias.

Para cumplir con dichas implementaciones, se deberá hacer un cuidadoso estudio sobre diferentes librerías, bibliotecas externas y algoritmos que se vayan a emplear en el proyecto. En la sección de las pa-

labras claves se enfocará sobre el algoritmo de extracción *Yet Another Keyword Extractor* (YAKE) y en el análisis de sentimientos se centrará en la librería *Valence Aware Dictionary and sEntiment Reasoner* (VADER).

Por último, conviene mencionar que toda esta información recolectada deberá albergar un carácter sencillo y deberá ser fácil de consumir, alejándose de la posibilidad de ser abrumante o atosigadora, como la de las redes sociales.

1.2.2 *Objetivos específicos*

- Se realizará un estudio sobre el diseño, la estructura y arquitectura más óptima al proyecto. Al igual que las distintas herramientas usadas en este.
- Se marcarán las pautas de unas buenas prácticas a la hora de realizar el desarrollo y las operaciones del proyecto.
- Se estudiarán las ventajas de recolectar las tendencias provenientes de la API de Twitter y se compararán con otros métodos destacables.
- La aplicación web debe de ser intuitiva y fácil de usar, el diseño debe ajustarse para el formato móvil.
- El contenido que se le brinda al usuario debe de ser corto y finito, pero a la vez suficiente con el fin de informarse de manera adecuada.
- Cada contenido proporcionado para el usuario se brindará en un diseño de carta y ocupará el mayor espacio de pantalla posible. De esta manera se aprovecharán al máximo la información disponible, siendo esta una noticia, un dato de una estadística o un enlace de interés.
- Cada información recolectada dispondrá de un elemento gráfico y textual. La popularidad, las palabras más relevantes o los sentimientos generales se pueden expresar mediante una gráfica y también explicarse textualmente. Al igual que las noticias, cada noticia puede tener una imagen y una descripción.
- El usuario podrá navegar por dichas tendencias y estar *actualizado* en cada momento del día, además podrá ver tendencias de otros días o países. Siempre respetando que el contenido sea finito, es decir no atosigante, y relevante o actual.

2 ESTADO DEL ARTE

2.1 DOMINIO DEL PROBLEMA PRESENTADO

Como se había mencionado en las secciones anteriores, la motivación detrás de este proyecto es la creación de una herramienta que permita al usuario un uso más moderado y responsable de la información dada por las redes sociales (RRSS). Para entender mejor el contexto, la definición que nos proporciona el autor Orihuela de las RRSS es:

> *Son servicios basados en la web que permiten a sus usuarios relacionarse, compartir información, coordinar acciones y en general, mantenerse en contacto. [4]*

Dando a entender que las RRSS son percibidas como un gran entorno social, en el cual se comparte información de una manera constante.

El hecho de que estos entornos sociales tengan un fácil acceso, fomentando la posibilidad de adicción y esto siendo de una manera deliberada termina siendo el problema real del consumo de información en las RRSS. Se han mencionado artículos y estudios anteriormente donde se explicaba que la adicción de las RRSS viene directamente de la parte de la programación [3] y que las RRSS pueden llegar a crear una dependencia más grande aún que el tabaco. [2]

Todo esto acaba reflejado directamente en los usuarios, los cuales son realmente el lucro de todas estas grandes compañías. El objetivo real de todas estas empresas es conseguir el mayor número de usuarios y el mayor uso de su aplicación posible. De este modo a ellos, a términos generales, les conviene conseguir algoritmos o técnicas que favorezcan la consumición de dicho producto.

La idea general, la cual nos adentraremos en más detalle a continuación, viene a ser la abrumación de la información donde el usuario ni si quiera está en control del consumo de dicha información y tampoco del tiempo empleado dentro de la aplicación.

Todo ello tendrá consecuencias reflejadas directamente y exclusivamente en los usuarios, lo cual se verá en el siguiente apartado 2.1.3.

Toda esta información puede llegar a ser obvia si se expone superficialmente. Por lo que es necesario entrar en mucho mayor detalle a la hora de exponer el problema.

2.1.1 *Causas del problema presentado*

El *smartphone* o teléfono móvil es solamente una mera herramienta que nos permite conectarnos a Internet y acceder a diferentes servicios virtuales o aplicaciones web. El sentimiento de la adicción no se origina en la propia herramienta, sino en los distintos servicios ofrecidos por ella.

El modelo de negocio de la mayoría de las aplicaciones web consiste en el intercambio de datos personales en cambio al uso del propio producto. De esta manera pueden posteriormente tener mejores resultados de ventas en base a anuncios personales. Se estima que alrededor de un 91 % de los compradores son más receptivos a la compra de un producto cuando una marca personaliza su comunicación con ellos, es decir, anuncios personalizados en base a sus perfiles. [5]

Por lo que no es de extrañar que los propietarios de las aplicaciones web tengan la intención de prologar el uso de la misma, ya que esto les proporcionará mayor estudio de los perfiles de sus usuarios, mayor posibilidad de compra en sus anuncios y obviamente mayor rendimiento o beneficio.

De esta manera, es bastante sencillo encontrar diseños, patrones o incluso algoritmos que tengan la intención de la prolongación de uso de los usuarios en la aplicación web. Existen distintos elementos utilizados para prolongar el tiempo de uso de las aplicaciones de redes sociales, algunos de ellos los detallaremos a continuación. [6]

1. *Scroll* o *Streaming* infinito:

   Gran cantidad de aplicaciones tienen un diseño inmersivo, en consecuencia, llegan a un producir lo que en psicología se denomina *flujo* ó *la zona* [1]. Dicho flujo mental, es una cualidad que normalmente es positiva, ya que puede llegar a incrementar en una gran medida la productividad. En otros casos puede llegar a ser altamente peligrosa, como es el caso de los dispositivos móviles; la actividad puede llegar a ser altamente gratificante y tener una retroalimentación directa con el usuario, dando lugar a la perdida de la noción del tiempo o su propia distorsión.

   Un claro ejemplo de esto es la prohibición del uso del teléfono móvil al circular con vehículo por las vías públicas, esto es debido del alto peligro que puede llegar a haber al prestar atención o variar el flujo mental por la mayor inmersión que ejerce el teléfono móvil y dichas aplicaciones instaladas en él.

1 El *flujo* ó *la zona* es un estado mental que se consigue cuando una persona está completamente inmersa en la actividad que esté ejecutando, es decir, es un estado de concentración de energía o de motivación.

Este estado mental es provocado directamente de las aplicaciones instaladas en los teléfonos y esto de manera deliberada por muchos programadores. El *Scroll* o *Streaming* infinito es solamente un diseño que fue propuesto por Raskin en 2006. Dicho diseño es sumamente adictivo, ya que no da tiempo a que lleguen todos los impulsos necesarios del cerebro mientras que el usuario sigue deslizando contenido tras contenido. El usuario además encuentra algo gratificante con cada contenido nuevo aportado, como puede ser un vídeo o foto graciosa. Este efecto llega a ser muy parecido al efecto que producen las máquinas tragaperras, la condición de inmersión al usuario es prácticamente igual. Otro diseño parecido puede ser el *Auto-play* [2], donde la lógica es la misma.

El creador de dicha herramienta, Raskin, expuso en una entrevista su arrepentimiento de haber creado algo tan adictivo [3]. Su intención era más bien intentar innovar las interfaces que volver a las personas tan adictivas. [3]

2. Efecto dotación o efecto de mera exposición:

   El efecto dotación consiste en que las personas atribuyen más valor a un objeto por el simple hecho de poseer dicho objeto. Esto es, que si una persona llega a poseerlo, ya sea tangible o intangible, luego le es más difícil desprenderse de ello, aunque le ofrezcan el mismo o incluso mayor valor que posee dicho objeto.

   El efecto de mera exposición también es un efecto psicológico, este se basa en que la reiteración de exposiciones incide o estimula en nuestro agrado o desagrado. En otras palabras, las personas llegan a familiarizarse con distintos estímulos por el simple hecho de repetirlos en diferentes ocasiones.

   Estos dos efectos influyen en nuestro contexto, de tal manera que los usuarios de distintas aplicaciones al gastar más tiempo, valor o dinero en ellas les resulte más complicado posteriormente abandonar o siquiera desinstalar la propia aplicación. Tanto por el simple hecho de llegar a tener algún tipo de valor en la propia aplicación, como por ejemplo seguidores o *likes*, o directamente por estar familiarizados con la aplicación por haberla usado día a día y verla siempre en la pantalla de inicio.

3. Presión social:

2 *Auto-play* es una herramienta donde se proporciona contenido infinito si no se llega a parar por el propio usuario. Un ejemplo de ello, aplicaciones como YouTube o Spotify, donde se requiere darle al botón *pause* para frenar dicho contenido.

3 Raskin además de admitir que nunca pudo predecir lo adictivo que su invento podría ser, también comentó que él mismo está usando un filtro monocromático en su dispositivo móvil para minimizar lo adictivo que puede llegar a ser tener tantos estímulos en la pantalla.

WhatsApp actualmente es la aplicación de mensajería más usada en todo el mundo, ahora mismo, posee más de 2.000 millones de usuarios. [7] Al igual que otras redes sociales dominantes en el mercado, como Instagram o Twitter, posee diseños construidos de tal manera que llegan a inferir desde muchas personas a un individuo en concreto, ejerciendo algo parecido a la presión social.

Un claro ejemplo de esto puede ser la función el doble *tick* azul. Consiste, en el caso de la mensajería instantánea, notificar al usuario de que el mensaje ha llegado hacía la otra persona y además fue leído por ella. Esta función compromete al usuario de manera que tenga responder rápidamente o incluso por obligación si el mensaje fue leído o entregado. Dicha función aparece activada por defecto, la cual muchos usuarios ni son conscientes de que pueden llegar a cambiar. Según un reporte, únicamente un 5 % de los usuarios de Mirosoft Word llegó a cambiar algún ajuste que venía por defecto. [8]

De manera análoga, se puede establecer un vínculo de las características mencionadas anteriormente con la presión social, como es el caso del doble *tick* azul, y el síndrome o fenómeno *fear of missing out*, «temor a dejar pasar» o «temor a perderse algo» (FOMO). Como su nombre expresa, es el miedo a perderse algo o ser excluido, dando lugar a la necesidad de estar permanentemente conectados y *actualizados*. Las consecuencias pueden llegar a ser muy graves, ya que cuanto mayor es el uso del dispositivo móvil mayor es el grado de FOMO; el temor de perderse algo y de esta manera no satisfacer nuestras necesidades psicológicas retroalimenta el uso problemático y abusivo de las aplicaciones. [9]

4. Contenido personalizado con la intención de causar agrado:

   Al entrar a cualquier aplicación tipo Facebook, Instagram o Twitter lo primero que se encuentra el usuario es con una abrumación de contenido para que este no tenga momento de aburrirse. Este contenido suele recibir el nombre de *Feed*, *NewsFeed* o *TimeLine*. Al presentar el *Feed* a cualquier usuario, este viene personalizado de antes con los gustos del usuario, para intentar mantenerlo el mayor tiempo posible.

   Los algoritmos que hay por detrás del *Feed* de cada usuario son enormemente complejos, ya que no solo se encargan de calcular el contenido que le agrada al usuario por medio de *likes* o *shares* del contenido, sino también del contenido que el usuario gasta en mirar cada publicación.

   Anteriormente el *Feed* de las aplicaciones funcionaba de manera cronológica, pero actualmente se basan en la retención de la

atención del usuario. Intentado mostrar al usuario contenido que sea afín a sus intereses. [10]

Aunque no siempre tienen la intención de causar agrado, en algunos reportes se declaró justo lo contrario. Gracias a Frances Haugen, en 2021 se filtró información sensible sobre Facebook donde entre mucha información se exponía que esta red social también llega a mostrar contenido que el usuario puede odiar, solamente para tener más retención por parte del usuario. [11] Esto además se traduce a que el algoritmo muestre contenido cada vez más violento para que el usuario tienda siempre a interaccionar con él.

5. Comparación y recompensa social:

   El *Like* fue desarrollado por Justin Rosenstein, esta función probablemente es el mecanismo más emblemático de recompensa social que hay hasta el momento. Es recompensa social, ya que hay retroalimentación social positiva al interaccionar con los *likes* de una publicación. Esta retroalimentación social está demostrada científicamente, en un experimento neurocientífico se presentaron imágenes con muchos *likes* o *me gustas*, estas imágenes provocan una actividad más fuerte en el cuerpo estriado del cerebro (parte del cerebro relacionada con los mecanismos de recompensa). Además, también se detectó que los volúmenes bajos de materia gris se asociaron con el uso prolongado de este tipo de aplicaciones, lo que daba lugar a mayores tendencias adictivas. [12, 13]

   Leah Pearlman, coinventora del botón *Me gusta* de Facebook, dijo que se sintió adicta a Facebook porque había empezado a basar su autoestima en la cantidad de *likes* que tenía. [3] Aunque este normalizado, sigue siendo un problema el que una persona sea capaz de compararse numéricamente por el hecho de cómo es percibida por su red social. Aunque hablemos de las consecuencias más adelante (2.1.3), hay que mencionar el hecho de compararse de esta manera puede dar lugar a una baja autoestima y otros muchos problemas.

6. Sistema de notificaciones:

   La funcionalidad de un simple sistema de recordatorios, nos presenta notificaciones en determinados momentos a lo largo del tiempo, ya sea a través del sonido o recuadros visuales. El objetivo de estas notificaciones puede ser una utilidad de la aplicación, como un mensaje o solo para tener presente el hecho de que podemos volver a usar la aplicación.

   De esta manera nos tienen pendientes mediante un sencillo sistema, el cual puede llegar a ser muy útil en determinados momentos, como un mensaje del banco. Sin embargo, el diseño de

dicho sistema puede llegar a ser perjudicial para el usuario. Un claro ejemplo de esto es Instagram, la cual agrupa notificaciones para poder otorgar al usuario mayor cantidad de recompensa o agrado, es decir una cantidad determinada de *likes*, esto se traduciría al control de diferentes neurotransmisores como la dopamina en el usuario, aunque indagaremos más a fondo en la sección 2.1.2.

Todos estos numerosos problemas están estrechamente relacionados con las aplicaciones que existen a día de hoy, ya sea solamente uno o numerosos de ellos en la misma aplicación. Estos además pueden llegar a aumentar su efecto de distintas maneras si se agrupan entre si.

El tipo de aplicaciones que se sugiere en el párrafo anterior son las aplicaciones que anteponen la posibilidad de monetización a la eticidad respecto al usuario. Este tipo de aplicaciones no priorizan agilizar nuestro modo de vida, como es el caso de una aplicación bancaria, sino que intentan generar estímulos presentes entre emociones negativas como la soledad o el aburrimiento.

Sandy Parakilas, antiguo director de operaciones de Facebook llego a comentar lo complicado que le era abandonar la plataforma, comparándola con cigarrillos o máquinas de casino. Aunque también comentó cuestiones de mayor peso, explicando que el uso de estas aplicaciones construía un hábito en nuestras vidas: [3]

> *There was definitely an awareness of the fact that the product was habit-forming and addictive. [...] You have a business model designed to engage you and get you to basically suck as much time out of your life as possible and then selling that attention to advertisers.*

Lo cual viene a ser un buen resumen de lo que se ha tratado de explicar en esta sección. En castellano tendría este significado:

> *Definitivamente hubo un conocimiento del hecho de que el producto era capaz de crear hábitos y era adictivo. [...] Tienes un modelo de negocio diseñado para engancharte y hacer que básicamente absorba la mayor cantidad de tiempo posible de tu vida y luego venda esa atención a los anunciantes.*

2.1.2 *Proceso gradual del problema*

2.1.2.1 *Inflexión en el individuo*

El uso de las redes sociales tiene numerosos efectos sobre el cerebro, estos pueden llegar a influir tanto de manera positiva como negativa.

Al navegar por las RRSS el cerebro se adapta llegando a crear nuevas redes neuronales. [14] Además, también pueden llegar a producir diferentes cambios en la manera que funcionan distintos neurotransmisores como la oxitocina, la adrenalina, la dopamina, la serotonina, la testosterona y el cortisol.

Todos estos neurotransmisores tienen un determinado papel a la hora de estar usando las distintas aplicaciones en los teléfonos móviles. La oxitocina está relacionada con la influencia de la familia y la pareja, mientras que la adrenalina se vincula con la agresividad. El aumento de serotonina se llega a traducir a comportamientos sociales en ámbitos más introvertidos. Por otra parte, altos niveles de testosterona y el cortisol tienen que ver con una alta fidelidad, aunque menor número de amistades virtuales.

Aunque la que podría llegar a ser más problemática para el usuario es la dopamina [4], ya que está mucho más relacionada con las redes sociales. Anteriormente mencionamos que las RRSS proporcionan al usuario diferentes tipos de interacciones y recompensas sociales, de esta manera generando al usuario emotividad y afectividad.

Existen cuatro vías que transmiten dopamina en nuestro cerebro llamadas vías dopaminérgicas, ellas son la mesocortical, la nigroestriada, la mesolímbica y la tuberoinfundibular. Las tres primeras de ellas están relacionadas con las recompensas que se mencionaba en el texto anterior. Al tener un estímulo satisfactorio o agradable por estas vías, ciertas acciones se disparan y nuestro cerebro refuerza dicho comportamiento. [16]

Aunque cabe mencionar que el abuso de estas vías por medio de sustancias o hábitos da lugar a la creación de la adicción. Además, la falta de dicho estímulo puede provocar un rechazo por parte del cerebro creándonos malestar o ansiedad, esto es debido a que estas vías se hayan podido *deformar* por un mal hábito o aprendizaje, como por ejemplo la transmisión de la posible tranquilidad que nos puede aportar estar continuamente consultando el teléfono.

Los seres humanos somos seres sociales y es normal que dependamos de interacciones continuamente, dando lugar a una posible adicción de conducta. Esto es, debido a que no solo se puede ser adicto a una sustancia, sino a diferentes conductas o hábitos, un ejemplo de ello puede ser una máquina de un casino. El efecto que genera una máquina de casino y una red social en el cerebro puede llegar a ser parecida, además las recompensas que nos sirven este tipo de

4 La dopamina regula la emotividad y la afectividad, así como en la comunicación neuroendocrina. Algunas alteraciones en la transmisión dopaminérgica han sido relacionadas con la adicción a drogas (anfetaminas y cocaína, por ejemplo). [15]

adicciones son mucho más satisfactorias y requieren menos esfuerzo que cualquier actividad más sana como el ejercicio o la lectura. [17]

Según un estudio del Universidad de Philipps, las recompensas sociales que disparan los distintos mecanismos de las vías dopaminérgicas son sencillas interacciones que tenemos con los seres humanos, ya sea una caricia, sonrisa o palabras afectivas. Todas estas interacciones son suficientes para influir como una recompensa social y crear dicho estímulo. Los investigadores evolutivos explican este fenómeno como un mecanismo social del ser humano necesario y positivo. [18]

Entrando más en detalle en las adicciones de conducta y las notificaciones de las aplicaciones. Podemos recurrir al clásico caso de la teoría de Pavlov del siglo XX, donde se presentaba un estímulo que no tenía nada que ver y terminaba siendo reflejo condicional (Ley del reflejo condicional o condicionamiento clásico). En la teoría de Pavlov se tocaba una campana al servir comida a unos perros, al cabo de un tiempo los perros al escuchar el estímulo de la campana llegaban a salivar sin que se les sirviera la comida. [19]

En cierto sentido hay parecidos entre el condicionamiento clásico de Pavlov y las notificaciones de los dispositivos móviles hoy en día. El estímulo de la notificación nos crea la conducta de estar pendientes de los teléfonos y estar mirándolos continuamente, llegando a coger el móvil decenas o centenares de veces al día. Además, este estímulo puede estar asociado a un sentimiento positivo, como puede ser el mensaje de un amigo. Un sencillo ejemplo de ello es acudir o prestar atención a un móvil ajeno por tener un sonido de notificaciones parecido al propio.

Según la investigadora de diseño de la red social Twitter, Ximena Vengoechea, afirma que el *enganche* de la notificación compromete a dos mecanismos, uno interno y otro externo. [16] Se refiere al interno como la emoción, mientras que el externo es la acción que pretende que hagamos. La sincronización de ambos es la combinación que despierta la motivación del usuario, creando la necesidad de la recompensa. Además, como ya sabíamos, las RRSS aprovechan la implicación de estímulos visuales o distintos diseños derivando en un mecanismo de muchos niveles. Esta mezcla nos crea distintas costumbres o hábitos que son, sin darnos cuenta, posiblemente adictivos.

Actualmente este tipo de temas son bastante comentados, aunque tristemente este uso problemático de las redes sociales fue ignorado debido a la gravedad de la situación surgida por el COVID-19. La mayoría de las actividades que realizábamos día a día se volvieron virtuales, dando lugar a un incremento de dependencia y a una autentica necesidad de comunicación por medios virtuales.

2.1.2.2 *Inflexión en la sociedad*

Otra consecuencia gradual en base al problema es la cantidad de información que es bombardeada al usuario día tras día, dando lugar a una gran cantidad de contenido que realmente es difícil de manejar. En cualquier red social, se masifica y se comparte la información o una noticia mediante opiniones. Creando así tendencias, pero quitando de esta manera la objetividad de la información o directamente dando lugar a su mal interpretación.

La noticia o tendencia por lo general suele tergiversarse debido a la cantidad ingente de opiniones e interacciones a las que se ve el sujeto. Dando lugar, a que el usuario no sea capaz de sacar en claro el estado de la noticia o tendencia, tanto si al final es positiva o negativa, en base a unas pocas compartidas. Además, es de suma importancia el recibo que tienen estas opiniones mediante sus interacciones o la cantidad de opiniones parecidas que se van publicando a lo largo del tiempo de vida de la tendencia. Por lo que la medida más óptima de informarse de dicho contenido es a través de una recopilación objetiva, con la meta de alejarse de las opiniones y abarcar una información final respecto al tema que se intenta compartir.

2.1.3 *Consecuencias producidas en base al problema*

Los problemas existentes acaban afectando a todos los usuarios, independientemente de su edad. Por normativa, legalmente las redes sociales exigen que sus usuarios tengan al menos 13 años a la hora de crearse una cuenta. Aunque es cierto que esto no se cumpla del todo y haya usuarios con menos de 13 años exponiéndose a diferentes peligros de las RRSS.

Estudios realizados en 2021 demuestran que los niños a partir de 10 años tienen tendencias de uso de redes sociales. Estas exposiciones a tan temprana edad pueden terminar siendo un peligro y llegar a ser una adicción. [20]

Estos problemas también pueden surgir en personas de más avanzada edad. Ya hemos visto que el deseo de acceder a redes sociales como Twitter o Instagram se encuentra como el deseo más complicado de resistir y más sencillo de complacer. Sustancias adictivas como el tabaco o el alcohol generan un deseo mucho más débil, esto además es muy fácil de demostrar sabiendo que un usuario promedio de estado unidos mira el teléfono unas 344 veces al día, lo que se traduce como usarlo cada 4 minutos al día. [21]

Otro estudio, igual de reciente, mostraba que el 40 % de la población española llegaba a tener problemas relacionados con la adicción al Internet. Para hacernos una idea, en 2008 dedicábamos una media

de 18 minutos al móvil, en cambio, ya a partir de 2015 gastábamos alrededor de 3 horas diarias. Esto llegaba a afectar a los adolescentes de tal manera, que el 80 % de ellos tenían la necesidad de mirar el móvil, al menos, una vez cada hora a lo largo del día. [22]

Las consecuencias de la adicción a las redes sociales pueden llegar a ser la ansiedad, dependencia emocional, baja autoestima o problemas de sociabilización. Estudios realizados en 2021, demostraron la correlación existente entre la dependencia de las RRSS, la autoestima, ansiedad y la obsesión. Para ello participaron 100 alumnos, estudiantes universitarios. Este estudio corroboró información que existía previamente, resultados que señalaban que las personas que tenían adicción al uso del Internet presentaban una menor autoestima y relaciones personales más inestables. [23]

Esto también fue expuesto en las redes sociales, la autoestima influía en el uso de RRSS de manera reciproca, es decir, tener una alta autoestima también significaba un menor uso de RRSS. Además, los estudiantes que presentaban una mayor autoestima exhibían niveles más bajos de obsesión, mayor control personal y empleo de dichas RRSS. En cambio, cuando el uso de las RRSS era mayor, los niveles de autoestima eran más bajos, incrementando de esta manera los niveles de ansiedad y depresión.

Sin embargo, el estudio no especifico la existencia de una correlación entre la adicción y la autoestima. Aun teniendo datos como el 66.7 % de los adolescentes fueron expuestos como adictos al Internet (no a las RRSS [5]) y 62.7 % manifestaron problemas de autoestima. Aunque si se demostró la existencia de correlación entre la adicción y la ansiedad. [23]

2.2 CONCLUSIÓN ELABORADA EN BASE AL PROBLEMA

En el texto anterior se quedó demostrado el hecho de que las redes sociales pueden llegar a ser adictivas, como son capaces de influir en un individuo y sus consecuencias. El problema real de esto recae en el propio diseño de las aplicaciones, si la información fuera expuesta de una manera clara y sin patrones con la intención de retener al usuario el mayor tiempo posible; el usuario podría usar dichas aplicaciones sin tan grave peligro.

La solución que se pretende proponer es una aplicación sin los diseños que se han comentado. En primer lugar, siendo una aplicación con un contenido finito y este siendo el más importante en el que se centre el usuario. La aplicación web se centraría en las tendencias

5 En dicho estudio solamente se midió, mediante cuestionarios, la adicción del Internet en general y no específicamente las redes sociales.

más importantes de cada país del usuario, de esta manera mostrando los eventos más importantes que hayan ocurrido al final del día.

Siendo una aplicación web, se elimina la necesidad de tenerla instalada y sus posibles notificaciones. Además, se centraría en ser objetiva, quitando el efecto FOMO y la personalización de contenido. Al quitar la parte subjetiva y distintas opiniones, es decir, eliminando la parte social de las RRSS y solo quedándonos con la información; eliminamos la posible comparación entre usuarios existente.

De esta manera, hay una posibilidad de seguir *actualizados* pudiendo ver el contenido importante que se ha proporcionado durante el día, pero quitando el peligro de volvernos adictos.

3 PLANIFICACIÓN

3.1 PLANIFICACIÓN INICIAL

La planificación mostrada a continuación fue realizada al inicio del proyecto, siendo ésta de muy alto nivel. De momento no vamos a indagar en detalles específicos sobre las metodologías usadas o la planificación real, esto en cambio, se detallará en la sección posterior.

Un trabajo de fin de grado está compuesto por 12 créditos en el Título de Grado en Ingeniería Informática de la Universidad de Granada. Además, por normativa de la universidad, a cada crédito ECTS se le establece un número de 25 horas de trabajo. En total, 300 horas de trabajo. Esto se traduce a casi dos meses de trabajo si contamos con una jornada laboral completa y descanso los fines de semana.

Aunque, al fin y al cabo, sigue siendo una estimación muy ligera a lo que realmente terminará siendo. Por lo que la estimación que usaré a continuación será en base a las fechas del inicio y el fin de la elaboración del proyecto, en mi caso será desde inicios de Julio hasta finales de Octubre:

Figura 3.1: Visión general sobre la estructura de la elaboración del proyecto.

- Inicio *(01-07-22 ∼ 01-08-22)*: etapa de planificación de la memoria y el estado del arte. Además, se intentará tener un primer prototipo de la aplicación.
- Diseño *(01-08-22 ∼ 01-09-22)*: inicios de la codificación y la elaboración de la parte correspondiente de la memoria.
- Lanzamiento *(01-09-22 ∼ 01-10-22)*: etapa donde se realizarán pruebas de usuario y finalizará la codificación.
- Entrega final *(01-10-22 ∼ 31-10-22)*: elaboración de conclusiones respecto al cumplimiento del software y sus posibles mejoras.

3.2 PLANIFICACIÓN DETALLADA

Al planificar el proyecto de la manera que está representado en la figura 3.1, he optado por una metodología ágil para confrontar el proyecto.

Esto es debido a que necesitamos un marco de trabajo que sea adaptativo y flexible a la hora de realizar distintas pruebas y su correspondiente arreglo a la hora de encontrarnos errores o cambios exigidos por el cliente.

Todas estas ventajas son las que me encaminado a elegir esta metodología, la cual parece tener las mejores prácticas para este proyecto en concreto.

Hoy en día, el marco de trabajo más ampliamente conocido y a la vez el más ventajoso sea posiblemente SCRUM. Se ha elegido este marco de trabajo para tener el mejor resultado posible a la hora de abarcar el proyecto. Sus principales ventajas son la flexibilidad y la productividad, pero abarcaremos más en detalle sus propiedades en la sección 4.1.

En la siguiente página se podrá ver la planificación en mucho mayor detalle. En la tabla (figura 3.2) se detalla toda la perspectiva del trabajo, desde las partes más fundamentales de la documentación hasta el diseño de la propia aplicación.

Figura 3.2: Visión más detallada sobre la estructura de la elaboración del proyecto.

3.3 PRESUPUESTO

Las horas de trabajo las tenemos especificadas desde un principio, es decir, el trabajo de fin de grado por normativa tiene 300 horas. Gracias a las páginas web como Talent podemos saber que el salario medio de un desarrollador web está en torno de los 11,79 €. [24] En cuanto a las horas de trabajo y sus diferentes secciones quedaría de la siguiente manera.

	Horas	Porcentaje	Coste
Introducción	16,9014	5,6338 %	199,27 €
Estado del arte	8,4507	2,8169 %	99,64 €
Formación	21,1267	7,0422 %	249,08 €
Entorno de trabajo	147,8873	49,2957 %	1743,59 €
Diseño	105,6338	35,2112 %	1245,42 €
Total	$\approx$ 300	$\approx$ 100 %	3536,99 €

Tabla 3.1: Presupuesto de la mano de obra, junto con las horas trabajadas.

En base a ello, podemos calcular que unas 300 horas de trabajo equivaldrían a 3537 € y la elaboración de la documentación o el manual equivaldrían a 300 €. En total 3837 € solamente por la mano de obra.

Buscaré además el uso de herramientas gratuitas de diferentes servicios para el mantenimiento del servicio web. En los servicios como Heroku varían los precios entre 100 € y 200 € al mes, pero también poseen herramientas gratuitas y muy limitadas, lo que nos permitirá un gran ahorro al inicio del proyecto. Igualmente, para el proyecto cubriré necesidades básicas, como un área de trabajo y un ordenador capaz de manejar dicho proyecto.

	Cantidad	Coste
Espacio de trabajo	4 meses	$\approx$ 400 €
Ordenador	1	$\approx$ 900 €
Total		$\approx$ 1300 €

Tabla 3.2: Presupuesto de servicios adicionales, junto con las diferentes cantidades.

De modo, que el coste total del proyecto sería la suma de ambas cantidades. En este caso, quedaría en 5137 €.

Parte II

PROPUESTA

4 ANÁLISIS

4.1 METODOLOGÍA DE DESARROLLO

El proceso de desarrollo de un software puede llegar a ser una tarea muy ardua, durante mucho tiempo esta labor se ha llevado a cabo sin ninguna metodología definida. Para situar el contexto, una metodología se define como una colección de procedimientos, técnicas, herramientas y documentos que llegan a ayudar a los desarrolladores a la hora de implementar software. [25]

Durante las últimas décadas, se han establecido dos grandes corrientes diferenciales que representan estas metodologías. Por un lado, las metodologías tradicionales se centran en el proceso y controlan estrictamente las actividades que lo acompañan. Las metodologías ágiles, por otro lado, se enfocan en el elemento humano, centrándose en la colaboración y participación del cliente en el proceso a través de iteraciones muy cortas. [25]

Metodologías Ágiles	Metodologías Tradicionales
Basadas en heurísticas provenientes de prácticas de producción de código	Basadas en normas provenientes de estándares seguidos por el entorno de desarrollo
Especialmente preparados para cambios durante el proyecto	Cierta resistencia a los cambios
Impuestas internamente (por el equipo)	Impuestas externamente
Proceso menos controlado, con pocos principios	Proceso mucho más controlado, con numerosas políticas/normas
No existe contrato tradicional o al menos es bastante flexible	Existe un contrato prefijado
El cliente es parte del equipo de desarrollo	El cliente interactúa con el equipo de desarrollo mediante reuniones
Grupos pequeños (menos de 10 integrantes) y trabajando en el mismo sitio	Grupos grandes y posiblemente distribuidos
Pocos artefactos y pocos roles	Más artefactos y más roles
Menos énfasis en la arquitectura del software	La arquitectura del software es esencial y se expresa mediante modelos

Tabla 4.1: Comparación de metodologías. [26]

La parte más importante de las fases iniciales es poder determinar qué metodología es mejor para nuestro proyecto con el fin de lograr los mejores resultados de manera oportuna.

El objetivo general de implementar una metodología es llegar a construir un producto de alta calidad. Esta elección implica un conjunto de principios básicos que deben ser seguidos y respetados. Estos incluyen actividades claras para comprender el problema y comunicarse con los clientes, un método definido para representar el diseño, las mejores prácticas para implementar la solución y estrategias sólidas. [27]

En base a las características mencionadas y lo explicado en la sección de planificación, se ha considerado que la solución óptima sería una metodología de desarrollo ágil. Considerando que el problema puede cambiar a lo largo del proyecto, por lo que necesitamos una metodología adaptativa y flexible, además la forma de trabajar concuerda mucho más (la cantidad del grupo o su propio manejo).

4.1.1 *Metodología de trabajo escogida (SCRUM)*

El nacimiento de la idea de la metodología Scrum fue en el año 1986, con el propósito de aumentar la velocidad de desarrollo de una aplicación y a la vez conservando flexibilidad. Los japonenses Hirotaka Takeuchi e Ikujiro Nonaka se inspiraron en el concepto de un equipo de rugby de 15 jugadores unido y con el mismo objetivo. Durante los años posteriores este concepto fue remodelado y reforzado hasta convertirse en lo que es a día de hoy, el método ágil más usado en el mundo. [28]

Scrum puede llegar a ser ventajoso debido a que el cliente se siente comprometido con el proyecto, implicándose en distintas necesidades de tipo funcional y permitiendo realizar revisiones tempranas de los desarrollos fomentando de esta manera las propias entregas. Otras ventajas a destacar pueden llegar a ser su adaptación o trabajo en equipo. [28]

Los componentes de la metodología Scrum son: [29]

1. Los **Roles de Scrum**: son los papeles o funciones de cada integrante del equipo, que puede ser:
    - *El Dueño del Producto (Product Owner)*: representará la función del cliente, con el objetivo de definir historias de usuario y validar la calidad del producto.
    - *Scrum Master*: su función es velar por la actividad o ejecución del Scrum, para que se realice correctamente por el equipo. También será el responsable de dirigir y concretar los diferentes eventos.

- *El Equipo de Desarrollo*: son los que tienen la función de realizar y concretar las historias de usuario.
- *Stakeholders*: no tienen función en la ejecución o realización del trabajo, pero están interesados en el producto y represan un papel de suma importancia.

2. Los **Eventos Scrum**: son las reuniones que se llevan a cabo en el equipo de trabajo. Están diseñados para la comunicación y colaboración continua entre equipos, con el objetivo de minimizar la necesidad de mantener reuniones indefinidas que pueden afectar de alguna manera a la planificación del proyecto. Se subdividen en estos tipos:
   - *El Sprint*: trata de un tramo de tiempo fijo, que puede llegar a ser entre una y cuatro semanas, durante el cual el producto que se esté desarrollando añade valor gradualmente, de manera que al final de cada sprint el cliente pueda tener una clara visión del producto.
   - *La planificación del Sprint (Sprint Planning)*: es una reunión que se lleva a cabo al comienzo o inicio de cada sprint para planificar el trabajo que se realizará a lo largo del sprint.
   - *Scrum Diario (Dailys)*: reuniones diarias de corta duración, en estas reuniones los miembros del equipo discuten y dialogan el trabajo realizado en el día anterior y el trabajo que se realizará en el día de la reunión.
   - *Revisión del Sprint (Sprint Review)*: una reunión donde cada miembro del equipo presenta lo que ha realizado a lo largo del sprint y lo que debería cambiar o mejorar en el producto para el próximo sprint.
   - *Retrospectiva del Sprint (Sprint Retrospective)*: reuniones con la intención de analizar el trabajo en equipo y discutir áreas para mejorar o colaborar como equipo para el próximo sprint.

3. Los **Artefactos de Scrum**: utilizadas para aumentar la transparencia de la información para que todos los miembros del equipo tengan constancia sobre lo que está ocurriendo. Se subdividen en:
   - *El Backlog del Producto (Product Backlog)*: contienen todas las historias de usuario, pero no evalúan el proyecto, y su gestión o responsabilidad recae en el Dueño del Producto.
   - *El Backlog del Sprint (Sprint Backlog)*: historias de usuario que contienen estimaciones y planes a realizar en el sprint.
   - *Incremento*: es el producto que se produce o ocurre al final del sprint.

4.1.2 *Adaptación de la metodología de trabajo escogida*

Habiendo definido los componentes en la sección anterior, es visible que Scrum está dirigido para trabajar en equipo, por lo que se tendrá que adaptar esta metodología al entorno académico del proyecto, que en el caso de este proyecto se reduciría a mi persona, Nikita Stetskiy. Debido a esto, me veré obligado a representar todos los roles del equipo y mis tutores darán función a los stakeholders y a Scrum Master.

Además, se puede dividir el proyecto en dos productos. La memoria, donde el cliente será el tribunal del trabajo de fin de grado y los tutores serán los Dueños del Producto. Mientras que para la propia aplicación, tendremos un cliente ficticio, por ejemplo Mauricio Mares, y el Dueño del Producto seré yo mismo, puesto que seré el que mejor conoceré los intereses del cliente.

En cuanto a los eventos, el Scrum diario no llega a tener sentido en este entorno, al estar formado por solo una persona y los demás eventos se pueden juntar en un mismo encuentro e iteración.

Finalmente, respecto a la velocidad del proyecto, se llegó a detallar en la sección 3.2 de la Planificación. Cada Sprint tiene acordada una duración de dos semanas.

4.2 ANÁLISIS DEL ENTORNO

Es necesario estudiar el entorno y a los posibles usuarios que necesiten de nuestro servicio antes de empezar a desarrollar el *Backlog* del Producto. Para ello propondré dos diferentes casos ficticios que estudiaremos a continuación.

1ª Persona	
Nombre	Mauricio Mares
Edad	23
Sexo	Masculino
Datos relevantes	
Terminó la carrera de bellas artes, aunque ahora mismo no se siente lo suficientemente apasionado para trabajar de ello. Uno de los principales motivos es el sobreúso de las redes sociales en su día a día.	
Contexto de uso	
El uso que pretende dar este usuario corresponde a un ámbito casual, ya que pretende usar menos las redes sociales, pero a la par quiere mantenerse actualizado de lo que pasa en el mundo. Por lo que la función que requiere de la aplicación es de informarse de eventos importantes que transcurran diariamente o incluso semanalmente.	
Necesidad diferencial	
La funcionalidad a destacar es que la aplicación deberá guardar todas las tendencias del día y poder dar datos que resuman información de dichas tendencias. Además, Mauricio no tiene que estar constantemente atento a la aparición de una nueva tendencia, ya que se van guardando por días. Esto le permitirá a Mauricio reducir las horas de consumo diario de aplicaciones que requerían su atención constante a cambio de información relevante o actual.	

Tabla 4.2: Primer caso de estudio de entorno.

En este primer caso necesitamos que la aplicación sea capaz de resumir información de tendencias diarias y que a la vez no sea atosigante o que agobie con demasiada cantidad de información, por ende, tiene que ser información simple y finita. La posible solución más óptima es organizar tendencias diarias en un *top* o una clasificación de diez tendencias más populares ocurridas a lo largo del día. Además, esto se puede guardar por semanas para que el usuario no tenga la obligación de entrar a la web diariamente.

El carácter de la información, simplificada para el usuario que tendrán las tendencias, será puramente informativo y además intentará ser objetiva. La principal diferencia es que al usar las RRSS, la información relevante se consigue por medio de la atención. Cuantas más publicaciones u opiniones de personas se lean, más informado estará el usuario. Por lo que el factor diferencial, es que la página web en desarrollo mostrará datos simplificados o resumidos de manera objetiva. Todos estos datos estarán comprimidos de tal manera, que el usuario pueda tener una visión clara sobre la tendencia sin la necesidad de leer un alto número de publicaciones u opiniones.

La información que se aporta al usuario en la aplicación web será modular. De tal manera que cada módulo, que tendrá el diseño de una carta, informará de un dato de interés. Los datos más importantes que puede tener una tendencia y que sea útil para el usuario son datos como la relevancia que está teniendo la tendencia, las palabras o *keywords* más usadas, las opiniones de las personas (vistas de manera objetiva a través de un análisis de sentimientos) y diferentes noticias actuales relacionadas con la propia tendencia.

De esta manera, al aportar información de manera simplificada, el usuario Mauricio podrá informarse diariamente de manera adecuada y sin la necesidad de perder el tiempo en las RRSS.

2ª Persona	
Nombre	Nuria Nevadas
Edad	37
Sexo	Femenino
Datos relevantes	
Terminó hace mucho el grado de Empresariales, en los últimos meses empezó a invertir en Bolsa, por su propia cuenta de manera profesional.	
Contexto de uso	
El uso que pretende dar este usuario corresponde a un ámbito profesional. Pretende tener una visión pequeña y objetiva de búsquedas concretas, en este caso del mercado financiero.	
Necesidad diferencial	
La funcionalidad a destacar en esta aplicación es que se podrá buscar tópicos y obtener la misma información actual que se obtienen también con las tendencias, es decir, datos de interés objetivos.	

Tabla 4.3: Segundo caso de estudio de entorno.

Al estudiar el segundo caso, la solución que se le plantea al usuario llega a ser bastante parecida al primer caso. El principal factor

diferencial es que la información que se aporta en el primer caso es sobre las tendencias y en este caso tendrá que ser sobre búsquedas específicas.

No se podrá dar información sobre la popularidad del tópico, ya que no es una tendencia, pero si información sobre las *keywords* o palabras más usadas, el sentimiento general y las noticias más destacables. Esta información tendrá que ser recopilada de distintas publicaciones que tengan relevancia con el tópico.

4.2.1 *Estudio competitivo*

En la actualidad las principales aplicaciones que existen que recopilan tendencias a diario son Google Trends y parcialmente la plataforma de Twitter. Ambas poseen fallos, el principal problema de Twitter se incluye en los riesgos de un uso continuo de una RRSS que se explicó anteriormente en la sección 2. Por otro lado, Google Trends posee multiples problemas a la hora de afrontar nuestras necesidades. Primero, no funciona en todos los países, incluyendo España. Además, no contienen información relevante que pueda resumir una tendencia o el porqué de su relevancia. Aunque Google Trends posea un exhaustivo historial de las búsquedas que ha tenido dicha tendencia, posibles noticias de interés y tópicos similares, no posee el sentimiento general que está teniendo dicha tendencia y tampoco diferentes *keywords* en relación (esto debido a que carece de publicaciones como Twitter). En resumen, ambas aplicaciones principales, tienden a ser incompletas a nuestras necesidades.

También existen otras páginas que usan información de Twitter o Google. Al buscar páginas o aplicaciones web sobre la información de tendencias actuales de Twitter salen resultados como Trendinalia, GetDayTrends o Trends24. Este tipo de páginas web llega a mostrar un contenido puramente estadístico sobre las tendencias, con el uso exclusivo de diagramas o distribuciones globales para explicar el comportamiento de una tendencia.

Realmente este tipo de páginas no indagan en la explicación del contenido de una tendencia para que el usuario pueda estar informado. El objetivo de estas páginas es informar al usuario de datos numéricos que se calculan mientras la tendencia exista, como puede ser la posición de la tendencia respecto a otras durante las últimas 24 horas.

Esto puede llegar a ser información útil si solo quieres saber datos sobre el comportamiento de la tendencia a lo largo de 24 horas. En el caso de nuestra aplicación, aparte de dar datos sobre el comportamiento de la tendencia, tendrá el objetivo de explicar el propósito de la tendencia.

4.2.2 *Estudio competitivo técnico*

Actualmente las librerías más populares sobre recopilación de tendencias actuales son Tweepy y Pytrends, ambas programadas en el lenguaje Python. Dichas librerías distan de ser perfectas, pero cada una tiene características de carácter diferencial.

- **Pytrends** [30]

  Esta librería es una API no oficial que usa información de Google Trends. Por lo que es una librería que aporta información muy valiosa respecto a diferentes tendencias. Esta librería posee numerosas ventajas, a la par que desventajas. Las siguientes características son los métodos que tiene la librería disponible:

  1. *Interés a lo largo del tiempo*: devuelve datos históricos e indexados de las búsquedas sobre una palabra clave. Esta información se muestra en la sección Interés a lo largo del tiempo de Google Trends.
  2. *Interés histórico por hora*: devuelve datos históricos, indexados y por hora de las búsquedas sobre una palabra clave. Envía múltiples solicitudes a Google, cada una de las cuales recupera una semana de datos por hora.
  3. *Interés por región*: devuelve datos sobre el lugar en el que se busca la palabra clave. Información mostrada en la sección Interés por región de Google Trends.
  4. *Temas relacionados*: devuelve datos de las palabras clave relacionadas con una palabra clave proporcionada. Se muestra en la sección Temas relacionados de Google Trends.
  5. *Consultas relacionadas*: devuelve datos de las palabras clave relacionadas con una palabra clave proporcionada. Se muestra en la sección Consultas relacionadas de Google Trends.
  6. *Búsquedas de tendencias*: devuelve datos de las búsquedas de tendencias más recientes. Se muestran en la sección Búsquedas de tendencias de Google Trends.
  7. *Gráficos principales*: devuelve los datos de un tema determinado. Se muestra en la sección Gráficos principales de Google Trends.
  8. *Sugerencia*: devuelve una lista de palabras clave sugeridas adicionales que se pueden usar para refinar una búsqueda de tendencia.

  Estas características pueden llegar a ser extremadamente útiles a la hora de gestionar nuestra aplicación. Aunque las principales desventajas son demasiado grandes como para no tenerlas en cuenta:

1. *Disponibilidad geográfica*: debido a ciertas leyes y normativas, Google Trends no funciona en todos los países del mundo. Por desgracia, España está excluida de momento, aunque se tiene pensado cambiar la normativa o adaptarse a ella.
2. *Longevidad*: esta API al no ser oficial, no tiene garantía de estar funcionando siempre. Ya que llegará un momento donde se cambie el *Back-End* de Google e incidirá en el comportamiento de esta API.
3. *Limitaciones de llamadas*: El límite de peticiones no se conoce públicamente. Aunque los usuarios hicieron medidas aproximadas, donde se demostraba que 1400 solicitudes secuenciales en un período de tiempo de 4 horas daban lugar al límite (replicado en 2 redes), aunque se podía volver a hacer peticiones después de 60 segundos de tiempo de espera.

- **Tweepy** [31]

  Es un paquete de código abierto que brinda una forma muy conveniente de acceder a la API de Twitter con Python. Tweepy incluye un conjunto de clases y métodos que representan los modelos de Twitter y los extremos de la API. Las principales características para nuestro uso son las siguientes:

  1. *Obtención de tendencias cerca de una ubicación*: devuelve las 50 tendencias principales para un WOEID (código de país) específico, si la información de tendencias está disponible. La respuesta es una lista de objetos "trend"que codifican el nombre del tema de tendencia, el parámetro de consulta que se puede usar para buscar el tema en la búsqueda de Twitter y la URL de búsqueda de Twitter. Esta información se almacena en caché durante 5 minutos.
  2. *Búsqueda de publicaciones*: devuelve una colección de publicaciones o *tweets* relevantes que coinciden con una consulta específica. Aunque la API de búsqueda no pretende ser una fuente exhaustiva de Tweets. No todos los Tweets se indexarán o estarán disponibles a través de la interfaz de búsqueda.

  Estas características pueden llegar a cumplir las necesidades explicadas anteriormente, aunque también tenemos que tener en cuenta las desventajas que presenta esta API.

  1. *Disponibilidad de versiones*: existen actualmente varias versiones de la API publicadas, cada una se accede de una manera diferente y tiene métodos diferentes. La documentación oficial de Twitter no está bien cuidada y no funciona de la misma manera que la aplicación web. Un sencillo

ejemplo de ello es que en la web se puede buscar tweets de un país en concreto, mientras que esta función no existe en la API.

2. *Historial pobre*: mientras que Google guarda todos los datos de interés sobre una tendencia, Twitter no, por lo que búsquedas antiguas no contendrán mucho valor al usuario en cuanto a la relevancia o interés que presenta un tópico.

4.2.3 *Conclusiones de los estudios realizados*

En los apartados anteriores se definieron claramente las ventajas y desventajas del uso de ambas aplicaciones o APIs. Se puede concluir que, aunque Google Trends posea más funciones versátiles y con mayor peso de utilidad en cuanto a nuestras necesidades, sigue teniendo mayores desventajas a la hora de usarlo. La mayor de todas es que no esté disponible en todos los países. La plataforma de twitter en cambio se usa en casi todo el mundo, además se pueden llegar a implementar funciones parecidas con métodos propios y no tener que depender en un mayor nivel de una API externa.

En cuanto a los aspectos que se deberán tratar a la hora de definir las características o funcionalidades de la aplicación se definirán en la siguiente lista. Aunque se entrará más en detalle en la sección posterior.

- Las personas requieren información relevante, concisa y simplificada.
- La información debe ser de contenido finito, a la misma hora aportando al usuario su pleno entendimiento.
- Los usuarios deberán ser capaces de usar la aplicación en todos los dispositivos que tengan acceso web. Además, deberá tener un diseño *responsive*, para que se adapte a la interfaz del usuario.
- Los usuarios tienen que ser capaces de navegar a través de distintas tendencias. Poder elegir tanto el país de búsqueda como la fecha para poder informarse debidamente.
- Los usuarios tienen que ser capaces de realizar búsquedas de distintos tópicos de interés y poder informarse con datos relacionados y actuales.
- El diseño será presentado mediante módulos o cartas. Cada carta tendrá información útil y de necesidad al usuario.
- Un módulo contendrá datos sobre la popularidad de la tendencia y su correspondiente gráfico.

- Otro módulo contendrá datos sobre palabras y *keywords* más usadas sobre la tendencia y su correspondiente gráfico.
- Otro módulo contendrá un sentimiento general y un gráfico visual sobre dicho comportamiento.
- También habrá tres módulos de noticias de interés general sobre la tendencia.
- Cada módulo deberá respetar tanto el idioma como la localización elegida por el usuario.
- Se deberá tener en cuenta que el diseño sea simple y agradable al usuario.
- Otro factor a tener en cuenta es el plan de peticiones que ofrece Twitter, el cual es muy limitado.

4.3 REQUISITOS DEL SISTEMA

Gracias a los estudios y conclusiones realizadas en el capítulo anterior, se puede detallar de una manera más sencilla los siguientes requerimientos o requisitos. Los requerimientos funcionales funcionan como servicios que puede prestar el sistema. Mientras que los requerimientos no funcionales especifican más bien la fiabilidad o el comportamiento de este.

4.3.1 *Requisitos Funcionales*

R.F. 1 *Gestión de Trends*. El sistema deberá ser capaz de realizar llamadas al servicio de la API de Twitter para recopilar tendencias. En la API o su plataforma se denominan Trends o TT.

- R.F. 1.1 *Gestión de Países*. El sistema deberá ser capaz de manejar la petición concorde a diferentes países y almacenarlas adecuadamente.
  - R.F. 1.1.1 *Búsqueda por WOEID*. El sistema deberá ser capaz de buscar los diferentes países por el identificador que provee Yahoo y que utiliza la plataforma de Twitter. En este caso se denominan WOEIDs.
- R.F. 1.2 *Gestión de la Fecha*. El sistema deberá ser capaz de manejar la respuesta generada por Twitter y poder diferenciarla por fecha, bajo el formato de yy-mm-dd.
- R.F. 1.3 *Limpieza de Trends*. El sistema deberá poder limpiar los Trends repetidos. Ya que la mayoría de las veces Twitter no reconoce el mismo TT, casos como el uso de tilde, mayúsculas o si son una sucesión de varias palabras. Por ejemplo, «Balón», «balón», «balon» y «Balón de Oro» se referirán a la misma tendencia, pero Twitter los trata como Trends diferentes con datos respectivos diferentes.
  - R.F. 1.3.1 *Preferencias de Limpieza*. El sistema preferirá un nombre de tendencias más largo y por defecto si son iguales, preferirá el que tenga más popularidad.
- R.F. 1.4 *Ordenación de los Trends*. El sistema deberá ser capaz de ordenar los TT bajo el atributo del volumen de popularidad proporcionado por Twitter.
- R.F. 1.5 *Historial Inteligente*. Además de guardar los Trends ordenados, al usuario le será importante que se guarden y se vayan comparando los Trends viejos y los nuevos (mediante el atributo de la popularidad).

R.F. 2 *Gestión de Popularidad*. El sistema deberá ser capaz de guardar el atributo del volumen de popularidad en una lista, que fue anteriormente proporcionado por Twitter.

R.F. 2.1 *Cálculo de Media*. El sistema deberá poder sacar la media o la popularidad promedio a partir de la lista guardada anteriormente.

R.F. 2.2 *Cálculo de Moda*. El sistema deberá poder sacar la moda o la popularidad más grande a partir de la lista guardada anteriormente.

R.F. 2.3 *Gestión de Hora*. El sistema tiene que guardar en una lista la hora a la que se ha calculado el valor de la popularidad.

R.F. 2.4 *Gestión de Gráfico*. Se deberá proporcionar un gráfico a partir de los valores calculados anteriormente. Proporcionado en formato de área.

R.F. 2.4.1 *Eje X*. Se deberá proporcionar la lista de la hora a la que se ha calculado la popularidad como eje X del gráfico.

R.F. 2.4.2 *Eje Y*. Se deberá proporcionar la lista del vector de popularidad como el eje Y del gráfico.

R.F. 2.4.3 *Análisis del Gráfico*. Además del gráfico visual, se deberán proporcionar datos de valor. En este caso, el primer dato calculado y el último, además de mostrar la hora de su cálculo.

R.F. 3 *Gestión de Tweets*. El sistema deberá ser capaz de guardar como máximo alrededor de cien publicaciones o tweets de usuarios diferentes.

R.F. 3.1 *Filtrado de Tweets*. Se debe filtrar los tweets mediante la API de Twitter. Primero que las publicaciones tengan un nº mínimo de interacciones y que a la vez sean recientes, luego por el lenguaje del país y que sean publicaciones originales no repetidas.

R.F. 3.2 *Limpieza de Tweets*. Se debe limpiar los tweets como los nombres de usuarios, enlaces u otros caracteres que complicarían el análisis del texto.

R.F. 3.2.1 *Gestión de Stop Words*. El sistema deberá diferenciar las palabras significativas de las que no, frecuentemente denominadas como *stop words* también llamadas palabras vacías o palabras comunes, son palabras que no suelen aportar significado a la hora de analizar textos. Cada idioma tiene su propia lista de *stop words*.

R.F. 4 *Clasificación de Palabras y Gestión de Keywords*. Al tener un conjunto de palabras significativas, el sistema deberá disponerlas al usuario como un top o clasificación ordenadas por el número de repeticiones en el texto. Por otro lado, el sistema deberá extraer *keywords* relevantes del conjunto de tweets mediante un algoritmo de extracción.

R.F. 4.1 *Filtrado de Keywords*. Se deberán eliminar *keywords* parecidas o repetidas.

R.F. 4.2 *Gestión de Gráfico*. Se deberá proporcionar un gráfico a partir de los valores calculados anteriormente. Mostrado como un gráfico de tipo barras radial.

R.F. 4.2.1 *Conjunto de Datos*. El conjunto de datos será proporcionado por la clasificación de seis palabras más repetidas de los tweets recogidos. Además, deberán representarse como un porcentaje.

R.F. 5 *Gestión de Sentimiento General*. Al tener un conjunto de palabras significativas, el sistema deberá disponerlas al usuario como un sentimiento general, clasificando el conjunto de palabras como positivo, negativo o neutral.

R.F. 5.1 *Gestión de Gráfico*. Se deberá proporcionar un gráfico a partir de los valores calculados anteriormente. Mostrado como un gráfico de tipo burbujas con eje X y eje Y.

R.F. 5.1.1 *Eje X*. Este eje representará el sentimiento general de las publicaciones o tweets.

R.F. 5.1.2 *Eje Y*. Este eje representará la relación con la clasificación de palabras, calculadas anteriormente, de las publicaciones o tweets.

R.F. 5.1.3 *Tamaño de las Burbujas*. Se representará un tamaño mayor o menor, en base a la popularidad de los tweets o las publicaciones.

R.F. 6 *Gestión de Noticias*. Se deberán proporcionar tres noticias adecuadas a la tendencia. Se buscarán noticias parecidas gracias al nombre y las *keywords* relacionadas a la tendencia.

R.F. 7 *Gestión de Rutas*. El sistema deberá disponer de rutas de navegación por cada país y fecha correspondiente a las tendencias.

R.F. 8 *Sistema de Búsqueda*. El sistema deberá disponer de un sistema de búsqueda que proporcione tópicos y datos.

R.F. 9 *Gestión de Guardado*. El sistema deberá poder guardar las tendencias y que el usuario pueda acceder a diferentes países o fechas.

4.3.2 *Requisitos No Funcionales*

R.N. 1 *Diseño*. El diseño debe ser sencillo, minimalista, pero a la vez poder dar toda la información que el usuario requiera. Además, debe ser *responsive*, enfocada en el uso móvil.

R.N. 2 *Disponibilidad*. El sistema debe intentará estar disponible siempre y actualizar el contenido relevante cada hora. Confiando en *plataforma as a service*, «plataforma como servicio» (PaaS) a la hora de construir, correr y mantener la aplicación.

R.N. 3 *Arquitectura de capas*. El sistema deberá tener una base de datos, una capa de negociación o gestión y por último una capa de presentación o interfaz.

R.N. 4 *Tests*. El sistema deberá estar *testeado* para poder comprobar su correcto funcionamiento. Además, tendrá reglas de escritura de código específicas, considerado como buenas prácticas para dicho lenguaje de programación.

4.4 PRODUCT BACKLOG

Después de listar los distintos requisitos, podemos visualizar más fácilmente la lista de Historias de Usuario que se trabajará y desarrollará posteriormente. La siguiente tabla contiene las diferentes Historias de Usuario, donde la estimación del esfuerzo (E) está expresada mediante los Puntos de Historia (1 a 10) y la prioridad (P) representada del 1 a 3, siendo el 1 el más prioritario para el Dueño del Producto, es decir, los tutores.

ID	Título de la Historia	E	P
H.U. 1	Cualquier usuario puede usar la aplicación sin necesidad de registrarse.	1	1
H.U. 2	El usuario debe poder visualizar las diez tendencias más populares por defecto.	5	1
H.U. 3	El usuario debe poder visualizar las diez tendencias más populares, pudiendo seleccionar el país como parámetro.	5	1
H.U. 4	El usuario debe poder visualizar las diez tendencias más populares, pudiendo seleccionar la fecha como parámetro.	5	1
H.U. 5	El usuario puede buscar sus propios tópicos y visualizarlos del mismo modo que las tendencias.	4	2
H.U. 6	El usuario debe saber, por medio de alertas, si el servicio o los parámetros no han funcionado.	3	3
H.U. 7	El usuario tiene que poder navegar intuitivamente, es decir por medio de gestos de *scroll*, por la página.	7	1
H.U. 8	El usuario debe poder analizar datos de interés sobre la popularidad.	8	1
H.U. 9	El usuario debe poder visualizar las palabras más repetidas o comunes mediante un porcentaje, a partir de los tweets recogidos referentes a la tendencia.	8	1
H.U. 10	El usuario debe poder visualizar las *keywords*, a partir de los tweets recogidos referentes a la tendencia.	8	1
H.U. 11	El usuario puede analizar los sentimientos generales mediante un porcentaje mostrado.	9	2
H.U. 12	El usuario debe poder ver tres noticias más actuales relacionadas con la tendencia y poder acceder a ellas.	10	1

Tabla 4.4: Títulos de Historias de Usuario.

Además, algunas Historias de Usuario pueden tener diferentes subniveles, como se detalla en la siguiente tabla.

ID	Título de la Historia	E	P
H.U. 8.1	El usuario debe poder visualizar la popularidad que está teniendo la tendencia por medio de una gráfica de área.	8	1
H.U. 9.1	El usuario verá los datos representados mediante un gráfico de radial de barras.	8	1
H.U. 11.1	El usuario debe poder ver los sentimientos generales, a partir de los tweets recogidos referentes a la tendencia, mediante una gráfica de burbujas.	9	2

Tabla 4.5: Subniveles de Títulos de Historias de Usuario.

4.5 SPRINT BACKLOG

Quedando los *Product Backlog* concluidos, ahora podemos dividirlos en los distintos *Sprint* acordados previamente.

4.5.1 *Sprint 1*

ID	Título de la Historia	E	P
H.U. 1	Cualquier usuario puede usar la aplicación sin necesidad de registrarse.	1	1
H.U. 6	El usuario debe saber, por medio de alertas, si el servicio o los parámetros no han funcionado.	3	3

Tabla 4.6: Títulos del Sprint 1.

H.U. 1	Cualquier usuario puede usar la aplicación sin necesidad de registrarse.
Descripción: El usuario debe poder entrar a la página web y poder visualizar su contenido sin ningún tipo de restricción. Esto debido a que el contenido aportado por la aplicación web debe centrarse al público general.	
Estimación: 1	**Prioridad**: 1
Pruebas de aceptación: ▪ Entrar a cualquier URL dinámico de la página y poder obtener una respuesta. ▪ Ante un error de carga de contenido o una URL mal escrita, el usuario deberá saber el error específico.	
Tareas: ▪ Instalar Vue.js, sus dependencias y el enrutador. ▪ Instalar un lint adecuado para Vue. ▪ Crear rutas y composiciones que devuelvan una página con contenido.	
Observaciones: ▪ Tener en cuenta rutas y errores por defecto.	

Tabla 4.7: Descomposición de la Historia de Usuario 1.

H.U. 6	El usuario debe saber, por medio de alertas, si el servicio o los parámetros no han funcionado.
Descripción: El usuario debe poder entrar a la página web y poder si está funcionando o no. En caso de fallo, deberá poder ver un mensaje de error propiamente explicado el fallo en concreto.	
Estimación: 3	**Prioridad**: 3
Pruebas de aceptación: ▪ Poder entrar a diferentes rutas por erróneas o cuando el servicio esté caído y diferenciar diferentes errores. ▪ Diferenciar errores como la carga de contenido, el propio funcionamiento del servidor o contenido erróneo.	
Tareas: ▪ Crear rutas por defecto, con alertas definidas para diferentes tipos de errores.	
Observaciones: ▪ Tener en cuenta todos los parámetros a la hora de realizar peticiones al servidor, como el parámetro de *status* de una petición.	

Tabla 4.8: Descomposición de la Historia de Usuario 6.

4.5.2 *Sprint 2*

ID	Título de la Historia	E	P
H.U. 2	El usuario debe poder visualizar las diez tendencias más populares por defecto.	5	1

Tabla 4.9: Títulos del Sprint 2.

H.U. 2	El usuario debe poder visualizar las diez tendencias más populares por defecto.
Descripción: El usuario debe poder visualizar el contenido de tendencias de un país en el día actual en la página de defecto, sin tener que poner ningún parámetro.	
Estimación: 5	**Prioridad**: 1
Pruebas de aceptación: ▪ Entrar a la página web por defecto que haya una petición correcta al servidor. ▪ Que el servidor devuelva las tendencias de un país y fecha del día actual.	
Tareas: ▪ Instalar Tweepy, sus dependencias y aprender las diferentes consultas a la API de Twitter. ▪ Filtrar las tendencias parecidas o iguales. Diferenciando tildes, minúsculas, mayúsculas o simplemente frases más largas. ▪ Manejar las diferentes entidades a la hora de construir el contenido a devolver para el usuario. ▪ Manejar una Base de Datos Mongo y definir distintos módulos para su correcto funcionamiento.	
Observaciones: ▪ La Base de Datos se tendrá que actualizar cada hora. ▪ Al momento de actualizar la Base de Datos se debe realizar una Unión y Diferencia Simétrica de los datos viejo y nuevos. Con la Unión se actualizan los datos viejos y con la Diferencia Simétrica, agregamos los datos nuevos junto a los viejos.	

Tabla 4.10: Descomposición de la Historia de Usuario 2.

4.5.3 *Sprint 3*

ID	Título de la Historia	E	P
H.U. 3	El usuario debe poder visualizar las diez tendencias más populares, pudiendo seleccionar el país como parámetro.	5	1
H.U. 4	El usuario debe poder visualizar las diez tendencias más populares, pudiendo seleccionar la fecha como parámetro.	5	1

Tabla 4.11: Títulos del Sprint 3.

H.U. 3	El usuario debe poder visualizar las diez tendencias más populares, pudiendo seleccionar el país como parámetro.
Descripción: El usuario debe poder entrar a la página web y poder visualizar el contenido, aportando el parámetro del país donde quiera buscar la tendencia.	
Estimación: 5	**Prioridad**: 1
Pruebas de aceptación: ▪ Entrar al URL dinámico de la página como país como parámetro y que el servidor devuelva el objeto correspondiente. ▪ Ante un error de carga de contenido o una URL mal escrita, el usuario deberá saber el error específico.	
Tareas: ▪ Configuración de respectivos enrutadores. Tanto en la parte Front-end y Back-end. ▪ Se deben poder buscar en la Base de Datos por países, con su fecha correspondiente. De modo que ya debe haber objetos creados con país como valor identificativo.	
Observaciones: ▪ Ninguna.	

Tabla 4.12: Descomposición de la Historia de Usuario 3.

H.U. 4	El usuario debe poder visualizar las diez tendencias más populares, pudiendo seleccionar la fecha como parámetro.
Descripción: El usuario debe poder visualizar las diez tendencias más populares, pudiendo seleccionar la fecha como parámetro.	
Estimación: 5	**Prioridad**: 1
Pruebas de aceptación: ▪ Entrar al URL dinámico de la página como fecha como parámetro y que el servidor devuelva el objeto correspondiente. ▪ Ante un error de carga de contenido o una URL mal escrita, el usuario deberá saber el error específico.	
Tareas: ▪ Configuración de respectivos enrutadores. Tanto en la parte Front-end y Back-end. ▪ Se deben poder buscar en la Base de Datos por fecha, con su país correspondiente. De modo que ya debe haber objetos creados con fecha como valor identificativo.	
Observaciones: ▪ Ninguna.	

Tabla 4.13: Descomposición de la Historia de Usuario 4.

4.5.4 *Sprint 4*

ID	Título de la Historia	E	P
H.U. 5	El usuario puede buscar sus propios tópicos y visualizarlos del mismo modo que las tendencias.	4	2

Tabla 4.14: Títulos del Sprint 4.

H.U. 5	El usuario puede buscar sus propios tópicos y visualizarlos del mismo modo que las tendencias.
Descripción: El usuario puede buscar sus propios tópicos, mediante un buscador general, y visualizarlos del mismo modo que las tendencias. Con las mismas características que las tendencias, menos la popularidad, que solo se puede calcular si es una tendencia, un dato proveniente de Twitter.	
Estimación: 4	**Prioridad**: 2
Pruebas de aceptación: ▪ Que el usuario identifique el buscador y pueda escribir sus propios tópicos que desee buscar. ▪ Que el usuario pueda cambiar de país como parámetro al buscador para los tópicos.	
Tareas: ▪ Adaptar el diseño para el buscador en Front-end. ▪ Adaptar la ruta correspondiente en forma de *query* y que el servidor sea capaz de procesarla.	
Observaciones: ▪ El tópico no se guarda en la Base de Datos, sino que se devuelve cada vez que se inicia una búsqueda.	

Tabla 4.15: Descomposición de la Historia de Usuario 5.

4.5.5 *Sprint 5*

ID	Título de la Historia	E	P
H.U. 7	El usuario tiene que poder navegar intuitivamente, es decir por medio de gestos de *scroll*, por la página.	7	1

Tabla 4.16: Títulos del Sprint 5.

H.U. 7	El usuario tiene que poder navegar intuitivamente, es decir por medio de gestos de *scroll*, por la página.
Descripción: El usuario tiene que poder navegar intuitivamente. Tanto haciendo *scroll* verticalmente o horizontalmente por la página web. Además, el contenido debe desplazarse automáticamente al centro de la pantalla, ocupando el mayor tamaño de espacio disponible. Todo esto por medio de gestos, tanto en una pantalla grande o en un móvil.	
Estimación: 7	**Prioridad**: 1
Pruebas de aceptación: ▪ Que el usuario pueda hacer *scroll* verticalmente y el contenido se mueva verticalmente dependiendo del gesto del usuario. ▪ Que el usuario pueda hacer *scroll* horizontalmente y el contenido se mueva horizontalmente dependiendo del gesto del usuario.	
Tareas: ▪ Adaptar el diseño en Front-end. Esto se hará gracias al *framework* Tailwind CSS. Consiguiendo que el gesto del usuario sea un *scroll* automático y el contenido se centre en medio de la pantalla siempre después de procesar cualquier gesto. ▪ Adaptar Vue para que la página web pueda proporcionar contenido dinámico, es decir, un vector de tendencias que puede tener diferente tamaño cada vez que se haga una petición al servidor.	
Observaciones: ▪ El diseño, al ser de cartas o módulos es fácilmente adaptable, pero hay que tener en cuenta las diferentes versiones de escritorio o móvil.	

Tabla 4.17: Descomposición de la Historia de Usuario 7.

4.5.6 *Sprint 6*

ID	Título de la Historia	E	P
H.U. 8	El usuario debe poder analizar datos de interés sobre la popularidad.	8	1
H.U. 8.1	El usuario debe poder visualizar la popularidad que está teniendo la tendencia por medio de una gráfica de área.	8	1

Tabla 4.18: Títulos del Sprint 6.

H.U. 8	El usuario debe poder analizar datos de interés sobre la popularidad.
Descripción: Uno de los contenidos principales sobre una tendencia para el usuario serán datos de interés sobre la popularidad de la tendencia. Datos como la moda, la media y el primer o último dato calculado.	
Estimación: 8	**Prioridad**: 1
Pruebas de aceptación: ▪ Que el usuario pueda informarse debidamente de la media o la popularidad promedio de la tendencia. ▪ Que el usuario pueda informarse debidamente de la moda o a popularidad más grande alcanzada de la tendencia. ▪ Que el usuario pueda informarse debidamente del primer y último volumen de popularidad calculados de la tendencia.	
Tareas: ▪ Adaptar el diseño en Front-end al diseño de la carta y desarrollar funciones para que el número tenga un separador de millares para una lectura más fácil. ▪ Crear el modelo correspondiente en la Base de Datos.	
Observaciones: ▪ Ninguna.	

Tabla 4.19: Descomposición de la Historia de Usuario 8.

H.U. 8.1	El usuario debe poder visualizar la popularidad que está teniendo la tendencia por medio de una gráfica de área.
Descripción: Uno de los contenidos principales sobre una tendencia para el usuario será la gráfica de área que representará la propia popularidad de la tendencia. El eje X será la hora a la que se ha calculado la popularidad y el eje Y será la popularidad de la tendencia en cuestión.	
Estimación: 8	**Prioridad**: 1
Pruebas de aceptación: ▪ Que el usuario pueda informarse debidamente con el contenido gráfico sobre la popularidad de una tendencia en un momento dado.	
Tareas: ▪ Adaptar el diseño en Front-end. Gracias a la librería ApexCharts.js, podremos mostrar al usuario un gráfico solamente necesitando dos listas, el eje X y el eje Y que habrán sido previamente calculados. ▪ Se adaptará el gráfico al diseño de la carta, además se apagará toda la interactividad con él porque su función solo es gráfica.	
Observaciones: ▪ El diseño tiene que ser *responsive*, afectando también al gráfico.	

Tabla 4.20: Descomposición de la Historia de Usuario 8.1.

4.5.7 *Sprint 7*

ID	Título de la Historia	E	P
H.U. 9	El usuario debe poder visualizar las palabras más repetidas o comunes mediante un porcentaje, a partir de los tweets recogidos referentes a la tendencia.	8	1
H.U. 9.1	El usuario verá los datos representados mediante un gráfico de radial de barras.	8	1
H.U. 10	El usuario debe poder visualizar las *keywords*, a partir de los tweets recogidos referentes a la tendencia.	8	1

Tabla 4.21: Títulos del Sprint 7.

H.U. 9	El usuario debe poder visualizar las palabras más repetidas o comunes, a partir de los tweets recogidos referentes a la tendencia.
Descripción: Uno de los contenidos principales sobre una tendencia para el usuario será visualizar las palabras más repetidas o comunes de una tendencia. Mostrado en porcentaje de aparición respecto a todas las publicaciones o tweets.	
Estimación: 8	**Prioridad**: 1
Pruebas de aceptación: ▪ El usuario debe informarse debidamente de las palabras más repetidas a través de cálculos de porcentaje sobre las apariciones de las palabras en los tweets.	
Tareas: ▪ Adaptar el diseño en Front-end y que conjunte con los demás elementos. ▪ Filtrar las palabras significativas, eliminando las que pertenezcan al grupo de las *stop words* o palabras vacías, es decir las que no suelen aportar significado a la hora de analizar textos. Posteriormente sólo se enseñarán las seis palabras más comunes. ▪ Crear el modelo correspondiente en la Base de Datos.	
Observaciones: ▪ El diseño tiene que ser *responsive*.	

Tabla 4.22: Descomposición de la Historia de Usuario 9.

H.U. 9.1	El usuario verá los datos representados mediante un gráfico de radial de barras.
Descripción: Otro contenido único, que aportará más significado a la hora de visualizar las palabras más repetidas o comunes de una tendencia será un gráfico de radial de barras, donde cada barra representará la palabra y el porcentaje de sus repeticiones.	
Estimación: 8	**Prioridad**: 1
Pruebas de aceptación: ▪ El usuario debe informarse debidamente de las palabras más repetidas a través de un gráfico de radial de barras. ▪ El usuario debe poder comparar todas las palabras repetidas. También saber cuál es la que tiene más valor y la que menos.	
Tareas: ▪ Adaptar el diseño en Front-end, gracias a la librería ApexCharts.js. ▪ Se debe configurar el gráfico de tal manera, que cada palabra y su valor tengan una representación única.	
Observaciones: ▪ El diseño tiene que ser *responsive*, afectando también al gráfico.	

Tabla 4.23: Descomposición de la Historia de Usuario 9.1.

H.U. 10	El usuario debe poder visualizar las *keywords*, a partir de los tweets recogidos referentes a la tendencia.
Descripción: Uno de los contenidos principales sobre una tendencia para el usuario será visualizar las *keywords* más relevantes sobre los textos recogidos de las publicaciones o tweets.	
Estimación: 8	**Prioridad**: 1
Pruebas de aceptación: ▪ El usuario debe informarse de cuáles son las *keywords* más relevantes de una tendencia. ▪ Las *keywords* deben expresar al usuario una perspectiva de lo que es relevante o de importancia en el conjunto de las publicaciones o tweets de la tendencia.	
Tareas: ▪ Adaptar el diseño en Front-end. ▪ Configurar el algoritmo YAKE, adaptándolo a extraer un conjunto de textos que serán parecidos, ya que son publicaciones sobre un mismo tópico. ▪ Filtrar el texto antes de extraer las *keywords* con el algoritmo YAKE. ▪ Filtrar las *keywords* parecidas o iguales, luego enseñar solo las tres más relevantes.	
Observaciones: ▪ El diseño tiene que ser *responsive*.	

Tabla 4.24: Descomposición de la Historia de Usuario 10.

4.5.8 *Sprint 8*

ID	Título de la Historia	E	P
H.U. 11	El usuario puede analizar los sentimientos generales mediante un porcentaje mostrado.	9	2
H.U. 11.1	El usuario debe poder ver los sentimientos generales, a partir de los tweets recogidos referentes a la tendencia, mediante una gráfica de burbujas.	9	2

Tabla 4.25: Títulos del Sprint 8.

H.U. 11	El usuario puede analizar los sentimientos generales mediante un porcentaje mostrado.
Descripción: Uno de los contenidos principales sobre una tendencia para el usuario será visualizar el sentimiento general que tienen los textos recogidos de las publicaciones o tweets. En este caso, será mostrado mediante un porcentaje.	
Estimación: 9	**Prioridad**: 2
Pruebas de aceptación: ▪ El usuario debe informarse de cuál es el sentimiento general de una tendencia y saber cuál es el porcentaje correspondiente de los sentimientos positivos, negativos y neutrales.	
Tareas: ▪ Mostrar al usuario tres tipos de porcentaje (positivos, negativos y neutrales). El porcentaje será calculado mediante el tamaño de la burbuja, ya que el tamaño o radio representa su relevancia. ▪ Crear el modelo correspondiente en la Base de Datos.	
Observaciones: ▪ El diseño tiene que ser *responsive* y adaptarse al contenido gráfico con sus colores correspondientes.	

Tabla 4.26: Descomposición de la Historia de Usuario 11.

H.U. 11.1	El usuario debe poder ver los sentimientos generales, a partir de los tweets recogidos referentes a la tendencia, mediante una gráfica de burbujas.
Descripción: Uno de los contenidos principales sobre una tendencia para el usuario será visualizar el sentimiento general que tienen los textos recogidos de las publicaciones o tweets. El análisis de sentimientos será basado en reglas de léxicos y se mostrará mediante un gráfico.	
Estimación: 9	**Prioridad**: 2
Pruebas de aceptación: ▪ El usuario debe informarse de cual es el sentimiento general de una tendencia de una manera visual o gráfica. Se le mostrará al usuario un gráfico de burbujas.	
Tareas: ▪ Configurar la herramienta VADER y adaptar el léxico de cada país. ▪ Mostrar al usuario tres tipos de burbuja (Positivo, Negativo y Neutral). Cada burbuja tendrá un tamaño diferente dependiendo de su relevancia, mediante la popularidad de la publicación o si hay muchos sentimientos parecidos. ▪ Mostrar un posicionamiento de la burbuja relevante. El eje X será el sentimiento general de una publicación. Mientras el eje Y representará la relación con la clasificación de palabras relevantes, que fueron calculadas anteriormente.	
Observaciones: ▪ El diseño tiene que ser *responsive*.	

Tabla 4.27: Descomposición de la Historia de Usuario 11.1.

4.5.9 *Sprint 9*

ID	Título de la Historia	E	P
H.U. 12	El usuario debe poder ver tres noticias más actuales relacionadas con la tendencia y poder acceder a ellas.	10	1

Tabla 4.28: Títulos del Sprint 9.

H.U. 12	El usuario debe poder ver tres noticias más actuales relacionadas con la tendencia y poder acceder a ellas.
Descripción: Uno de los contenidos principales sobre una tendencia para el usuario será visualizar las distintas noticias relevantes que pueden tener alguna relación las tendencias.	
Estimación: 10	**Prioridad**: 1
Pruebas de aceptación: ▪ El usuario debe informarse de tres noticias con editoriales diferentes, pero que tengan afinidad respecto a la tendencia. ▪ El usuario podrá ver una foto, un título, la editorial, la fecha de publicación y algún texto sobre el articulo o noticia.	
Tareas: ▪ Desarrollar *web scraping* o raspado web, una técnica utilizada para extraer información de sitios web. En nuestro caso se utilizará para extraer noticias relevantes desde Google News y posteriormente desde cada página del artículo. ▪ Se buscarán artículos relacionado mediante el nombre de la tendencia y algunas palabras relevantes que se extrajeron anteriormente, con el objetivo de precisar la búsqueda. ▪ Crear el modelo correspondiente en la Base de Datos.	
Observaciones: ▪ El diseño tiene que ser *responsive* y adaptarse al contenido, tanto con la foto o el texto.	

Tabla 4.29: Descomposición de la Historia de Usuario 12.

5 DISEÑO

En este capítulo se pretende abordar la principal tarea del diseño o la planificación de la estructura idónea sobre las necesidades que se han ido explicando en capítulos anteriores. Para ello se intentará plantear las decisiones tomadas desde un nivel alto, eligiendo la estructura o arquitectura correcta, bajando gradualmente hasta encajar todos los componentes, con el objetivo de obrar un sistema escalable y ágil que proporcione la solución a los problemas supuestos del cliente.

5.1 ARQUITECTURA PLANTEADA

Las necesidades que se han ido explicando en capítulos anteriores se pueden dividir en diferentes aplicaciones, funcionalidades o divisiones de capas. Cada aplicación debe funcionar cumpliendo un único objetivo, pero finalmente deben coaccionar como una sola aplicación para el usuario final. Las distintas funcionalidades que surgen ante las necesidades que se explicaron anteriormente se pueden dividir en tres tipos (operaciones de datos, su gestión y su visualización). Al tratar de construir una aplicación que requiere muchas capas horizontales independientes y a la vez que puedan interactuar entre ellas, el patrón de software que surge ante esta necesidad es el patrón arquitectónico en capas. Además de ser un patrón muy común en el ámbito web, es bastante sencillo de entender e implementar. Cada capa tiene un papel determinado y una responsabilidad específica dentro de la aplicación, dichas capas pueden cooperar e interactuar entre sí, pero no dependen una de otra. Se debe saber que una capa de nivel superior puede interactuar con la capa de nivel inferior, pero no al revés. [32]

Por norma general, las capas de una aplicación tienen la posibilidad de residir en una sola maquina física (misma capa) o pueden estar compartidas sobre diferentes máquinas (n-capas). En nuestro caso, la solución más óptima sería n-capas, debido a la posibilidad de mejorar la escalabilidad y optimización. Además, al dividir el proyecto por capas se mejora la flexibilidad y se puede usar en diferentes plataformas como servicio, esto último debido a que es fácilmente portable. Gracias a ello se puede dar uso de PaaS como Vercel, Heroku o MongoDB Atlas, consiguiendo numerosas ventajas ya que se facilita la configuración, el despliegue, las tareas de automatización y el escalado.

La arquitectura de n-capas se suele dividir en tres. Las necesidades diferenciales que se han planteado son operar datos, gestionarlos y visualizarlos. Estas necesidades se traducen a distintas capas en la arquitectura elegida. Desde el nivel bajo hasta el más alto, la capa que se encargaría de operar con los datos se domina Capa de Base de Datos, luego la capa encargada de gestionarlos se domina Capa de Dominio o Negocio y por último la capa encargada de visualizarlos es dominada como Capa de Presentación. Cada una de ellas tiene funcionalidades y características propias, además pueden llegar a subdividirse en distintos niveles para la correcta estructuración de la aplicación.

Cada capa consigue independencia lógica ya que funciona mediante una aplicación independiente. La indecencia física la podemos conseguir usando las diferentes plataformas como servicio PaaS planteadas anteriormente. La comunicación de distintas peticiones será bidireccional, aunque siempre siguiendo la norma de que una capa inferior no puede llegar a usar los servicios de una capa superior.

Figura 5.1: Diagrama conceptual de la arquitectura.

5.2 CAPA DE BASE DE DATOS

La capa de base de datos se encarga de las operaciones de los propios datos de la aplicación. Esta capa es invisible para el usuario, ya que su funcionamiento no le incumbe. En esta capa residen los datos y además se acceden a los mismos, mediante solicitudes de almacenamiento o recuperación de información desde la capa de dominio o negocio. Comúnmente esta capa suele pertenecer a la parte Back-end del sistema.

Al tener planteado el *Sprint Backlog* es importante estudiar y analizar las herramientas que compondrán la estructuración de esta capa. Incluso aunque se haya dado ligeras pinceladas de un estudio técnico (4.2.2), sigue siendo crucial reiterar y volver a incidir en este apartado antes de adentrarse en el diseño de dicha capa.

5.2.1 *Estudio técnico*

Los dos grandes pilares o modelos que existen actualmente en las bases de datos son denominados SQL (base de datos relacional) y NoSQL (base de datos no relacional). La principal diferencia es que NoSQl no requiere una estructura sólida y definida para trabajar con los datos, lo cual para nuestra aplicación es conveniente ya que requiere de flexibilidad a la hora de operar datos, debido a que estos cambian y se actualizan en un determinado intervalo de tiempo.

Se ha elegido MongoDB como el sistema de base de datos NoSQL, el cual es orientado a documentos y de código abierto. Es de los sistemas más fiables que existen actualmente, además de su conveniencia por ser orientado a objetos, simple, dinámico y escalable. [33]

Se ha elegido el lenguaje Python principalmente por el estudio que se realizó anteriormente en la sección 4.2.2, debido al peso que suponían ambas librerías. Por lo que se ha decidido continuar usando dicho lenguaje y adaptar la base de datos. Pese a que este lenguaje no ha demostrado ser el más eficiente, sigue siendo muy versátil a la hora de trabajar. Con esto, se han seguido las buenas prácticas y se ha usado el *driver* oficial de MongoDB, llamado PyMongo, consiguiendo de manera nativa poder almacenar datos en documentos tipo JSON. Además, Python tiene extensas bibliotecas que procesan directamente los formatos de datos de tipo JSON y BSON. [33]

En cuanto a la plataforma como servicio que se ha elegido en este caso será MongoDB Atlas. Esto debido a que posee un plan gratuito, aunque con muchas limitaciones. La base de datos en la nube funcionaria de la misma manera que en la versión local, pero teniendo en cuenta las limitaciones técnicas.

Figura 5.2: Diagrama conceptual de la arquitectura, con la Capa de Base de Datos detallada.

5.2.2 *Modelado de datos*

El modelo de datos principal será compuesto por dos clases. Las dos clases harán referencia al mismo modelo, pero tendrán dos conceptos diferentes, uno para objetos nuevos y otro para su actualización. De esta manera tenemos dos clases, las cuales se llaman *Country* y *CountryUpdate*. La diferencia es que la primera clase creará un id nuevo para cada objeto y la segunda no.

Ambas poseen los mismo atributos y entidades. Están formadas por país y fecha, como los parámetros principales para poder clasificar el objeto y buscarlo en la base de datos. Luego está la lista de diferentes tendencias, la cual se va actualizando a lo largo del día.

Muchos de los atributos pueden llegar a ser opcionales, como por ejemplo la imagen del artículo, la cual si no existe se cargará otra por defecto desde la interfaz. En el siguiente diagrama se puede visualizar el conjunto del modelado, aunque las diferentes entidades serán explicadas en la capa de dominio o negocio.

Cabe mencionar el uso de la librería *pydantic*, la cual me ha facilitado la implementación de estas clases. Proporciona le lectura y escritura de los *BaseModel* y además, proporciona un decorador de clases de datos que crea (casi) clases de datos de Python estándar con análisis y validación de datos de entrada. Además, contiene funciones como la generación automática de archivos *JavaScript Object Notation*, «notación de objeto de JavaScript» (JSON). [34]

Figura 5.3: Diagrama de relación de la base de datos.

5.2.2.1 *Entidad Tendencia*

Esta entidad representa el modelado de datos de las historias de usuario cubiertas en el Sprint 2 (4.5.2). En ellas el usuario debe poder visualizar datos de diez tendencias como máximo.

Figura 5.4: Diagrama de relación de Tendencia.

Para poder representar debidamente la historia de usuario se debe ir calculando progresivamente los siguientes valores:

name: el nombre de la tendencia.
popularity: el objeto popularidad de la tendencia.
keywords: el objeto *keywords* de la tendencia.
sentiments: el objeto sentimientos general de la tendencia.
wrappeds: lista de objetos noticia de la tendencia.

5.2.2.2 *Entidad Popularidad*

Esta entidad representa el modelado de datos de las historias de usuario cubiertas en el Sprint 6 (4.5.6). En ellas el usuario debe poder visualizar datos de interés sobre la popularidad y también una gráfica de área.

Figura 5.5: Diagrama de relación de Popularidad.

Para poder representar debidamente las H.U. se debe ir calculando progresivamente los siguientes valores:

volume: el volumen de popularidad que está teniendo la tendencia.
hour_popularity: la hora en la que fue registrada el volumen.
avarage_popularity: popularidad media de la tendencia.
peak_popularity: popularidad más alta de la tendencia.

5.2.2.3 *Entidad Palabras más Comunes y Keywords*

Esta entidad representa el modelado de datos las historias de usuario cubiertas en el Sprint 7 (4.5.7). En ellas el usuario debe poder visualizar datos de interés sobre las palabras más comunes, las *keywords* y una gráfica de radial de barras correspondiente.

Figura 5.6: Diagrama de relación de Palabras más Comunes y Keywords.

Para poder representar debidamente las H.U. se debe ir calculando progresivamente los siguientes valores:

total: el volumen de publicaciones recogidas de la tendencia.
count: las repeticiones de palabras comunes.
label: las palabras comunes en cuestión.
words: las *keywords* extraídas.

5.2.2.4 *Entidad Sentimiento General*

Esta entidad representa el modelado de datos de las historias de usuario cubiertas en el Sprint 8 (4.5.8). En ellas el usuario debe poder visualizar datos de interés sobre el sentimiento general de los tweets o publicaciones y una gráfica de burbujas correspondiente.

Figura 5.7: Diagrama de relación de Sentimiento General.

Para poder representar debidamente las H.U. se debe ir calculando progresivamente los siguientes valores:

positive: datos sobre el sentimiento positivo.
negative: datos sobre el sentimiento negativo.
neural: datos sobre el sentimiento neutro.

5.2.2.5 *Entidad Noticas Relacionadas*

Esta entidad representa el modelado de datos las historias de usuario cubiertas en el Sprint 9 (4.5.9). En ellas el usuario debe poder visualizar datos de interés sobre las noticias o artículos relacionados sobre la tendencia.

Figura 5.8: Diagrama de relación de Noticas Relacionadas.

Para poder representar debidamente la H.U. se debe ir calculando progresivamente los siguientes valores:

date: fecha de publicación de la noticia.
image: imagen relacionada de la noticia.
summary: texto relacionado de la noticia.
title: título de la noticia.
url: enlace de la noticia.
source: editorial de la noticia.

5.3 CAPA DE DOMINIO O NEGOCIO

La capa de dominio o negocio permite la interacción entre la capa de base de datos y la capa de presentación. Es la capa encargada de gestionar tanto las distintas peticiones recibidas desde la capa de presentación, como los datos almacenados en la capa de base de datos. Comúnmente esta capa suele pertenecer a la parte Back-end del sistema.

5.3.1 *Estudio técnico*

Al plantear la aplicación en la capa de dominio o negocio, surgen ciertas pautas o normativas que conviene seguir. Surge la denominación *application programming interface*, «interfaz de programación de aplicaciones» (API) para las interacciones que surgen entre la capa de presentación y la capa actual. Otra denominación que conviene saber es *representational state transfer*, «transferencia de estado representacional» (REST), el cual es un conjunto de límites a la hora de implementar una API. [35]

Cuando un usuario envía una petición a través de una API REST o RESTful, esta transfiere una representación del estado del recurso solicitado al solicitante o al extremo. Generalmente la información se envía a través de *Hypertext Transfer Protocol*, «Protocolo de Transferencia de Hipertexto» (HTTP) en formato JSON, un lenguaje popular y fácilmente entendible. [35]

La mayoría de las interacciones se pueden resumir en el nemónico de las cuatro funciones del almacenamiento persistente llamado *Create, Read, Update and Delete*, «Crear, Leer, Actualizar y Borrar» (CRUD). Las funciones CRUD implican el uso de las operaciones HTTP, las cuales son PUT, GET, POST, DELETE o PATCH.

Habiendo definido propiamente las pautas de una API REST, podemos proceder a elegir el *microframework* que cumpla y revele las operaciones salientes mediante la interfaz REST y *websockets*. Para ello se ha elegido de nuevo el lenguaje Python, debido a que exista compatibilidad entre la capa de base de datos y la de dominio o negocio y además por su versatilidad a la hora de trabajar con diferentes librerías que serán extremadamente útiles para la realización de diversas historias de usuario.

Actualmente, en Python existen *frameworks* que funcionan con el estándar *Web Server Gateway Interface* (WSGI) o *Asynchronous Server Gateway Interface* (ASGI). El último mencionado es el más actual y utiliza corrutinas introducidas en las versiones más recientes de Python, lo que mejora el uso de la CPU en servidores web intensivos de E/S.

Las principales implementaciones de servidor ASGI existentes actuales son Uvicorn, Hypercorn y Daphne. Los tres son fácilmente implementables, pero el más estable es Uvicorn, además usa una implementación escrita directamente en el lenguaje C, lo cual lo vuelve más rápido y eficiente. [36]

Tras haber elegido el servidor, podemos continuar con el *microframework*. Actualmente existe una cantidad variada de ellos, como FastAPI, Starlette o Django, pero finalmente me he decantado por la primera opción. FastAPI es mucho más ligero que Django y además funciona de una manera parecida a Starlette, pero le añade funcionalidades como validación o documentación. [37]

La implementación de las rutas fue realizada con el objeto *router*, el cual tiene la función de declarar todas las rutas con decoradores y añadir el prefijo que llevaran dichas rutas. Posteriormente se crea la documentación automática.

Figura 5.9: Documentación generada por FastAPI.

En cuanto a la plataforma como servicio (PaaS) he elegido Heroku, la cual tiene opciones gratuitas y está basada en contenedores. La gran ventaja de Heroku, aparte de tener un plan gratuito, es que tiene múltiple soporte de lenguajes y entre ellos Python. Las rutas declaradas y la de por defecto funcionarían de la misma manera que localmente. [38]

Además, otra ventaja crucial, la cual es requerida para el correcto funcionamiento de la aplicación, es que un proceso se ejecute cada determinado intervalo de tiempo. Esto es debido a que necesitamos que se actualice la información de las tendencias, por lo que localmente tendríamos que configurar un *cron* ejecutando un *script* de Python y en Heroku tenemos un *Scheduler*, el cual tiene el mismo comportamiento. Dicho proceso ha sido configurado para que se ejecute cada hora y el *script* en cuestión recopilará toda la información necesaria.

Figura 5.10: Diagrama conceptual de la arquitectura, con la Capa de Dominio o Negocio y Base de Datos detallada.

5.3.2 *Diagramas de interacción de distintas rutas*

Figura 5.11: Diagrama interacción de rutas: *Actualización de Country*.

Figura 5.12: Diagrama interacción de rutas: *Creación de Country*.

Figura 5.13: Diagrama interacción de rutas: *Lectura de lista de Country*.

Figura 5.14: Diagrama interacción de rutas: *Lectura de Country por parámetros*.

Figura 5.15: Diagrama interacción de rutas: *Eliminación de Country por parámetros*.

Figura 5.16: Diagrama interacción de rutas: *Lectura de la búsqueda de Country*.

Figura 5.17: Diagrama interacción de rutas: *Actualización de Country por id*.

Figura 5.18: Diagrama interacción de rutas: *Eliminación de Country por id*.

5.4 CAPA DE PRESENTACIÓN

La capa de presentación se encarga de visualizar el contenido de la aplicación, donde el usuario puede interactuar directamente. De este modo esta capa será la única visible para el usuario. Comúnmente esta capa se suele denominar como Front-end.

5.4.1 *Estudio técnico*

Al buscar el *framework* idóneo para la visualización de la aplicación surgen muchas posibilidades. Una entre ellas fue Vue, esta opción destaca por su rendimiento, escalabilidad y buena documentación. A la vez es mucho más ligera que sus competidores y además ofreciendo las mismas características, como por ejemplo el uso del *Document Object Model* (DOM) virtual, el cual tiene sus ventajas a la hora de renderizar los mismos componentes de una página. En nuestro caso los componentes como las cartas se podrían beneficiar de ello. [39]

Uno de los principales objetivos de la aplicación, que ya se había planteado antes, es que el contenido se pueda visualizar por medio de módulos o cartas. Por lo que, al diseñar la propia aplicación, tendremos una vista principal y cada una de las cartas estará dividida en una subvista. Cada subvista tendrá sus propiedades, sus propias funciones y clases CSS. La mayoría de las clases estarán implementadas directamente a mano, menos las más básicas o convenientes ya que usaré el *framework* TailwindCSS para agilizar el trabajo.

En cuanto a la plataforma como servicio (PaaS) para la capa de presentación me he decantado por Vercel, realmente tenía mucho parecido a otras plataformas como Netlify, aunque tras usar ambas plataformas he podido concluir que Vercel funciona más eficientemente y de una manera más simple. Además, posee integración con Vue, lo cual me ha permitido exportar fácilmente el proyecto.

Figura 5.19: Diagrama conceptual de la arquitectura, con la Capa de Dominio o Negocio, Base de Datos y Presentación detallada.

El contenido de la vista principal se adapta mediante parámetros y rutas. Aunque existan diferentes rutas en la capa de presentación, todas ellas devuelven la misma vista principal con los mismos componentes, a no ser que haya un error. En el siguiente diagrama se explica el funcionamiento de las distintas rutas, tanto la ruta por defecto, la ruta de la búsqueda de un tópico o la ruta por parámetros.

- *Ejemplo de ruta por defecto:*
  https://(...)/
- *Ejemplo de ruta de búsqueda:*
  https://(...)/search?name=Spain&query=baloncesto
- *Ejemplo de ruta por parámetros:*
  https://(...)/Spain/22-10-24

Figura 5.20: Diagrama de actividad.

La ruta por defecto cargará como parámetro el país España y la fecha del día actual. Las demás rutas son ejecutadas al interactuar con los componentes de distintos formularios de la página. En este caso son dos, un formulario de selección (fecha y país) y otro formulario de búsqueda (los tópicos).

Figura 5.21: Formularios de selección y búsqueda.

Cada uno funcionan de una manera independiente, mediante eventos por click. Aunque si precisan de la información que hay en cada formulario, es decir, si el usuario cambia el parámetro país, entonces se carga una ruta con la fecha y el país seleccionado. Esto ocurre de la misma manera seleccionando la fecha o buscando un tópico, ambos necesitan la información del país seleccionado para procesar la ruta.

5.4.2 *Transcurso de la Interfaz de Usuario*

La vista principal es la única vista que se le llega mostrar al usuario, por lo que es importante mostrar el transcurso que ha tenido su propio diseño. Aunque como hemos explicado la vista principal se subdivide y cada subvista tiene sus propias clases de CSS, en español «Hojas de estilo en cascada». Cabe mencionar, que también se ha empleado una biblioteca que fue crucial llamada TailwindCSS.

Figura 5.22: Primer boceto de la aplicación.

Figura 5.23: Segundo boceto más detallado.

Figura 5.24: Concepto final de la aplicación.

Figura 5.25: Concepto del error en la visualización de parámetros.

5.4.3 *Vista Principal*

La vista principal contiene todas las subvistas y componentes necesarios para el funcionamiento de la interfaz. Cada subvista funciona como un componente esencial dentro de la vista principal. Al cargar correctamente los datos desde la capa de dominio o negocio, estos se representan en la vista principal y a la vez cada entidad se representa con una subvista diferente.

Figura 5.26: Diagrama de clase de las Vistas.

Cada subvista es llamada desde la vista principal, cada una con propiedades y funciones diferentes, ya que la representación de cada vista debe ser única y esencial.

La Vista principal representa la visualización de la historia de usuario que fue contemplada en el Sprint 2 (4.5.2) y esta relaciona con la entidad explicada en la sección 5.2.2.1. Esta vista es llamada *HomePage* y está formada por *components*, los cuales son las diferentes subvistas que se ha mencionado anteriormente. Los parámetros de las

distintas rutas están referenciados por *country*, *date*, *countryOptions* y *dateOptions*, además las funciones de esta vista tienen el objetivo de construir los distintos formularios y los datos de la tendencia se cargarán en *wrapped*.

También existen propiedades más complementarias como *num* y *rank*, para enumerar las tarjetas o cartas y el *ranking* de la tendencia, respectivamente. Las funciones de alerta o los distintos atributos como *loading*, existen para enriquecer las animaciones de la página y para mostrar el cuadro de dialogo que se ha mencionado anteriormente (5.25). Finalmente, la función de *encodeURL* codifica el nombre de la tendencia para redirigirlo a la plataforma Twitter mediante un enlace.

5.4.4 *Subvista Popularidad*

Esta subvista representa la visualización de las historias de usuario que fueron contempladas en el Sprint 6 (4.5.6). En ellas el usuario debe poder visualizar datos de interés sobre la popularidad y también una gráfica de área. La implementación de la gráfica se consigue mediante la librería ApexCharts.

Figura 5.27: Estructura y Diseño de la primera subvista.

Los diferentes valores que posee esta subvista están relacionados con la entidad Popularidad (5.2.2.2), a excepción de propiedades de la gráfica y atributos complementarios como *url*, para redirigir a la plataforma de Twitter. La función debe transformar el texto en formato *Date* de JavaScript.

5.4.5 *Subvista Palabras más Comunes y Keywords*

Esta subvista representa la visualización de las historias de usuario que fueron contempladas en el Sprint 7 (4.5.7). En ellas el usuario debe poder visualizar datos de interés sobre las palabras más comunes, las *keywords* y un gráfico de radial de barras correspondiente. La implementación de la gráfica se consigue mediante la librería ApexCharts.

Figura 5.28: Estructura y Diseño de la primera subvista.

Los diferentes valores que posee esta subvista están relacionados con la entidad explicada en la sección 5.2.2.3. La función adapta el formato a los porcentajes.

Esta subvista tuvo un diseño inicial que fue descartado por problemas de diseño y también por posible confusión a la hora de interpretar el gráfico. El diseño tuvo problemas a la hora de encajar el gráfico en su encuadre y no quedaba muy claro el porcentaje de uso de las distintas palabras.

Figura 5.29: Diseño descartado de la segunda subvista.

5.4.6 *Subvista Sentimiento General*

Esta subvista representa la visualización las historias de usuario que fueron contempladas en el Sprint 8 (4.5.8). En ellas el usuario debe poder visualizar datos de interés sobre el sentimiento general de los tweets o publicaciones y un gráfico de burbujas correspondiente. La implementación de la gráfica se consigue mediante la librería ApexCharts.

Figura 5.30: Estructura y Diseño de la tercera subvista.

Los diferentes valores que posee esta subvista están relacionados con la entidad explicada en la sección 5.2.2.4. La función calcula el porcentaje dependiendo del radio de las burbujas.

5.4.7 *Subvista Noticas Relacionadas*

Esta subvista representa la visualización las historias de usuario que fueron contempladas en el Sprint 9 (4.5.9). En ellas el usuario debe poder visualizar datos de interés sobre las noticias o artículos relacionados sobre la tendencia.

Figura 5.31: Estructura y Diseño de la última subvista.

Los diferentes valores que posee esta subvista están relacionados con la entidad explicada en la sección 5.2.2.5. La función devuelve una imagen por defecto si la carga de la imagen de la noticia falla.

Parte III

REALIZACIÓN

6 TERMINOLOGÍA DEVOPS

Al haber consolidado la metodología de desarrollo SCRUM en los capítulos anteriores (4.1), sabemos que dicha metodología pertenece a un ámbito de desarrollo iterativo e incremental, donde el proyecto se planifica en diversos bloques temporales. Debido a esto es muy frecuente que el equipo de desarrollo tenga numerosas entregas, ya que debe proporcionar una versión incrementada en cada iteración, esto también se suele denominar como despliegue.

Generalmente para proporcionar un despliegue el equipo de desarrollo se divide en dos ámbitos o departamentos. El primero tiene la función de desarrollo o la propia codificación de la aplicación, reconocido como «development» o «Dev» del inglés. El segundo se encarga de la parte operacional o el gestionado de versiones y la parte física del despliegue, reconocido también como «operations» o «Ops». [40]

El problema más usual es que ambos departamentos tienden a estar separados, por lo que el flujo de comunicación tiende a fallar y a tener retrasos a la hora de proporcionar el despliegue. Bajo esta premisa, surge una solución en 2009 denominada como *DevOps*, la cual tiene como propósito unificar estos dos ámbitos [40]

Figura 6.1: Imagen explicativa del concepto DevOps [41].

A groso modo, *DevOps* trata de romper estos silos organizativos establecidos y intenta centrarse en la mejora de calidad de la entrega al cliente. Para ello, *DevOps* se apoya en tres principios o también llamadas «the three ways», las cuales se deben implementar de forma consecutiva. [42]

1. *Mejora del flujo de entrega*: consiste en general valor al producto, mediante entregas del software de una forma temprana y periódica. Tratando de dividir los procesos en tareas más pequeñas o automatizando todas las tareas que sean posibles, con esto se

consigue un aumento del flujo de trabajo global, ya que en este caso un proceso no ralentiza a otro.
Un ejemplo de este principio puede ser la ejecución de tests u otras operaciones cada vez que haya cambios en el repositorio del código fuente.

2. *Buen uso de los ciclos de feedback*: consiste en el uso de la retroalimentación con el objetivo de optimizar los procesos, es decir, cada vez que se lleve a cabo una entrega conviene tener un *feedback* rápido y constante. Con esto se consigue que los ciclos de retroalimentación obtengan la información necesaria para que las correcciones se puedan aplicar de manera continua.
Un ejemplo de ello puede ser la monitorización del sistema y la comunicación ante la detección de errores.

3. *Asentar el aprendizaje y la mejora continua*: se centra en aplicar todo lo anteriormente realizado y aprendido para implementar un sistema de mejora continua. De esta manera se consigue innovar el producto, pero sin criminalizar los errores mediante el aprendizaje y las pruebas continuas.
Un ejemplo puede llegar a ser usar programas extremos de pruebas o prestaciones.

6.1 CONTROL DE VERSIONES

Una de las principales plataformas en sacar provecho de los flujos o vías que se han explicado anteriormente es GitHub. Aunque el objetivo principal de la plataforma no sea este, sino alojar proyectos mediante el control de versiones Git y la creación de código fuente. Debido a esto, a la hora de alojar el código fuente del proyecto, es usual llegar a incidir sobre las diferentes vías de *DevOps*. [43]

Mi aplicación también ha aprovechado esta plataforma y el uso de la herramienta Git. La principal ventaja a la hora de usar Git es la capacidad de controlar los cambios en el código fuente del proyecto y tener múltiples versiones en caso de posible retorno a una versión previa, un funcionamiento parecido a un historial. Por otro lado, la principal ventaja a la hora de usar GitHub es poder compartir el código con una comunidad y la integración continua en el repositorio ante cualquier cambio.

De esta manera, las plataformas de servicio (PaaS) como Heroku o Vercel pueden encargarse del despliegue continuo ante cualquier cambio del repositorio del proyecto y de la monitorización. Gracias a esto mejoramos el flujo de la entrega continuo y sus ciclos de *feedback*, ya que también se envía un correo ante un despliegue correcto o fallido.

6.2 IMPLEMENTACIÓN BACK-END

Para la gestión, virtualización y empaquetado de dependencias en Python del proyecto se ha elegido la herramienta *Poetry*. Esta herramienta nos facilitará la portabilidad de nuestro proyecto, ya que declara todas las dependencias y gestiona las actualizaciones en un archivo. Mediante este archivo se puede repetir y distribuir la instalación en diferentes máquinas. Además, dicho gestor nos ayudará a cumplir las diferentes vías de *DevOps*. [44]

Terminal:	*poetry install*

Tabla 6.1: Instalación del proyecto.

Terminal:	*poetry run python app/data/wrapped_extractor.py*

Tabla 6.2: Ejecución de *script* mediante *Poetry*.

Terminal:	*poetry run python backend/main.py*
	poetry run uvicorn backend.server:server --host=0.0.0.0 --port=8080

Tabla 6.3: Ejecución de servidor mediante *Poetry*.

La primera herramienta encargada de mejorar el flujo de la entrega se denomina *Mypy*, la cual es un verificador de tipo estático para Python 3. Con esta herramienta se verifica el código del proyecto para encontrar errores comunes, como por ejemplo advertencias sobre la conversión de una expresión a su tipo inferido o código que es inalcanzable o redundante, con el objetivo de conseguir buenas prácticas en el código. [45]

Terminal:	*poetry run mypy app backend*

Tabla 6.4: Ejecución de *mypy* mediante *Poetry*.

Además de la herramienta Mypy, se usa también *Flake8*, una herramienta para el seguimiento de una guía de estilo, corrección de errores de programación y para verificar la complejidad ciclomática. En mi caso, se usa *Flake8* para crear un estilo consistente, y mypy para hacer que el desarrollo sea menos propenso a crear errores. [46]

Terminal:	*poetry run flake8 app backend*

Tabla 6.5: Ejecución de *flake8* mediante *Poetry*.

Finalmente, la herramienta con mayor peso para la mejora del flujo de entrega es una que tenga como objetivo la comprobación del buen funcionamiento de la aplicación mediante *tests*. Si una aplicación no está *testeada* no podríamos asegurar su buen funcionamiento. En este caso se ha usado *Pytest*, recopilando en total 47 *tests* y consiguiendo una cobertura del 97 %. [47]

Terminal:	*poetry run pytest*

Tabla 6.6: Ejecución de *pytest* mediante *Poetry*.

Además, con dichas herramientas solo nos podemos asegurar del funcionamiento de nuestro proyecto en una sola versión de Python y un determinado sistema operativo. Para poder asegurar diferentes versiones, tanto del lenguaje Python como del sistema operativo, se ha utilizado la herramienta *Tox*. Dicha herramienta tiene el enfoque de asegurar pruebas automatizadas mediante el uso de entornos virtuales. De esta manera podemos asegurar el funcionamiento, tanto en Python 3.9 y 3.10 como en ubuntu y windows. [48]

Terminal:	*poetry run tox*

Tabla 6.7: Ejecución de *tox* mediante *Poetry*.

Todas estas herramientas tienen un papel determinado en la integración continua del proyecto y su mejora del flujo de entrega. Al cambiar el código del repositorio, se ejecutan dichas herramientas gracias a *GitHub Actions*, avisan al desarrollador mediante la plataforma de correo electrónico y se realiza un despliegue en la plataforma Heroku, consiguiendo una retroalimentación continua, asegurando un buen uso de los ciclos de *feedback* y mejorando la entrega continua del proyecto.

Figura 6.2: Esquema *DevOps* en el *Back-end*.

Cabe mencionar que cada herramienta tiene su propio archivo de configuración, al igual que *GitHub Actions*, el cual se tiene que configurar mediante un archivo *YAML Ain't Markup Language*, «YAML no es un lenguaje de marcado» (YAML) dentro de la carpeta *.github/workflows*.

6.3 IMPLEMENTACIÓN FRONT-END

Al igual que en la parte *Back-end*, en esta capa también se ha usado un sistema de gestión de paquetes. Se ha usado el gestor de paquetes por defecto que ofrece Node.js, un entorno de ejecución para JavaScript, llamado *npm*. Gracias a este gestor, podremos repetir y distribuir diferentes versiones del proyecto con el fin de cumplir las diferentes vías de *DevOps*. [49]

Terminal:	*npm install*

Tabla 6.8: Instalación del proyecto.

Terminal:	*npm run serve*

Tabla 6.9: Ejecución del proyecto.

Terminal:	*npm run build*

Tabla 6.10: Construcción del proyecto.

En cuanto a las herramientas utilizadas para cumplir con la integración continua del proyecto fueron *ESLint* y *Prettier*. En primer lugar, *ESLint* tiene como objetivo principal analizar el código y mostrar errores cometidos mediante reglas, las reglas que he escogido para ello son reglas predefinidas por el estándar de la compañía Airbnb, siendo este estándar un pilar dentro de la programación de JavaScript. [50]

Por otro lado, *Prettier* es un formateador de código obstinado, su función es ajustar el código mediante las reglas propuestas por el usuario. Ambas herramientas pueden llegar a coexistir e incluso beneficiarse de diferentes funciones que poseen.

Terminal:	*npm run lint*

Tabla 6.11: Ejecución del linter.

Con ambas herramientas se termina consiguiendo una mejora de flujo de entrega e integración continua, ya que estas herramientas

también son ejecutadas al realizar cambios y confirmaciones en el repositorio del código fuente. Además, se consigue un despliegue continuo gracias a la plataforma Vercel, ya que al igual que Heroku, también esta contactada con la plataforma de GitHub. Vercel también cumple con la retroalimentación continua, avisando por correo electrónico del estado del despliegue del proyecto.

Figura 6.3: Esquema *DevOps* en el *Front-end*.

7 IMPLEMENTACIÓN

Una vez habiendo establecido tanto la estructura como distintos conceptos de implementación, se puede empezar a incidir sobre diferentes implementaciones realizadas. El código fuente del proyecto, como he explicado en el capítulo anterior, está alojado en la plataforma de GitHub bajo el enlace de `https://github.com/nikitastetskiy/Wrapped`, el *Back-end* en la plataforma Heroku bajo el enlace de `https://wrappedtrends.herokuapp.com` y el *Front-end* en la plataforma Vercel bajo el enlace de `https://wrapped-trends.vercel.app`. Aún así, es conveniente analizar y explicar los diferentes códigos fuentes y algoritmos que se han implementado a lo largo del proyecto.

7.1 IMPLEMENTACIÓN BACK-END

La implementación de las capas pertenecientes a *Back-end* se localizan en los directorios *app* y *backend*. El primero, guarda todas las entidades y *scripts* necesarios para la obtención y gestión de datos de las tendencias. Mientras que el segundo guarda toda la estructura del servicio del servidor y el modelo perteneciente a la base de datos.

7.1.1 *Implementación Principal*

La primera implementación de código que se estudiará será la función principal encargada de gestionar las tendencias. Está denominada bajo el nombre de *main_operation*. Dicha función consigue las tendencias acordes al país y construye el diccionario apropiado, dependiendo de si existe dicho país en la base de datos o no. Si existe se tendrá que hacer una llamada de edición o PUT, en cambio si no existe se tendrá que crear y hacer una llamada POST. Además, se comprueba si existe un país con una antigüedad de más de 7 días para eliminarlo, con el objetivo de optimizar la base de datos.

En este primer caso, creo que es conveniente mostrar tanto el pseudocódigo como el código fuente de la función. En casos posteriores se alternará en base a la relevancia de la función, con el fin de optimizar la comprensión lectora.

Algoritmo 1: Operación Principal

```
Function Main
    Input: country
    trends ⟵ getTrends(country);
    if trends then
        today = datetime.now(country);
        yesterday = today - 1;
        oldCountry = requests.get(URI_ENDPOINT);
        if oldCountry then
            dict ⟵ createDictCountry(country, trends, today,
             yesterday, oldCountry);
        end
        else
            dict ⟵ createDictCountry(country, trends, today,
             yesterday);
        end
        if dict then
            if oldCountry then
                requests.put(URI_ENDPOINT, dict.json());
            end
            else
                requests.post(URI_ENDPOINT, dict.json());
                sevenDaysOldCountry =
                 requests.get(URI_ENDPOINT);
                if sevenDaysOldCountry then
                    requests.delete(URI_ENDPOINT);
                end
            end
        end
    end
end
```

La función tiene por *input country*, el cual es un diccionario con valores predefinidos que necesita cada país, es decir, valores como el nombre del país, el WOEID, el código del país, su lenguaje y su zona horaria. Otro valor predefinido es el *endpoint* de mi servidor, para realizar llamadas HTTP.

```
1 def main_operation(country):
2
3     sorted_trends = get_trends_with_country(country['woeid'])
4     if sorted_trends:
5
6         DATE = datetime.now(
7             timezone(country['timezone'])).strftime("%y-%m-%d")
8         YESTERDAY = (datetime.now(timezone(country['timezone']))
9                     - timedelta(days=2)).strftime("%Y-%m-%d")
```

```
10          resp = requests.get(f"{TRENDING_WRAPPED_ENDPOINT}
11                  {country['name']}/{DATE}")
12
13          # Checking if the object exists already
14          if resp.status_code == 200:
15              my_dict = create_dict_country(country,
16                          sorted_trends, DATE, YESTERDAY,
17                          json.loads(resp.content)["tendencias"])
18
19          else:
20              my_dict = create_dict_country(
21                  country, sorted_trends, DATE, YESTERDAY, False)
22
23          if my_dict:
24              my_dict_json = json.dumps(
25                  my_dict, default=lambda o: o.__dict__)
26              my_headers = {
27                  'accept': 'application/json',
28                  'Content-Type': 'application/json'
29              }
30              # Checking if the object exists already
31              if resp.status_code == 200:
32                  req = requests.put(
33                      TRENDING_WRAPPED_ENDPOINT, data=my_dict_json,
34                      headers=my_headers)
35              else:
36                  req = requests.post(
37                      TRENDING_WRAPPED_ENDPOINT, data=my_dict_json,
38                      headers=my_headers)
39                  OLD_DATE = (datetime.now(timezone(
40                              country['timezone'])) -
41                              timedelta(days=7)).
42                              strftime("%y-%m-%d")
43                  old_resp = requests.get(
44                      f"{TRENDING_WRAPPED_ENDPOINT}
45                      {country['name']}/{OLD_DATE}")
46                  # Checking for old objects
47                  if old_resp.status_code == 200:
48                      old_resp = requests.delete(
49                          f"{TRENDING_WRAPPED_ENDPOINT}
50                          {country['name']}/{OLD_DATE}")
```

Código 7.1: Operación principal

Para la creación de los diccionarios, el *endpoint* y el acceso a la API de Twitter se han creado unas constantes globales. Se ha propuesto tres países relevantes como ejemplos. En cuanto al acceso a la API de Twitter se puede hacer de varias maneras, pero las funciones y los recursos de la API cambian dependiendo de la manera que se autentifique en la API. En este caso se ha hecho con V1, pero también se ha comentado la otra posibilidad.

```
COUNTRIES = [
    {
        'name': 'Spain',
        'woeid': 23424950,
        'code': 'ES',
        'language': 'es',
        'timezone': 'Europe/Madrid'
    },
    {
        'name': 'United States',
        'woeid': 23424977,
        'code': 'US',
        'language': 'en',
        'timezone': 'US/Pacific'
    },
    {
        'name': 'Ukraine',
        'woeid': 23424976,
        'code': 'UA',
        'language': 'uk',
        'timezone': 'Europe/Kiev'
    }
]
```

Código 7.2: Declaraciones de paises

```
TRENDING_WRAPPED_ENDPOINT = "https://wrappedtrends.herokuapp.com/
    country/"
```

Código 7.3: Declaración de *endpoint*

Cada proyecto iniciado en la API de Twitter proporciona claves de acceso o secretos únicos que no deben ser compartidos con nadie. Estos son utilizados para acceder y autenticarse en la API.

```
# ENV VARIABLES OF TWITTER API
consumer_key = os.getenv('API_KEY')
consumer_secret = os.getenv('API_SECRET')
access_token = os.getenv('ACCESS_TOKEN')
access_token_secret = os.getenv('ACCESS_TOKEN_SECRET')

# TWITTER API
auth = tweepy.OAuthHandler(consumer_key, consumer_secret)
auth.set_access_token(access_token, access_token_secret)
api = tweepy.API(auth)

# client = tweepy.Client(bearer_token=BEARER_TOKEN)
```

Código 7.4: Autentificación de la API de Twitter

La operación principal se tendrá que ejecutar para cada país, lo cual es bastante costoso, debido a que cada país o valor del diccionario tendrá que estar esperando hasta la finalización del valor que haya por delante. Debido a esto he implementado una función que se ejecute mediante el multiprocesamiento. La función recorre los diferentes países de manera asíncrona, bajo la premisa del número máximo de procesos que se pueden utilizar para ejecutar las llamadas dadas, en el caso dado es 200. Si no se proporciona, se crearán tantos procesos de trabajo como procesadores tenga la máquina.

```
1  def main():
2
3      # Multiprocessing
4      executor = concurrent.futures.ProcessPoolExecutor(200)
5      futures = [executor.submit(main_operation, country) for
       country in COUNTRIES]
6      concurrent.futures.wait(futures)
7
8  if __name__ == "__main__":
9
10     main()
```

Código 7.5: Ejecución con multiprocesamiento

Por lo que algo que puede tardar minutos en ejecutarse de manera síncrona, lo hace en menos de un tercio de tiempo debido al multiprocesamiento, aunque la consumición de energía del procesador es mayor. En la siguiente imagen se pueden ver los resultados, la primera imagen será sin el multiprocesamiento y la segunda lo incluirá.

Figura 7.1: Comparación de la ejecución sin multiprocesamiento.

Aunque el mayor coste de la ejecución se basa en la descarga de diferentes artículos o noticias en base a la tendencia, esto es debido a que dependen del ancho de la red y su velocidad de descarga, la implementación de la función correspondiente se verá más adelante.

Figura 7.2: Comparación de la ejecución con multiprocesamiento.

7.1.2 *Implementación de la Generación del Diccionario*

La generación del diccionario a simple vista puede parecer sencilla, ya que solo tenemos que descargar los nombres de las tendencias y convertirlos en distintos objetos para que la base de datos pueda interpretarlos. Aunque esto se puede complicar debido a varios factores.

Primero, si se tiene un objeto ya creado en la base de datos, pues se tendrá que descargarlo y compararlo con el nuevo. Esto es debido a que las tendencias cambian cada hora y se debe tener en cuenta las tendencias viejas y sus distintos atributos. Por lo que se hará una unión entre las tendencias nuevas y las viejas, y posteriormente una diferencia simétrica entre las tendencias restantes con el objetivo de actualizar los atributos de tendencias viejas o incorporar tendencias completamente nuevas. Luego, se tiene que realizar distintas llamadas con distintos parámetros a la API de Twitter, con la intención de crear diferentes objetos funcionales y agregarlos al diccionario (objetos como la popularidad, los sentimientos o artículos).

Al descargar tendencias nuevas y compararlas con las viejas, se hará unión si hay tendencias bajo el mismo nombre. Esto es debido a que es necesario actualizar atributos como la popularidad, pero a la vez mantener el historial del volumen de publicaciones. Luego, la diferencia simétrica se refiere a las tendencias que no aparecen en la unión, ellas también estarán en cuenta a la hora de ordenar la lista de tendencias bajo el valor de la popularidad.

Algoritmo 2: Creación del Diccionario Country

```
Function createDictCountry
    Input: country, sortedTrends, date, yesterday, oldTrends
    Output: dictCountry
    sortedParsed ⟵ ∅;
    hour = datetime.now(country);
    cont = 0;
    if oldTrends then
        sortedParsed ⟵ parseTrend(sortedTrends) ∪
         oldTrends;
        sortedParsed ⟵ sortedParsed △ oldTrends;
    end
    else
        sortedParsed = parseTrend(sortedTrends)
    end
    while cont ≠ 10 do
        tweets ⟵ api.searchTweets(sortedParsed_cont);
        keywords ⟵ searchKeywords(tweets);
        sentiments ⟵ searchSentiments(tweets);
        articles ⟵ searchArticles(tweets);
        cont++;
    end
    dictCountry = dictCountry(keywords, sentiments,
     articles);
    return dictCountry;
end
```

Habiendo entendido el pseudocódigo podemos comprender de una manera más sencilla el código fuente. Primero se comprueba si existen tendencias viejas, después se procedería a asignar una estructura constituyente a dichas tendencias mediante la unión y la diferencia simétrica. Además, sólo se procede al final con 20 tendencias y se calcula su popularidad, esto es debido a la optimización de la base de datos y su gestión. Posteriormente se calculan los demás valores (keywords, sentimientos y artículos) sobre la mitad de las tendencias clasificadas, es decir, un top 10. Las otras diez tendencias, que no llegaron a clasificarse, sirven como referencia posteriormente a la hora de pintar el gráfico de la popularidad, ya que esa información sigue siendo relevante, como historial, si la tendencia llega a ser mostrada.

```
1 if old_trends:
2     for item in sorted_trends:
3         t = parse_trend(item['name'], item['tweet_volume'],
4         hour, None)
5         for k, old_item in enumerate(old_trends):
6             if old_item["name"] == item["name"]:
7                 t.popularity.refresh(old_item["popularity"]
```

```
8                           ["volume"],old_item["popularity"]
9                           ["hour_popularity"])
10                  old_trends.pop(k)
11                  break
12          sorted_parsed.append(t)
13      for i in old_trends:
14          sorted_parsed.append(Trend(**i))
15
16  # When there aren't any new trends on the platfrom yet
17  else:
18      for item in sorted_trends:
19          t = parse_trend(item['name'], item['tweet_volume'],
20              hour, None)
21          sorted_parsed.append(t)
22
23  sorted_parsed = comparing_names(sorted_parsed)
24  sorted_parsed = sorted(sorted_parsed, key=lambda trend:
25      trend.popularity.peak_popularity, reverse=True)[:20]
```

Código 7.6: Creación de la estructura de tendencias

Una vez teniendo la estructura y los objetos de las tendencias, podemos proceder a calcular los demás valores, donde necesitaremos descargar como máximo cien publicaciones o tweets relacionadas con la tendencia. Primero se hace una llamada a la API con determinado valor hacia el ámbito popular. Se filtran las publicaciones repetidas y se intenta que sean publicaciones populares, si no se consiguen se opta por publicaciones menos famosas. Además, sólo se procede a asignar los valores una vez habiendo calculado todos ellos, para que solo existan objetos completos a la hora de visualizarlos.

```
1  wts = 0
2  while wts < 10 and cont < len(sorted_parsed):
3      name = sorted_parsed[cont].name
4      tweets = api.search_tweets(q=f'{name} -filter:retweets
5              min_faves:20 since:{yesterday}', lang=country
6              ["language"], count=100, result_type="mixed",
7              tweet_mode="extended")
8      if (len(tweets)) < 50:
9          tweets = api.search_tweets(q=f'{name} -filter:retweets
10             since:{yesterday}', lang=country["language"],
11             count=100, result_type="mixed",
12             tweet_mode="extended")
13     t = get_tweets(tweets, country["language"])
14     if t:
15         name = more_specific_name(t.words, name)
16         w = get_wrappeds(name, country["code"],
17             country["language"])
18         if w:
19             s = get_sentiments(tweets, country["language"],
20                 t.label)
```

```
21            if s:
22                sorted_parsed[cont].wrappeds = w
23                sorted_parsed[cont].keywords = t
24                sorted_parsed[cont].sentiments = s
25                wts += 1
26    cont += 1
27
28 dict_country = {
29    'pais': country["name"],
30    'dia': date,
31    'tendencias': sorted_parsed
32 }
33
34 return dict_country
```

Código 7.7: Creación de la estructura constituyente

La búsqueda de un tópico funcionaría de la misma manera, pero solo consiguiendo un solo objeto completo. También se limita el cálculo de la popularidad, ya que esa información proviene de las tendencias de la API de Twitter. Por lo que las únicas diferencias a la hora de implementarlo serían las funciones de estructuración.

```
1  # PARSING OBJECT TREND
2  def parse_trend(name, vol, time, array):
3
4      popularity = parse_popularity(vol, time)
5
6      values = {
7          'name': name,
8          'popularity': popularity,
9          'wrappeds': array
10     }
11
12     return Trend(**values)
13
14 # PARSING OBJECT SINGLE TREND
15 def parse_single_trend(name, array):
16
17     values = {
18         'name': name,
19         'popularity': None,
20         'wrappeds': array
21     }
22
23     return Trend(**values)
```

Código 7.8: Estructuración de la tendencia

A la hora de descargar las tendencias de Twitter, se tiene que comparar los nombres de distintas tendencias tal como se ha comentado en las tareas de la Historia de Usuario del Sprint 2 (4.5.2). Para ello se

comprueban si hay parecidos en los nombres, sin contemplar las mayúsculas, minúsculas y tildes. Por lo que para ello se han creado dos funciones, una normaliza una oración (quitando tildes, mayúsculas y caracteres de puntuación) y la otra recorre las distintas tendencias en busca de parecidos.

```
1  def normalize_str(c):
2      s = string.punctuation + " "
3      return c.translate(str.maketrans("áéíóú","aeiou", s)).lower()
4
5  def comparing_names(s):
6      i = 0
7      while i < (len(s)-1):
8          j = i+1
9          while j < (len(s)):
10             if normalize_str(s[j].name) in normalize_str(s[i].
11             name) or normalize_str(s[i].name) in
12             normalize_str(s[j].name):
13                 if len((s[i].name).split()) > len((s[j].name).
14                 split()):
15                     s.pop(j)
16                     j -= 1
17                 elif len((s[i].name).split()) <
18                 len((s[j].name).split()):
19                     s.pop(i)
20                     i -= 1
21                     break
22                 else:
23                     if s[j].popularity.peak_popularity >= s[i].
24                     popularity.peak_popularity:
25                         s.pop(i)
26                         i -= 1
27                         break
28                     else:
29                         s.pop(j)
30                         j -= 1
31             j += 1
32         i += 1
33
34     return s
```

Código 7.9: Comprobación de los nombres de tendencia

7.1.3 *Implementación de la Popularidad*

La entidad Popularidad está compuesta por distintos valores comentados en la sección 5.2.2.2. Dentro de la entidad Popularidad está la función que actualiza los valores y se calculan con una librería popular orientada hacia campo matemático llamada *numpy*.

```
# PARSING OBJECT POPULARITY
def parse_popularity(vol, time):

    values = {
        'volume': [vol],
        'hour_popularity': [time],
        'avarage_popularity': vol,
        'peak_popularity': vol,
    }

    return Popularity(**values)
```

Código 7.10: Estructuración de la popularidad

```
def refresh(self, vol, hour):
    self.volume = vol + self.volume
    self.hour_popularity = hour + self.hour_popularity
    self.avarage_popularity = int(np.average(self.volume))
    self.peak_popularity = int(np.max(self.volume))
```

Código 7.11: Actualización de la popularidad

7.1.4 *Implementación de las Keywords*

La entidad *Keyword* está compuesta por distintos valores comentados en la sección 5.2.2.3.

```
# PARSING OBJECT KEYWORD
def parse_keyword(k):

    values = {
        'total': k['total'],
        'count': k['count'],
        'label': k['label'],
        'words': k['words'],
    }

    return Keyword(**values)

def get_tweets(tweets, lng):

    t = fetch_tweets(tweets, lng)
    if t:
        t = parse_keyword(t)
        return t
    else:
        return False
```

Código 7.12: Estructuración de las keywords

A la hora de extraer las *keywords*, de los distintos tweets o publicaciones descargados de la API de Twitter, se deben limpiar tal y como se comentó en el Sprint 7 (4.5.7). Para ello se implementaron varias versiones para comprobar la eficiencia de cada una de ellas. Primero un sistema regex, luego el sistema regex de numpy y por último funciones de sintaxis de Python. Finalmente, las más eficientes fueron las expresiones regex o expresiones regulares. La siguiente captura de código fue para comprobar los distintos tiempos de ejecución, muchos de ellos fueron similares, pero las expresiones regex siempre tuvieron resultados ventajosos y que a la larga resultarían mucho más beneficiosos en tiempos de cómputo.

```
def clean1(text):
    text = text.lower()
    # Removes all mentions (@username) from the tweet since it is
     of no use to us
    text = re.sub(r'@[\w]*', '', text)
    # Removes any link in the text
    text = re. sub(r'http://\S+ https://S+', '', text)
    # Only considers the part of the string with char between a
    to z or digits and whitespace characters
    # Basically removes punctuation
    text = re.sub(r'[^\w\s]', '', text)

def remove_pattern(input_txt, pattern):
    r = re.findall(pattern, input_txt)
    for i in r:
        input_txt = re.subli, '', input_txt)
    return input_txt

def clean2(tweets):
    # remove twitter handles (xxx)
    tweets = np.vectorize(remove_pattern)(tweets, "@[\w] *")
    # remove URL links (httpxxx)
    tweets = np.vectorize(remove pattern)(tweets,
            "https?://(A-Za-z0-9./1*")
    # remove special characters, numbers, punctuations
    tweets = np.core.defchararray.replace(tweets,
            "[^a-zA-Z]", " ")
    return tweets

def clean3(tweet):
    # Removing twitter handles which start with @ or #  ' or http
    tweet = ''.join(word for word in tweet.split(' ') if not
            (word.startswith('#') or word.startswith('@') or
            word.startswith("'") or word.startswith('http'))
    tweet = tweet.translate(str.maketrans('áéíóú',
            'aeiou')).lower()
    return(tweet)
```

Código 7.13: Limpieza de las publicaciones

Al final se estableció las expresiones regex como función definitiva y se procedió a implementar la manera de limpiar las palabras vacías o *stop_words*. La eliminación de palabras vacías solamente se utiliza en el recuento de frecuencia de palabras.

```
def load_stop_words(file):
    """
    Utility function to load stop words from a file and return as
     a list of words
    """
    with open(file, 'r+') as f:
        stop_words = f.read().splitlines()
    return set(stop_words)

def simple_clean(text):
    text = text.translate(str.maketrans('áéíóú', 'aeiou')).lower
    ()
    # Removes all mentions (@username) or hashtags from the tweet
     since it is of no use to us
    text = re.sub(r'@[\w]*|#[\w]*', '', text)
    # Removes any link in the text
    text = re.sub(r'http://\S+|https://\S+', '', text)
    # Removes any \n
    text = re.sub(r'\n', ' ', text)

    return text

def clean(text, lng):
    # Basically removes punctuation
    text = re.sub(r'[^\w\s]', '', text)
    # Removes stop words that have no use in sentiment analysis
    stopwords = load_stop_words(f"app/data/ml_stopwords/my-{lng}-
    stopwords.txt")
    text = " ".join([word for word in text.split() if word
            not in stopwords])

    return text
```

Código 7.14: Limpieza de las publicaciones implementada

Las dos funciones de limpieza existen porque una se usa en el recuento de palabras y la otra en la propia extracción de *keywords*, la extracción de *keywords* o palabras clave se hará mediante un algoritmo de extracción. Para ello se han estudiado varios algoritmos, entre ellos YAKE, *Rapid Automatic Keyword Extraction* (RAKE) y las herramientas de *Natural Language Toolkit* (NLTK). El más eficiente y eficaz resultó ser YAKE, primero quedó descartado NLTK ya que debía ser descargado a la hora de ejecución, por lo que tardaba más. RAKE en cambio, solía tardar casi el mismo tiempo en ejecutarse, aunque su principal problema es que no reconocía distintas entidades como nombres o

acrónimos («CEO» ó «Google»), algo fundamental en los tweets, por lo que muchas *keywords* extraídas no contenían mucha relevancia.

Otra de las ventajas de YAKE es que se puede configurar el idioma, la limitación de duplicación de claves o la cantidad de palabras en una *keyword*. El sistema de YAKE se rige por seis componentes: preprocesamiento de texto, extracción de características, puntaje de términos individuales, generación de listas de palabras clave candidatas, reduplicación de datos y clasificación. A continuación se explicarán las distintas etapas del algoritmo, pero se recomienda la lectura del libro *YAKE! Collection-Independent Automatic Keyword Extractor* para su completo entendimiento. [51]

Primero, se aplica una etapa de preprocesamiento, la cual divide el texto en términos individuales cada vez que se encuentra un espacio vacío o un carácter especial (por ejemplo, saltos de línea, corchetes, coma, punto, etc.). En segundo lugar, se diseñan un conjunto de cinco rasgos para capturar las características de cada término individual. Estos son: *Casing*, *Word Positional*, *Word Frequency*, *Word Relatedness to Context* y *Word DifSentence*.

El primero, refleja el aspecto de *casing* de una palabra (capitalización en castellano). *Word Positional* valora más aquellas palabras que aparecen al principio de un documento basándose en la suposición de que las palabras clave relevantes a menudo tienden a concentrarse más al principio de un documento. *Word Frequency* indica la frecuencia de la palabra, puntuando más aquellas palabras que aparecen con mayor frecuencia. La cuarta función, *Word Relatedness to Context*, calcula el número de términos diferentes que aparecen delante o atrás de la palabra candidata. Cuanto mayor sea el número de términos diferentes que coexisten con la palabra candidata (en ambos lados), es probable que la palabra candidata carezca más de sentido. Finalmente, *Word DifSentence* cuantifica con qué frecuencia aparece una palabra candidata dentro de diferentes oraciones.

En el tercer paso, se combina heurísticamente todas estas características en una sola medida, de modo que a cada término se le asigna una puntuación S(w). Este peso dará lugar al proceso de generación de palabras clave del cuarto paso. Se considera una ventana deslizante de 3 gramos, generando así una secuencia contigua de palabras clave candidatas de 1, 2 y 3-gramos. A cada palabra clave candidata se le asignará una S(kw) final, de modo que cuanto menor sea la puntuación, más significativa será la palabra clave. De esta manera se formaliza la siguiente ecuación.

$$S(kw) = \frac{\prod_{w \in kw} S(w)}{TF(kw) * (1 + \sum_{w \in kw} S(w))} \tag{7.1}$$

S(kw) es la puntuación de una palabra clave candidata, determinada multiplicando (en el numerador) la puntuación S(w) del primer término de la palabra clave candidata por las puntuaciones posteriores de los términos restantes. Esto se divide por la suma de las puntuaciones de S(w) para promediar la longitud de la palabra clave, de modo que los n-gramos más largos no se beneficien solo porque tienen una n más alta. El resultado se divide aún más por TF(kw) (frecuencia del término de la palabra clave) para penalizar a los candidatos menos frecuentes. En el quinto paso, se eliminan candidatos similares provenientes de los pasos anteriores. Para ello se usa la *distancia de Levenshtein* [52]. Finalmente, el sistema generará una lista de palabras clave relevantes, formada por 1, 2, 3-gramos, de modo que cuanto menor sea la puntuación de S(kw), más importante será la palabra clave.

En el siguiente código se puede ver la implementación de la librería YAKE y el recuento de palabras relevantes. Además las *keywords* extraídas pasan por un filtro, comprobando las repeticiones y un umbral mínimo de S(kw). Cabe señalar que estas funciones poseen un manejo de errores que no se muestran para facilitar la lectura.

```
1  texto = ''
2  for tweet in query: # Cleans the tweet
3      clean_txt = simple_clean(tweet.full_text)
4      texto += clean_txt + ' '
5  keywords = (clean(texto, lng).split())
6  counter = Counter(keywords)
7  data: dict[str, Any] = {
8      'total': len(query),
9      'count': [],
10     'label': [],
11     'words': []
12 }
13 for x, y in counter.most_common(6): # Words Counter
14     data['label'].append(x)
15     data['count'].append(round((y/len(query)) * 100))
16
17 # YAKE Extraction
18 stopwords = load_stop_words(f"app/data/ml_stopwords/
19             my-{lng}-stopwords.txt")
20 custom_kw_extractor = yake.KeywordExtractor(
21     lan=lng, n=3, dedupLim=0.6, dedupFunc="seqm",
22     top=15, stopwords=stopwords, windowsSize=1)
23 yake_keywords = custom_kw_extractor.extract_keywords(texto)
24 yake_keywords = comparing_names(yake_keywords)
25 for k in yake_keywords:
26     data['words'].append(k)
27
28 return data
```

Código 7.15: Extracción de recuento de palabras y *keywords*

El algoritmo YAKE se rige por parámetros como la función de duplicación o la cantidad de *keywords* extraídas. Estos parámetros fueron seleccionados cuidadosamente, para que finalmente solo se muestren tres *keywords*. En un principio se extraen quince, pero al pasarlos por el filtro se quedarían en tres como máximo. En cambio, el contador de palabras es seis como máximo. El filtro intenta guardar las palabras claves más relevantes, por lo que elimina las repetidas.

Algoritmo 3: Filtrado de *Keywords*

```
Function comparingNames
    Input: setTuple
    Output: left(setOut, 3)
    setOut ⟵ ∅;
    // Filtrado de palabras repetidas en una keyword
    for s ∈ setTuple do
        words = s[0].split();
        setOut ⟵ set(words).toString();
        // Umbral de S(kw)
        if s[1] >0.05 then
            break;
        end
    end
    // Filtrado keywords similares o repetidas
    i = 0;
    while i <setOut.size()-1 and setOut.size() >3 do
        j = i+1;
        while j <setOut.size() do
            if setOut[j] ∈ setOut[i] then
                if setOut[i].length() ≥ setOut[j].length() then
                    setOut.pop(j);
                end
                else
                    setOut[i] = setOut[j];
                    setOut.pop(j);
                    i--;
                    break;
                end
                j--;
            end
            j++;
        end
        i++;
    end
    return left(setOut, 3);
end
```

El motivo de la implementación del filtro fue debido a que al usar los parámetros de extracción de YAKE los resultados no fueron del todo precisos. Al usar un limitador de duplicación muy alto, las palabras terminaban siendo poco coherentes. Mientras que al usar un limitador de duplicación muy bajo, todas las palabras claves resultaban ser muy parecidas o iguales. Por lo que la solución que he propuesto es extraer muchas palabras claves bajo un limitador de duplicación medio (0.6), donde existen palabras con poca relevancia y palabras coherentes muy similares entre sí.

De modo que al aplicar el filtro, las palabras poco coherentes quedarían eliminadas (palabras con S(kw) más grandes que 0.05). Al igual que las palabras más coherentes, serán seleccionadas por su mayor longitud y menor S(kw).

```
def comparing_names(s):
    i = 0
    s_out = []

    for item in s:
        unique_words = item[0].split()
        s_out.append((" ".join(sorted(set(unique_words),
            key=unique_words.index))))
        if item[1] > 0.05:
            break

    while i < (len(s_out)-1) and len(s_out) > 3:
        j = i+1
        while j < (len(s_out)):
            if any(word in s_out[j] for word in
            s_out[i].split()):
                if len((s_out[i]).split()) >=
                len((s_out[j]).split()):
                    s_out.pop(j)
                else:
                    s_out[i] = s_out[j]
                    s_out.pop(j)
                    i -= 1
                    break
                j -= 1
            j += 1
        i += 1

    return s_out[:3]
```

Código 7.16: Filtrado de las *keywords*

De esta manera, la tendencia *Joji* tendría las siguientes *keywords* extraídas con YAKE, previas al filtrado:

[('joji', 0.0023616958679190565), ('album', 0.0207...), ('amo joji', 0.0218...), ('nuevo album', 0.0228...), ('die', 0.0279...), ('nuevo disco', 0.0295...), ('amo', 0.0321...), ('album de joji', 0.0361...), ('you', 0.0433...), ('die for you', 0.0459...), ('goes hard tbh', 0.0598...), ('escuchar', 0.0608...), ('album nuevo', 0.0686...), ('joji freakin goes', 0.0730...), ('cancion', 0.0766...)].

Luego, al pasarlas por el filtrado de palabras obtendríamos las siguientes tres *keywords*: *['album de joji', 'die for you', 'nuevo disco']*. Las cuales tienen mucho más sentido y contexto que las tres primeras de la extracción de YAKE. En este caso *nuevo album* no se ha contemplado, debido a que la primera *keyword* contiene la clave *album*. Cabe aclarar, que el nombre del álbum es *die for you*.

7.1.5 *Implementación de los Sentimientos*

La entidad de los sentimientos está compuesta por distintos valores comentados en la sección 5.2.2.4. Para su funcionamiento se ha buscado un analizador léxico enfocado a las redes sociales. Se ha descartado el uso de *Natural Language Processing*, «procesamiento de lenguajes naturales» (NLP) como la librería *TextBlob*, debido a que funcionan por medio de *datasets* por lo que iba a ser más complicado adaptarlos a otros lenguajes y además no funcionarían igual de rápido que otros analizadores de texto estático.

Durante el estudio de diferentes analizadores léxicos, el que mejor resultado obtenía generalmente era el analizador VADER. Las ventajas de VADER al ser estático es que maneja bien las distintas negaciones o interpretaciones que pueda hacer sobre emoticonos y expresiones en las redes sociales. [53]

```
1  def parse_sentiment(s):
2
3      values = {
4          'positive': s['positive'],
5          'negative': s['negative'],
6          'neutral': s['neutral'],
7      }
8      return Sentiment(**values)
9
10 def get_sentiments(tweets, lng, kw):
11
12     s = fetch_sentiments(tweets, lng, kw)
13     if s:
14         s = parse_sentiment(s)
15         return s
16     else:
17         return False
```

Código 7.17: Estructuración de los sentimientos

El proyecto al funcionar con varios idiomas tiene que poseer diferentes léxicos con sus respectivas puntuaciones, para ello se puede usar léxicos definidos o traducir cada publicación al inglés y usar el léxico proveniente de VADER. Claramente, la segunda opción iba a ser más costosa y con muchas más limitaciones, por lo que se ha decidido traducir el léxico de VADER y configurarlo a los países definidos anteriormente, en este caso, el español y ucraniano. Obviamente al estar traducido automáticamente, se ha intervenido manualmente para configurar los distintos tiempos verbales o conjugaciones en femenino y masculino. También se han añadido léxicos externos para complementar el conjunto.

De esta manera, se ha adaptado la librería VADER desde su repositorio de GitHub [54] a las necesidades del proyecto. Se ha traducido el léxico inglés al español y ucraniano. Se han añadido varios léxicos más y sus conjugaciones. También se han traducido y añadido manualmente distintas negaciones, diminutivos y aumentativos correspondientes a cada idioma. [55, 56]

El léxico de sentimientos de VADER es sensible tanto a la polaridad como a la intensidad de los sentimientos expresados en los contextos de las redes sociales, y también es generalmente aplicable al análisis de sentimientos en otros dominios.

El léxico en inglés obtuvo calificaciones sobre los sentimientos de 10 evaluadores humanos independientes. Se clasificaron alrededor de 7500 rasgos de tokens en una escala de «[–4] Extremadamente negativo» a «[4] Extremadamente positivo», con margen para «[0] Neutral (Ninguno o N/A)». Dichos rasgos o características con puntuaciones válidas indicaban tanto la polaridad del sentimiento (positivo/negativo) como la intensidad del sentimiento en una escala de -4 a +4. Por ejemplo, la palabra «bien» tiene una valencia positiva de 0,9, «bueno» es 1,9 y «excelente» es 3,1, mientras que «horrible» es -2,5, el *emoji* con el ceño fruncido :(es -2,2. [54]

El puntaje de sentimiento de una oración se calcula sumando los puntajes de sentimiento de cada palabra incluida en el diccionario o el léxico de VADER. Las palabras individuales tienen una puntuación de sentimientos entre -4 y 4, pero la puntuación de sentimientos devuelta de una oración está entre -1 y 1. Esto es debido, a que la puntuación de sentimiento de una oración es la suma de la puntuación de sentimiento de cada palabra portadora de sentimiento. Debido a ello, se aplica una normalización al total para asignar a un valor entre -1 y 1. La normalización utilizada es: [57]

$$\frac{x}{\sqrt{x^2 + \alpha}} \tag{7.2}$$

En ella, x es la suma de los puntajes de sentimientos de las palabras constituyentes de la oración y alfa es un parámetro de normalización que se ha establecido como 15. La normalización se representa gráficamente a continuación.

Figura 7.3: Normalización utilizada con alfa como 15.

Se puede observar que a medida que x crece, se acerca cada vez más a -1 o 1. Con un efecto similar, al usar muchas palabras en el sentimiento de VADER se obtendrá una puntuación cercana a -1 o 1. Por lo tanto, el análisis de sentimientos de VADER funciona mejor en documentos cortos, como tweets o oraciones, no en documentos grandes.

Las características léxicas no es lo único que afecta al sentimiento de la oración. Existen otros elementos contextuales, como la puntuación, las mayúsculas y los modificadores que también imparten emoción. El análisis de sentimientos de VADER los tiene en cuenta al considerar cinco heurísticas simples. El efecto de estas heurísticas se cuantifica, nuevamente, utilizando evaluadores humanos.

La primera heurística es la puntuación gramatical. Donde la oración «Me gusta». expresa menos puntaje que «¡¡¡me gusta!!!», esto es debido a que VADER tiene en cuenta la cantidad de signos de exclamación y de interrogación que terminan la oración. VADER primero calcula la puntuación de sentimiento de la oración. Si la puntuación es positiva, VADER añade una cierta cantidad obtenida empíricamente por cada signo de exclamación (0,292) y signo de interrogación (0,18). Si la puntuación es negativa, entonces resta.

La segunda heurística es la capitalización. «INCREÍBLE actuación» es definitivamente más intenso que «increíble actuación». Por ello, se aumenta o disminuye el puntaje de sentimiento de la palabra en 0.733, dependiendo de si la palabra es positiva o negativa, respectivamente.

Esta característica no se ha implementado en mi proyecto, debido a que muchos tweets llevaban mayúsculas exclusivamente para llamar la atención y no para expresar sentimientos.

La tercera heurística es el uso de modificadores de grado. Por ejemplo, «muy lindo» y «más o menos lindo». El efecto del modificador en la primera oración es aumentar la intensidad de lindo, mientras que en la segunda oración es disminuir la intensidad. VADER mantiene un diccionario de refuerzo que contiene un conjunto de refuerzos y amortiguadores. Además, el efecto del modificador de grado también depende de su distancia a la palabra que está modificando. Las palabras más alejadas tienen un efecto intensificador relativamente menor sobre la palabra base. Un modificador al lado de la palabra suma o resta 0.293 al puntaje de sentimiento de la oración. Un segundo modificador de la palabra base suma o resta el 95 % de 0,293, y un tercero suma/resta el 90 %.

La cuarta heurística es el cambio de polaridad debido al «pero». A menudo, «pero» conecta dos cláusulas con sentimientos contrastantes. El sentimiento dominante, sin embargo, es el último. Por ejemplo, «Te amo, pero ya no quiero estar contigo». La primera frase «te amo» es positiva, pero la segunda «ya no quiero estar contigo». es negativo y obviamente más dominante en cuanto al sentimiento. VADER implementa un verificador de «pero». Básicamente, todas las palabras sentimentales antes del «pero» tienen su valor reducido al 50 %, mientras que las que están después del «pero» aumentan al 150 %.

La quinta heurística es examinar el trigrama antes de una característica léxica cargada de sentimientos para captar la negación de la polaridad. Aquí, un trigrama se refiere a un conjunto de tres características léxicas. VADER mantiene una lista de palabras negativas. La negación se captura multiplicando la puntuación de sentimiento de la característica léxica cargada de sentimiento por un valor determinado empíricamente -0,74.

Debido a todas estas características, la traducción del léxico inglés se ha realizado de una manera cuidadosa y selectiva (dejando las mismas puntuaciones que el léxico original), aunque se da por sentado que el léxico original tiene mayor capacidad de extraer los sentimientos. Sin embargo, la opción de traducir cada publicación iba a resultar mucho más costosa a la hora de ejecutarse. Primero la gran latencia de las múltiples traducciones que se puedan realizar y luego el limite de llamadas gratuitas diarias que claramente se van a traspasar.

```
1 def fetch_sentiments(query, lng, kw):
2
3     if lng == 'es':
4         analyzer = SentimentIntensityAnalyzerES()
5     elif lng == 'uk':
```

```
6           analyzer = SentimentIntensityAnalyzerUK()
7       else:
8           analyzer = SentimentIntensityAnalyzerEN()
9
10      data: dict[str, list[list[float]]] = {
11          'negative': [],
12          'neutral': [],
13          'positive': [],
14      }
15
16      try:
17          if query:
18              ppt = []
19              for tweet in query:
20                  clean_txt = simple_clean(tweet.full_text)
21                  vader = round((analyzer.polarity_scores(clean_txt
22                          ))['compound'], 4)
23                  y_axis = topic_relation(clean(clean_txt, lng),kw)
24                  popularity = ((tweet.retweet_count +
25                          tweet.favorite_count)*0.5)
26                  ppt.append(popularity)
27                  data = parse_data(data, vader, y_axis,popularity)
...
46          return data
47      except Exception as e:
48
49          print(e)
50          return False
```

Código 7.18: Extracción de los sentimientos

Como se ha explicado en la Historia de Usuario 11.1 (4.5.8) los sentimientos deberán estar representados en un gráfico. Por lo que los resultados proporcionados por VADER pertenecerán al Eje X y la relación con las palabras clasificadas anteriormente será el Eje Y. Además, cada burbuja tiene un radio distinto, esto estará calculado conforme a la popularidad, es decir, si supera la popularidad media de la tendencia tendrá un radio más grande, pero también aumenta su radio si las coordenadas de las burbujas están muy cerca, para que no haya demasiadas repeticiones.

```
1  def parse_data(data, vader, y, ppt):
2      if vader > 0.35:
3          data['positive'].append([vader, y, ppt])
4      elif vader < -0.35:
5          data['negative'].append([vader, y, ppt])
6      else:
7          data['neutral'].append([vader, y, ppt])
8
9      return data
10
```

```
11 def topic_relation(text, kw):
12     suma = 0
13     if len(text.split()):
14         for k in kw:
15             suma += text.split().count(k)
16         relation = round(suma/len(text.split()), 4)
17     else:
18         relation = 0
19     return relation
20
21 def fetch_sentiments(query, lng, kw):
...
28     m = np.max(ppt)
29     a = round(np.average(ppt), 4)
30     if m:
31         for item in data.values():
32             for k, i in enumerate(item):
33                 if i[2] > a:
34                     i[2] = 2
35                 else:
36                     i[2] = 1
37                 next = k + 1
38
39                 while next < len(item):
40                     if round(i[0], 1) == round(item[next][0], 1)
41                     and round(i[1], 1) == round(item[next][1],1):
42                         i[2] = i[2] + 0.2
43                         item.pop(next)
44                     else:
45                         next += 1
```

Código 7.19: Extracción de las coordenadas de los sentimientos

7.1.6 *Implementación de las Noticias*

La entidad de las noticias está compuesta por distintos valores comentados en la sección 5.2.2.5. La estructuración de las noticias resulta tan sencilla como las que se han implementado anteriormente. Aunque la implementación tuvo varios cambios importantes.

```
1 # PARSING OBJECT WRAPPED
2 def parse_wrapped(item):
3
4     date = item['date']
5     image = item['image']
6     summary = item['summary']
7     title = item['title']
8     url = item['url']
9     source = item['source']
10
```

```
    values = {
        'date': date,
        'image': image,
        'summary': summary,
        'title': title,
        'url': url,
        'source': source
    }

    return Wrapped(**values)
```

Código 7.20: Estructuración de las noticias

Al empezar a implementar el código, pensé que la manera más sencilla de obtener información de noticias sería mediante una API. Aunque al usar distintas APIs sobre noticias, como *News API* me di cuenta de que muchas noticias no estaban actualizadas, tenían errores en la descripción o el titulo y las APIs tenían por costumbre limitar drásticamente el número de peticiones por día.

Por ello tuve que cambiar el código y recurrir al scraping, una técnica utilizada para extraer información de sitios web. Para ello usé librerías como *BeautifulSoup* o *newspaper*, bibliotecas de Python para analizar documentos HTML y extraer de ellas información que resulte relevante.

Primero se descarga la búsqueda de noticias relevantes mediante el recurso XML de Google News. Esto se hace mediante los tópicos que se han extraído anteriormente y la biblioteca *BeautifulSoup*, de esta manera las noticias serán mucho más refinadas. Además, siempre se intenta conseguir tres noticias diferentes y actuales, limitando la búsqueda por fecha.

Otro punto diferencial en la herramienta scraping y la API, es que con la herramienta scraping se conseguían artículos muchos más actuales y a la vez diferentes. Esto es debido a que al descargar artículos mediante la API solamente se podía condicionar el parámetro de búsqueda mediante la relevancia o actualidad, por lo que muchos artículos se podían repetir o quedaban desfasados. Gracias al scraping se pueden conseguir artículos recientes y a la vez relevantes, combinado los parámetros de búsqueda en Google News. Además, el problema de los artículos repetidos quedaba solucionado al usar un buscador más sotisficado que el de la API News.

```
# GETTING ARTICLES
def get_everything(query, code, lang):
    # Getting the XML Content from Google News
    # An example: #https://news.google.com/rss/search?q=Riot+when
    :1d&hl=es&gl=ES&ceid=ES:es
    query = urllib.parse.quote(query.encode('utf8'))
```

```
6     BASE_URL = f"https://news.google.com/rss/search?q={query}&hl
      ={lang}&gl={code}&ceid={code}:{lang}"
7     xml_content = requests.get(BASE_URL).content
8     soup = BeautifulSoup(xml_content, features="xml")
9     g_items = soup.find_all("item")
10
11    if len(g_items) > 0:
12        # Google Articles with poor info and not parsed yet
13        g_articles = []
14        for item in g_items:
15            article = {}
16            article["link"] = item.find("link").text
17            article["title"] = item.find("title").text
18            article["publisher"] = item.find("source").text
19            article["published_date"] = item.find("pubDate").text
20            g_articles.append(article)
21
22        # Google with more info and parsed
23        articles: list[dict[str, str]] = []
24        cont = 0
25        while len(articles) != 3 and cont < len(g_articles):
26            article_info = get_article_info(g_articles[cont],
27                        lang)
28            if article_info:
29                articles.append(article_info)
30            cont += 1
31        if len(articles) == 3:
32            return articles
33
34    return False
```

Código 7.21: Extracción de las noticias

Al estar descargando información de Google News lo hacemos desde la ruta *rss*. *Really Simple Syndication*, «sindicación realmente simple» (RSS) contiene un resumen de las actualizaciones de un sitio web y está escrito en XML. Una de las etapas más importantes del análisis de archivos XML es la búsqueda de etiquetas. Hay varias formas de hacer esto, mediante nombres o relaciones.

Al estar en recopilando información de una página simple y estática, como son los artículos de Google News se ha utilizado la extracción de información mediante nombres. La función de *find* recibe el nombre de la etiqueta que desea obtener y devuelve un objeto BeautifulSoup de la etiqueta si encuentra uno, de lo contrario, devuelve *None*.

De esta manera se puede realizar un bucle de todos los artículos relacionados con la tendencia. Consiguiendo información importante, como su enlace, el título, la editorial y la fecha de publicación.

Gracias a los artículos recopilados y sus enlaces correspondientes, podemos proseguir consiguiendo información relevante como una imagen de portada o la descripción.

```
# GETTING ARTICLES
def get_article_info(item, lang):
    user_agent = 'Mozilla/5.0 (Macintosh; Intel Mac OS X 10.15;
    rv:78.0) Gecko/20100101 Firefox/78.0'
    config = Config()
    config.browser_user_agent = user_agent
    config.request_timeout = 100
    config.MAX_TEXT = 500
    config.language = lang
    config.number_threads = 100
    a = Article(item["link"], language=lang, config=config)

    try:
        a.download()
        a.parse()
        date = parser.parse(item["published_date"])
        date_parsed = date.strftime("Date: %y-%m-%d Time: %H:%M")
        values = {
            'date': date_parsed,
            'image': a.top_image,
            'summary': a.text,
            'title': a.title,
            'url': item["link"],
            'source': item["publisher"]
        }
        if values["title"] and values["summary"]
        and len(values["summary"].split()) > 20
        and values["title"] != "Are you a robot?":
            return values
        else:
            return False

    except ArticleException as ae:
        print(ae)
        return False

    except Exception as e:
        print(e)
        return False
```

Código 7.22: Extracción de la información de las noticias

En esta sección de código, en vez de usar la biblioteca BeautifulSoup, se ha utilizado *Newspaper3k*. Las ventajas de ello es que, la biblioteca Newspaper3k también puede realizar funciones más avanzadas, como descubrir fuentes RSS, buscar URL de artículos de una fuente de noticias principal e incluso extraer varios subprocesos si

tiene que buscar más de un artículo. Además, termina siendo más conveniente al extraer información de sitios web diferentes.

Se ha establecido un gran tiempo para la conexión, para asegurar la descarga del articulo y su información, debido a esto el proceso de recopilación de datos sobre la tendencia resulta muy costoso. Aunque gracias al multiprocesamiento, se ha podido realizar las descargas de manera no secuencial y aligerar el proceso.

Después de añadir otras configuraciones como un *header* para poder acceder a la página web sin limitaciones, se recopilan los diferentes datos y se estructuran de una manera conveniente. Además, pasan por validación por si la información recopilada no tiene sentido, en este caso la validación es bastante simple aunque efectiva. La descripción debe tener al menos 20 palabras, debe existir un título y que no sea «Are you a robot?». La última validación es debido a que algunas páginas web comprueban cuidadosamente las peticiones que se hacen por estos métodos y devuelven esa información, en vez del articulo.

7.1.7 *Implementación del Servidor*

Anteriormente se ha descrito las diferentes implementaciones sobre las entidades, su estructuración y extracción de datos. Dichas implementaciones se encuentran en la carpeta *app*, mientras que todo lo relacionado con el servicio del servidor se encuentra en el directorio *backend*. No entraré en mucho detalle en la implementación de las clases o rutas, ya que se han explicado en la sección de diseño (5). Aunque interferiré en el modelo *Country* y su obtención.

```
1  class Country(BaseModel):
2      id: str = Field(default_factory=uuid.uuid4, alias="_id")
3      pais: str = Field(...)
4      dia: str = Field(...)
5      tendencias: List[Trend] = Field(...)
6
7      class Config:
8          allow_population_by_field_name = True
9          schema_extra = {
10             "example": {
11                 "_id": "066de609-b04a-4b30-b46c-32537c7f1f6e",
12                 "pais": "Spain",
13
...
1
2  @router.get("/{name}/{date}", response_description=
3      "Get a single country by the name and date",
4      response_model=Country)
5  def find_country(name: str, date: str, request: Request):
```

```
6     if (country := request.app.database["countries"].find_one(
7     {"pais": name, "dia": date})) is not None:
8         country["tendencias"] = list(
9             filter(itemgetter("wrappeds"),
10            country["tendencias"]))[:10]
11        return country
12
13    raise HTTPException(status_code=status.HTTP_404_NOT_FOUND,
14                        detail=f"Country with the name {name}
15                        and date {date} not found")
```

Código 7.23: Modelo *Country* y ejemplo de ruta

La ruta devuelve un objeto bajo el modelo de *Country* mediante la fecha y el nombre del país, tal y como se ha explicado en la sección de diseño de interacción de rutas (diagrama 5.14). Todas las demás rutas y entidades tienen un comportamiento parecido. La peculiaridad de esta ruta es que filtra diez tendencias que tengan el objeto de noticias o *wrappeds*, para no mostrar tendencias vacías.

7.2 IMPLEMENTACIÓN FRONT-END

La implementación de la capa de *Front-end* se localiza bajo el directorio *front-end*, la estructura de la aplicación de Vue sigue las buenas prácticas conteniendo archivos correspondientes en cada directorio relevante a la aplicación, es decir, se han separado en *assets*, *components*, *router*, *styles* y la carpeta *public*.

7.2.1 *Implementación Principal*

La primera implementación de código que se estudiará será la vista principal descrita anteriormente en la sección 5.4.3. Cabe mencionar que no se podrá estudiar todo el código, debido a que incluir todas las reglas de diseño CSS y la estructura HTML complicarían la lectura y su propia comprensión. Aunque indagaré en las diferencias más destacables.

La estructura de un archivo Vue se compone de tres apartados esenciales, el primer apartado llamado *template* contiene la implementación HTML, el segundo llamado *script* contendrá implementaciones de JavaScript y por último *style* incluirá implementaciones de diseño.

```
1   <template>
...
121         <div class="flex overflow-x-auto gap-6 snap-x snap-
    mandatory sm:snap-normal before:shrink-0 before:w-[20%] sm:
    before:w-[0%] after:shrink-0 after:w-[20%] sm:after:w-[0%] sm
    :horizontal sm:pb-14 pb-5">
122             <div
```

```
123                 v-if="articulos.popularity"
124                 class="basis-4/5 sm:basis-3/12 xl:basis-2/12 shrink
    -0 snap-center sm:snap-align-none shadowdrop"
125               >
126                 <GraphViewer
127                   :array-popularity="articulos.popularity.volume"
128                   :array-hours="articulos.popularity.
    hour_popularity"
129                   :avarage-popularity="articulos.popularity.
    avarage_popularity"
130                   :peak-popularity="articulos.popularity.
    peak_popularity"
131                   :num="1"
132                   :rank="index+1"
133                   :url="encondeURL(articulos.name)"
134                 />
135               </div>
...
164 </template>
165
166 <script>
167 import axios from 'axios';
...
271 export default {
272   components: {
273     CardViewer,
274     GraphViewer,
275     MapViewer,
276     BubbleViewer,
277   },
...
411 </script>
412
413 <style>
...
426 .truncador-title {
427   overflow: hidden;
428   display: -webkit-box;
429   -webkit-box-orient: vertical;
430   -webkit-line-clamp: 2;
431   -webkit-margin-collapse: discard;
432 }
...
571 </style>
```

Código 7.24: Secciones de Vue

En la sección *template* de esta vista se encuentra toda la estructura HTML de la página y llamadas a otros componentes. Gracias a funciones Vue se ha podido implementar bucles o funciones directamente en dicha estructura. Un ejemplo de ello es el *v-if* o *v-for* los cuales renderizan contenido mediante condiciones o listas. La renderización

de listas fue algo esencial a la hora de renderizar un JSON devuelto de la base de datos.

Luego, en la sección de *script* se implementa la parte programable. En el caso de la vista principal se realizan llamadas a la base de datos justo en la creación de la página web, esto es debido a que el contenido se cargue justo al inicio de la creación de la página web, sin tener que esperar a la carga de todos los componentes necesarios como se puede ver en el esquema posterior.

Figura 7.4: Ciclo de vida de una aplicación Vue [58].

El código mostrado es una versión simplificada, con el objetivo de facilitar la comprensión. Lo importante del código es que se ha implementado un manejo de errores y la creación de contenido de la página dependiendo de la URL que se haya introducido.

```
1  async created() {
2      await this.fetch();
3  },
4  methods: {
5      async fetch() {
6        this.loading = true;
7        try {
8          let data = {};
9          if (this.route.query.name and this.route.query.query) {
10           data = await axios.get((`{SEARCHTRENDINGENDPOINT}`));
11         } else { data = await axios.get((`{TRENDINGENDPOINT`));}
12         if (data.status === 200 and data.data.
13             tendencias[0].wrappeds) {
14           this.wrapped = data.data.tendencias;
15           this.loading = false;
16           this.error = false;
17         } else {
18           this.country = 'Spain';
19           this.date = this.dateString(new Date());
20           this.alert('There was an error trying to load (...)');
21         }
22       } catch (error) {
23         this.country = 'Spain';
24         this.date = this.dateString(new Date());
25         this.alert('Right now it looks like the web is (...)');
26         console.log(error);
27       }
28     },
29 },
```

Código 7.25: Funciones de carga de contenido en Vue

Otra sección a destacar el el diseño. La implementación de esta sección tuvo el mismo o incluso más arduo trabajo que las demás, primero al querer tener un diseño original y que funcionase en todos los dispositivos, se han implementado muchísimas reglas en CSS, entre ellas reglas de medida o consultas de medio para cambiar toda la visualización de contenido dependiendo de la medida del dispositivo.

Esto es debido a que se ha querido que se viera de una manera óptima sin sacrificar la visibilidad del contenido. Por ello se ha recurrido a herramientas como el truncamiento en CSS. Luego otro de los detalles del diseño es que las sombras varían dependiendo del navegador, debido al rendimiento que tienen sobre estos. El navegador Chrome llega a funcionar muy bien con la opción *filter()* de CSS, pero sin embargo Safari no. También hay que tener en cuenta las personas que tengan el zoom activado en el teléfono y adaptar también su contenido, incluso pantallas gigantes o extremadamente pequeñas, todo esto hace que el diseño pueda ser debidamente *responsive*.

Otro detalle, a parte de las sombras, es que, en los dispositivos no móviles, se han implementado animaciones de *hover*. Estas animaciones levantan y oscurecen ligeramente la carta o módulo cada vez que el ratón pasa por encima, facilitando así su visualización.

También se han seguido las buenas prácticas y se ha respetado el contraste de los colores para la una lectura más fácil de los enlaces web o el propio contenido de la página. Además, se han usado varios *svg* con formas de ola, para que la página sea más agradable a la vista.

```
1  @media (min-height: 550px) {
2    contenido {
3      height: 10rem;
4    }
5
6    .truncador {
7      overflow: hidden;
8      display: -webkit-box;
9      -webkit-box-orient: vertical;
10     -webkit-line-clamp: 5;
11     -webkit-margin-collapse: discard;
12   }
...
38 }
39
40 /* Safari 10.1+ */
41 @media not all and (min-resolution:.001dpcm) { @media {
42   .shadowdrop {
43     filter: none;
44     -webkit-filter: none;
45     -moz-filter:  none;
46     -ms-filter:  none;
47     -o-filter: none;
48     box-shadow: rgba(0, 0, 0, 0.03) 0px 15px 7px -7px;
49   }
50 }}
```

Código 7.26: Detalles de la implementación del diseño

7.2.2 *Implementación de la Popularidad*

El código implementado en esta sección se refiere a la subvista popularidad explicada en la sección 5.4.4. Las principales diferencias de implementación en este módulo es la implementación de diferentes funciones en el *template* de la vista y también la propia implementación del gráfico, ya que es una librería que requiere mucha personalización para poderse visualizar adecuadamente.

```
1  <ul class="text-sm contenido-stats text-ellipsis overflow-hidden"
     >
2    <li class="py-0.5 pt-1">
3      - First data at {{ encondeHour(arrayHours[0]) }}.
4    </li>
5    <li class="py-0.5">
6      - Last data at
7      {{ encondeHour(arrayHours[arrayHours.length - 1]) }}.
8    </li>
9    <li class="py-0.5">
10     - Peak popularity is {{ peakPopularity.toLocaleString('en-US'
       ) }}.
11   </li>
12   <li class="py-0.5 pb-1">
13     - Average popularity is {{ avaragePopularity.toLocaleString('
       en-US') }}.
14   </li>
15 </ul>
...
63 chartOptions: {
64     chart: {
65       type: 'area',
66       toolbar: {
67         show: false,
68       },
69       zoom: {
70         enabled: false,
71       },
72       animations: { enabled: false },
73     },
74     xaxis: {
75       type: 'datetime',
76       categories: this.arrayHours,
77       labels: {
78         format: 'HH:mm',
79         },
...
95         markers: {
96           size: 3,
97         },
98       },
```

Código 7.27: Diferencias destacables de la implementación de la popularidad

Las horas en este caso se guardan en formato JavaScript y se transforman en objeto *date* al implementarlo en el código Vue. Además, se ha respetado las horas locales de cada país como se puede ver en el código.

En el siguiente ejemplo de visualización, se ha usado el navegador de Chrome bajo la premisa de simular la vista de un iPhone SE (un

teléfono bastante compacto). El contenido que se puede visualizar es el descenso de popularidad que tuvo la tendencia *Joji*. Podemos ver cómo empezó siendo popular sobre la madrugada, pero que poco a poco fue decayendo. En el día de 5 de noviembre, se encuentra en la octava posición del top 10 tendencias más populares de España. El valor del volumen de publicaciones, es decir el número de veces que se ha mencionado la tendencia fue de 86,748 como máximo y 59,957 de media.

Figura 7.5: Ejemplo de tendencia, vista de popularidad.

7.2.3 *Implementación de las Keywords*

El código implementado en esta sección se refiere a la subvista *keywords* explicada en la sección 5.4.5. En cuanto a las principales dife-

rencias, es la cantidad de código que ha requerido la implementación de la personalización del gráfico.

```
series: this.arrayCount,
chartOptions: {
chart: {
  type: 'radialBar',
  toolbar: {
    show: false,
  },
  animations: { enabled: false },
  sparkline: {
    enabled: true,
  },
},
colors: [
  '#008FFB',
  '#00E396',
  '#FEB019',
  '#FF4560',
  '#775DD0',
  '#5DABD0',
],
plotOptions: {
  radialBar: {
    offsetY: -5,
    startAngle: 0,
    endAngle: 360,
    hollow: {
      margin: 5,
      size: '0%',
      background: 'transparent',
    },

...
```

Código 7.28: Diferencias destacables de la implementación de las keywords

Los colores y sombras han sido definidos manualmente, al igual que la posición del gráfico y el etiquetado de sus datos.

En el siguiente ejemplo podemos visualizar el contexto de la tendencia. Obviamente el nombre de la tendencia es la palabra que más se repite en las publicaciones, se ha repetido 99 veces en los 100 tweets recopilados. Para entender la situación podemos ver las palabras más repetidas y los *keywords*, la información que nos dan es «Album De Joji», «Die For You» y «Nuevo Disco». Al parecer Joji es un cantante o al menos una persona famosa, que ha sacado un disco o álbum nuevo llamado "Die For You".

Figura 7.6: Ejemplo de tendencia, vista de *keywords*.

7.2.4 *Implementación de los Sentimientos*

El código implementado en esta sección se refiere a la subvista sentimientos explicada en la sección 5.4.6. Aquí también destaca la implementación la personalización del gráfico, aunque también hay funciones que requieren explicación.

Todos los ejes, el marco o las propias burbujas han tenido su implementación de posición, color y grosor de trazado. Además, el lienzo en el que se pinta es algo más grande que la notación del etiquetado, esto es debido a que se pretende que se visualice mejor la gráfica y no se corten las burbujas.

```
1 xaxis: {
```

```
2     min: -1,
3     max: 1,
4     tickAmount: 1,
5     type: 'numeric',
6     position: 'bottom',
7     decimalsInFloat: 0,
8     labels: {
9       offsetY: -4,
10      style: {
11        fontSize: '11px',
12      },
13    },
14    axisBorder: {
15      show: false,
16    },
17    axisTicks: {
18      show: false,
19    },
20    title: {
21      text: 'Compound',
22      offsetY: -22,
23      style: {
24        fontSize: '12px',
25        fontWeight: 350,
26      },
27    },
28  },
29
...
133 percentageSentiment(array, arrayA, arrayB) {
134   let suma = 0;
135   let val = 0;
136   array.forEach((item) => {
137     suma += item[2];
138   });
139   val = suma;
140   arrayA.forEach((item) => {
141     suma += item[2];
142   });
143   arrayB.forEach((item) => {
144     suma += item[2];
145   });
146   return (Math.round((val / suma) * 100));
147 },
```

Código 7.29: Diferencias destacables de la implementación de los sentimientos

El porcentaje se calcula en torno al radio de la burbuja, ya que no tiene sentido calcular la cantidad de burbujas porque como se ha explicado en la sección de su implementación (7.19) se eliminan las que tienen coordenadas muy similares.

En el ejemplo siguiente, del cantante Joji, la mayoría de las publicaciones tienen que ser positivas debido a los seguidores de esta persona, al comprobarlo vemos que efectivamente el 56 % son publicaciones positivas. También puede haber publicaciones negativas o neutras, tal como análisis de críticos que no expresan una opinión o personas a las que no les ha gustado. Claramente también puede haber errores, ya que la extracción del sentimiento no es cien por cien precisa, aunque se puede ver que el sentimiento global es acertado.

Figura 7.7: Ejemplo de tendencia, vista de sentimientos.

También se puede comprobar por tweets o publicaciones específicas, ya que cada burbuja es una:

> *"lo siento jefe hoy no puedo trabajar el album de joji me ha puesto triste"*

Puntuación: -0.7061

Podemos ver y comprobar la publicación en la propia plataforma.

Figura 7.8: Ejemplo de tweet (1).

> *"hoy es un buen dia porque joji saco un nuevo album con musica muy buena, asi que, buenos dias; escuchen a joji, es arte."*

Puntuación: 0.902

El sentimiento en este caso también es justificado. Podemos ver y comprobar la publicación en la propia plataforma.

Figura 7.9: Ejemplo de tweet (2).

> *"toca llorar toda la noche escuchando el nuevo album de joji en loop"*

Puntuación: -0.34

El sentimiento en este caso también es justificado, aunque obviamente es una publicación sarcástica, algo que la herramienta de extracción no entendería.

Lo bueno de comprobar estas publicaciones que las imágenes u otros elementos como los *hashtags* no interfieren en el análisis de sentimientos. Además, se recogen publicaciones con mucha interacción, tal y como lo hemos especificado en el algoritmo.

Figura 7.10: Ejemplo de tweet (3).

7.2.5 *Implementación de las Noticias*

El código implementado en esta sección se refiere a la subvista *keywords* explicada en la sección 5.4.7. En cuanto a las principales diferencias, no destaca mucho en la implementación, ya que solo tiene

que mostrar valores proporcionados de la página web, pero destaca en el diseño.

Las animaciones y otros detalles parten de esta vista. También existe una función que, ante un fallo de carga de imagen, provee una imagen por defecto para las noticias.

```
1   import img from '../assets/img_news.jpg';
2   export default {
3     methods: {
4       replaceByDefault(e) {
5         e.target.src = img;
6       },
7     },
8
...
123 @media (hover: hover) and (pointer: fine) {
124   .animation:hover {
125     --tw-translate-y: -0.5rem /* -8px */;
126     transform: translate(var(--tw-translate-x), var(--tw-
      translate-y))
127       rotate(var(--tw-rotate)) skewX(var(--tw-skew-x)) skewY(var
      (--tw-skew-y))
128       scaleX(var(--tw-scale-x)) scaleY(var(--tw-scale-y));
129   }
130   .animation-shadow:hover {
131     --tw-shadow: 0 20px 25px -12px rgb(0 0 0 / 0.35);
132     --tw-shadow-colored: 0 20px 25px -12px var(--tw-shadow-color)
      ;
133     box-shadow: var(--tw-ring-offset-shadow, 0 0 #0000),
134       var(--tw-ring-shadow, 0 0 #0000), var(--tw-shadow);
135   }
136 }
```

Código 7.30: Diferencias destacables de la implementación de las noticas

Hay tres noticias bastante parecidas. Las tres anuncian el nuevo álbum, a lo mejor también información sobre sus conciertos o donde poder escucharlo.

Figura 7.11: Ejemplo de tendencia, vista de noticias.

La noticia presenta un titulo, descripción y fotografía explicativa. La fotografía es del cantante Joji. Se puede acceder a dicho articulo mediante el enlace de la editorial o directamente pulsando sobre la foto.

Figura 7.12: Ejemplo de tendencia, vista de noticias por defecto.

La búsqueda por tópicos funcionaria de la misma manera. Como ultima mención, la implementación de errores fue implementada tal cual descrita en la sección 5.25. El mensaje de la alerta cambia dependiendo del error ocurrido, para que el usuario tenga conciencia de cuál fue el problema.

8 DEDUCCIONES Y RESULTADOS

8.1 CONCLUSIONES Y PRUEBAS DE PRESTACIONES

Este trabajo, aunque solamente tenga un objetivo, el cual es enseñar una pequeña visión del mundo en un instante dado, ha demostrado aparte de cumplir dicho objetivo, ser una aplicación rígida y estar construida de una manera actual.

Aparte de haber demostrado la capacidad de poder programar el funcionamiento de la aplicación, se ha demostrado la competencia en la construcción del planteamiento, de la arquitectura y la propia estructura de la aplicación. Todo ello gracias al aprendizaje llevado a cabo en el grado de ingeniería de informática.

Aunque al hacer diferentes pruebas de prestaciones con herramientas como JMeter o benchmarks de distintas páginas web, me he podido dar cuenta de las limitaciones que ha supuesto usar distintas herramientas como *framework* Vue o las plataformas gratuitas de servicio.

Las plataformas, al ser gratuitas funcionan con más limitaciones. La latencia puede llegar a ser más grande ya que te limitan el uso a los servidores gratuitos y además los servidores están separados en distintas máquinas, cada una conectada en una región diferente.

Al estar usando Vue me pude dar cuenta de las limitaciones que supone usar un DOM virtual, al tener mucha carga de JavaScript, debido a los diferentes gráficos, el tiempo de la carga de la página se hace mayor. Esto no pasaría en un *framework* estático como Svelte, aunque otra manera de arreglarlo sería construir componentes *skeleton* o esqueleto, los cuales cargan la estructura de toda la página y solo cambian los componentes necesarios. Intenté implementar la segunda opción, pero he tenido bastantes fallos debido a la librería de áreas y gráficas, y porque dicho componente esqueleto sigue aún en fase de beta.

El escenario supuesto de la herramienta de Jmeter consiste en analizar el rendimiento de la página donde 5 usuarios interaccionan durante un intervalo de tiempo predeterminado, en este caso ocho minutos.

Figura 8.1: Prueba de prestaciones con Jmeter.

Al analizar el rendimiento nos damos cuenta de varias cosas que ya hemos comentado. Primero, hay que tener en cuenta que los errores son debido a la carga de contenido externo, en este caso imágenes de los artículos de las noticias. Esto se puede arreglar si descargamos las imágenes en vez de cargar el contenido de fuera. En general el rendimiento de la página es aceptable para la carga proporcionada, aunque el rendimiento puede llegar a ser cuestionable debido a la cantidad de JavaScript proporcionado por la visualización de las gráficas.

Podemos ver también otros benchmarks que analicen nuestra página web. En este caso he elegido *PageSpeed Insights*. En este benchmark, se puede comprobar cómo se han respetado las buenas prácticas, la accesibilidad y el SEO (parámetros de buscadores). Aunque podemos ver que el rendimiento decae debido a los problemas comentados anteriormente (latencia y JavaScript).

Figura 8.2: Prueba de prestaciones con *PageSpeed Insights*.

8.2 TRABAJOS FUTUROS

Como mejora del producto, creo que se puede seguir perfeccionando el contenido de la propia aplicación. Un ejemplo de ello puede ser añadiendo más países o diversificando el contenido, añadiendo vídeos como otro medio de visualización de la tendencia. Los vídeos pueden pertenecer tanto a la propia plataforma de Twitter o a plataformas como YouTube o Vimeo.

Otra mejora a tener en cuenta es la monetización, esta se puede llegar a integrar como un módulo o carta dentro de la propia aplicación (algo que tenga relación con la tendencia) o directamente cobrar por cada redirección a la noticia u otro enlace que pueda ser monetizado.

Aunque, a mi parecer, las mejoras más importantes a tener en cuenta son las que están relacionadas con la elaboración del proyecto. Se puede seguir mejorando la arquitectura de la aplicación, hacer que sea más escalable y buscar otras vías que no sean las plataformas de servicio PaaS gratuitas, de esta manera se mejoraría notablemente la latencia.

Otra cuestión de importancia, es mejorar el rendimiento en la interfaz de usuario. En la sección anterior se ha comentado la posibilidad de cambiar de *framework*, se podrían hacer pruebas de rendimiento o prestaciones con distintos *frameworks* que tengan el DOM estático (como Svelte) o virtual (como Vue o React), de esta manera comprobar cuál es más eficiente con mi aplicación. Aunque se hayan hecho estudios técnicos sobre distintos *frameworks*, no es lo mismo que ver cómo se comporta mi aplicación en cada uno de ellos.

Incluso se podría comparar el rendimiento con otra librería de gráficos y áreas. También se ha comentado la posibilidad de incorporar vistas de esqueletos a la aplicación, aligerando así la carga de contenido cambiante. Incluso se podría incorporar algún tipo de paginación, de manera que solo se cargue el contenido que vea el usuario.

Como última aportación creo que es importante perfeccionar primero o anteponer las posibles mejoras comentadas sobre la estructura de la aplicación, que seguir agregando contenido nuevo o formas de monetización. De esta manera conseguiríamos una aplicación mucho más robusta y escalable, la cual no nos daría problemas en el futuro a la hora de agregar implementaciones nuevas.

BIBLIOGRAFÍA

[1] Belén Villa. «Estudio de Redes Sociales 2022 de IAB Spain». En: *Studio Aloha* (2022).

[2] Brainstation. «ADDICTION: Study Finds Social Media to Be Like Smoking Cigarettes». En: *Digital* (2012).

[3] Hilary Andersson. «Social media apps are 'deliberately' addictive to users». En: *BBC Panorama* (2018).

[4] José Luis Orihuela-Colliva. «Internet: la hora de las redes sociales». En: (2008).

[5] Blake Morgan. «50 Stats Showing The Power Of Personalization». En: *Forbes* (2020).

[6] Montag C., Lachmann B., Herrlich M. y Zweig K. «Addictive Features of Social Media/Messenger Platforms and Freemium Games against the Background of Psychological and Economic Theories». En: *National Library of Medicine* (2019). URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6679162/#B67-ijerph-16-02612.

[7] Manuel Moreno. «10 estadísticas sobre WhatsApp que quizá no sabías - TreceBits». En: *TreceBits* (2022).

[8] Jared Spool. «Do users change their settings?» En: *Archive UIE* (2011).

[9] Orbium. «El fenómeno FOMO: fear of missing out». En: *Orbium* (2020).

[10] Laura Montells. «Qué es Edgerank o cómo funciona el algoritmo de Facebook». En: *Metricool* (2021).

[11] Chema Carvajal. «El algoritmo de Facebook te muestra contenido que odias para que interacciones más». En: *ComputerHoy* (2021).

[12] Christian Montag y col. «Facebook usage on smartphones and gray matter volume of the nucleus accumbens». En: *Behavioural Brain Research* 329 (2017), págs. 221-228. ISSN: 0166-4328. DOI: https://doi.org/10.1016/j.bbr.2017.04.035. URL: https://www.sciencedirect.com/science/article/pii/S0166432817305090.

[13] Lauren E. Sherman, Patricia M Greenfield, Leanna M Hernandez y Mirella Dapretto. «Peer influence via instagram: Effects on brain and behavior in adolescence and young adulthood». En: *Child development* 89.1 (2018), págs. 37-47.

[14] Javier Robledo Vico. «Así afectan las redes sociales a nuestro cerebro». En: *AS* (2017).

[15] Bahena-Trujillo, Ricardo, Flores, Gonzalo, Arias-Montaño y José A. «Dopamina: síntesis, liberación y receptores en el Sistema Nervioso Central». En: *Revista Biomédica* 11.1 (2000), págs. 39-60.

[16] Santiago Campillo. «Lo que pasa en nuestro cerebro cuando recibimos una notificación, según la ciencia». En: *Xataka* (2019).

[17] Quintero-Cacique, Juan Camilo, Lesmes-Silva y Anggy Karina. «¿ Por qué el ser humano es adicto a las redes sociales?» En: *Revista CONVICCIONES* 8.15 (2021), págs. 6-9.

[18] Krach S., Paulus F.M., Bodden M. y Kircher T. «The rewarding nature of social interactions». En: *Frontiers in behavioral neuroscience* (2010).

[19] Jairo Rozo Castillo. «Pavlov y los reflejos condicionados: la verdadera historia sobre un descubrimiento». En: *PsicologiaCientifica* (2015).

[20] Raquel Rodríguez Cortés. «Adicción a las redes sociales: síntomas y consecuencias». En: *Psiquion* (2021).

[21] Trevor Wheelwright. «2022 Cell Phone Usage Statistics: How Obsessed Are We?» En: *Reviews* (2022).

[22] Sergio Parra. «La razón de que sea tan fácil volvernos adictos a contenidos digitales». En: *MuyInteresante* (2020).

[23] Portillo-Reyes, J. A. V. Ávila Amaya y J. W. Capps. «Relación del Uso de Redes Socia-les con la Autoestima y la Ansiedad en Estudiantes Universitarios». En: *Enseñanza e Investigación en Psicología* (2021).

[24] Talent. *Salario medio para Programador Web en España, 2022*. Inf. téc. Talent, 2022.

[25] Oscar Tinoco Gómez, Pedro Pablo Rosales López y Julio Salas Bacalla. «Criterios de selección de metodologías de desarrollo de software». En: *Industrial data* 13.2 (2010), págs. 70-74.

[26] José H Canós, Patricio Letelier y Mª Carmen Penadés. «Metodologías ágiles en el desarrollo de software». En: *Universidad Politécnica de Valencia, Valencia* (2003), págs. 1-8.

[27] Esteban Gabriel Maida y Julián Pacienzia. «Metodologías de desarrollo de software». En: (2015).

[28] César Rodríguez y Rubén Dorado. «¿ Por qué implementar Scrum?» En: *Revista Ontare* 3.1 (2015), págs. 125-144.

[29] Nader K Rad y Frank Turley. *Los Fundamentos de Agile Scrum*. Van Haren, 2019.

[30] John Hogue y Burton DeWilde. *Libreria Pytrends*. URL: https://pypi.org/project/pytrends/.

[31] Joshua Roesslein. *Libreria Tweepy*. URL: https://docs.tweepy.org/en/latest/index.html.

[32] J Pelaez. «Arquitectura basada en Componentes. Extractado de La Guía de Arquitectura Versión 2.0a del grupo de Patterns and Practices de Microsoft.» En: ().

[33] MongoDB. *Why Use MongoDB and When to Use It?* URL: https://www.mongodb.com/why-use-mongodb.

[34] Pydantic. *Pydantic Documentation*. URL: https://pydantic-docs.helpmanual.io.

[35] Red Hat. *What is a REST API?* URL: https://www.redhat.com/es/topics/api/what-is-a-rest-api.

[36] Uvicorn. *Uvicorn Documentation*. URL: https://www.uvicorn.org.

[37] FastAPI. *FastAPI Documentation*. URL: https://fastapi.tiangolo.com.

[38] Heroku. *Heroku Documentation*. URL: https://devcenter.heroku.com/articles/getting-started-with-python.

[39] Maja Nowak. «Vue vs React in 2022 - Comparison of Two Most Popular JS Frameworks.» En: (2022).

[40] Red Hat. *El concepto de DevOps*. URL: https://www.redhat.com/es/topics/devops.

[41] Angel Pérez. «Azure DevOps + Herramientas de Comunicación». En: (2020).

[42] iLimit. «DevOps». En: (2020).

[43] GitHub. *GitHub Documentation*. URL: https://docs.github.com/es.

[44] Poetry. *Poetry Documentation*. URL: https://python-poetry.org/docs/.

[45] Mypy. *Mypy Documentation*. URL: https://mypy.readthedocs.io/en/stable/index.html.

[46] Flake8. *Flake8 Documentation*. URL: https://flake8.pycqa.org/en/latest/.

[47] Pytest. *Pytest Documentation*. URL: https://docs.pytest.org/en/7.2.x/.

[48] Tox. *Tox Documentation*. URL: https://tox.wiki/en/latest/.

[49] Npm. *Npm Documentation*. URL: https://docs.npmjs.com.

[50] ESLint. *ESLint Documentation*. URL: https://eslint.org/docs/latest/.

[51] Ricardo Campos, Vítor Mangaravite, Arian Pasquali, Alípio Mário Jorge, Célia Nunes y Adam Jatowt. *YAKE! Collection-Independent Automatic Keyword Extractor*. Ed. por Gabriella Pasi, Benjamin Piwowarski, Leif Azzopardi y Allan Hanbury. Cham: Springer International Publishing, 2018, págs. 806-810. ISBN: 978-3-319-76941-7.

[52] Vladimir I Levenshtein y col. «Binary codes capable of correcting deletions, insertions, and reversals». En: *Soviet physics doklady*. Vol. 10. 8. Soviet Union. 1966, págs. 707-710.

[53] Towards AI. «TextBlob vs. VADER for Sentiment Analysis Using Python». En: (2021).

[54] VADER. *VADER Manual*. URL: https://github.com/cjhutto/vaderSentiment.

[55] Saralegi X. y San Vicente I. «XXIX Congreso de la Sociedad Española de Procesamiento de lenguaje natural». En: *Elhuyar at TASS 2013* 978-84-695-8349-4 (2013), págs. 143-150.

[56] skupriienko. *Ukrainian-Sentiment-Analysis*. URL: https://github.com/skupriienko/Ukrainian-Sentiment-Analysis.

[57] Pio Calderon. «VADER Sentiment Analysis Explained». En: *Medium* (2017).

[58] Vue. *Lifecycle Hooks*. URL: https://vuejs.org/guide/essentials/lifecycle.html.

COLOFÓN

This document was typeset using the typographical look-and-feel `classicthesis` developed by André Miede and Ivo Pletikosić. The style was inspired by Robert Bringhurst's seminal book on typography "*The Elements of Typographic Style*". `classicthesis` is available for both LaTeX and LyX:

https://bitbucket.org/amiede/classicthesis/

Happy users of `classicthesis` usually send a real postcard to the author, a collection of postcards received so far is featured here:

http://postcards.miede.de/

Thank you very much for your feedback and contribution.

Final Version as of 9 de noviembre de 2022 (`classicthesis` Version 0.1).